In [614…

```
import numpy as np
from numpy import nan
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import time
import warnings
warnings.filterwarnings("ignore")
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
#from sklearn.feature_extraction.text import CountVectorizer  #DT does not take strings as input for the model fit step....
from IPython.display import Image
#import pydotplus as pydot
from scipy.stats import zscore
from sklearn import tree
from os import system
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import BaggingClassifier,RandomForestClassifier
from sklearn.ensemble import AdaBoostClassifier,RandomForestClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.linear_model import Lasso
from sklearn.svm import OneClassSVM
from sklearn.metrics import classification_report, confusion_matrix
from sklearn.ensemble import GradientBoostingClassifier,RandomForestClassifier
from sklearn import metrics
from sklearn.model_selection import GridSearchCV
%matplotlib inline
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.cluster import KMeans
from scipy.spatial.distance import cdist
from sklearn.decomposition import PCA
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import LeaveOneOut
from sklearn.metrics import confusion_matrix
from imblearn.over_sampling import SMOTE
from sklearn.neighbors import KNeighborsClassifier
plt.style.use('fivethirtyeight')
```

- **DOMAIN:**

- Semiconductor manufacturing process

• CONTEXT:

- A complex modern semiconductor manufacturing process is normally under constant surveillance via the monitoring of signals/variables collected from sensors and or process measurement points. However, not all of these signals are equally valuable in a specific monitoring system. The measured signals contain a combination of useful information, irrelevant information as well as noise.Engineers typically have a much larger number of signals than are actually required. If we consider each type of signal as a feature, then feature selection may be applied to identify the most relevant signals. The Process Engineers may then use these signals to determine key factors contributing to yield excursions downstream in the process. This will enable an increase in process throughput, decreased time to learning and reduce the per unit production costs. These signals can be used as features to predict the yield type. And by analysing and trying out different combinations of features, essential signals that are impacting the yield type can be identified.

• DATA DESCRIPTION: signal-data.csv : (1567, 592)

- The data consists of 1567 datapoints each with 591 features. The dataset presented in this case represents a selection of such features where each example represents a single production entity with associated measured features and the labels represent a simple pass/fail yield for in house line testing. Target column " –1" corresponds to a pass and "1" corresponds to a fail and the data time stamp is for that specific test point.

• PROJECT OBJECTIVE:

- We will build a classifier to predict the Pass/Fail yield of a particular process entity and analyse whether all the features are required to build the model or not.

Steps and tasks: [Total Score: 60 points]

1. Import and understand the data. [5 Marks]

1 A. Import 'signal-data.csv' as DataFrame. [2 Marks]

In [615…
```
# creating variable name for dataframe name df and load csv file.
df=pd.read_csv("C:\\Users\\CHIRAG\\Desktop\\Greatlearning- Projects\\AIML project - Featurization and model selection tuinning\\s
```

In [616…
```
print("The total number of observation is = ",df.shape[0],"\nAnd total number of column is = ",df.shape[1])
```

```
The total number of observation is =  1567
And total number of column is =  592
```

In [617…
```
# view first five rows
df.head()
```

Out[617]:

	Time	0	1	2	3	4	5	6	7	8	...	581	582	583	584	585	586	587
0	2008-07-19 11:55:00	3030.93	2564.00	2187.7333	1411.1265	1.3602	100.0	97.6133	0.1242	1.5005	...	NaN	0.5005	0.0118	0.0035	2.3630	NaN	NaN
1	2008-07-19 12:32:00	3095.78	2465.14	2230.4222	1463.6606	0.8294	100.0	102.3433	0.1247	1.4966	...	208.2045	0.5019	0.0223	0.0055	4.4447	0.0096	0.0201
2	2008-07-19 13:17:00	2932.61	2559.94	2186.4111	1698.0172	1.5102	100.0	95.4878	0.1241	1.4436	...	82.8602	0.4958	0.0157	0.0039	3.1745	0.0584	0.0484
3	2008-07-19 14:43:00	2988.72	2479.90	2199.0333	909.7926	1.3204	100.0	104.2367	0.1217	1.4882	...	73.8432	0.4990	0.0103	0.0025	2.0544	0.0202	0.0149
4	2008-07-19 15:22:00	3032.24	2502.87	2233.3667	1326.5200	1.5334	100.0	100.3967	0.1235	1.5031	...	NaN	0.4800	0.4766	0.1045	99.3032	0.0202	0.0149

5 rows × 592 columns

- first column is time Time where at different instances the signals are captured
- last column is result that signal passes or fails -1 correspods to pass and 1 corresponds to fails

In [618…
```
df['Pass/Fail'].value_counts()
```

Out[618]:
```
Pass/Fail
-1    1463
 1     104
Name: count, dtype: int64
```

- the data is little biased towards pass signals we may require to balance it

In [619…
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1567 entries, 0 to 1566
Columns: 592 entries, Time to Pass/Fail
dtypes: float64(590), int64(1), object(1)
memory usage: 7.1+ MB
```

1 B. Print 5 point summary and share at least 2 observations. [3 Marks]

In [620…
```
df.describe().T
```

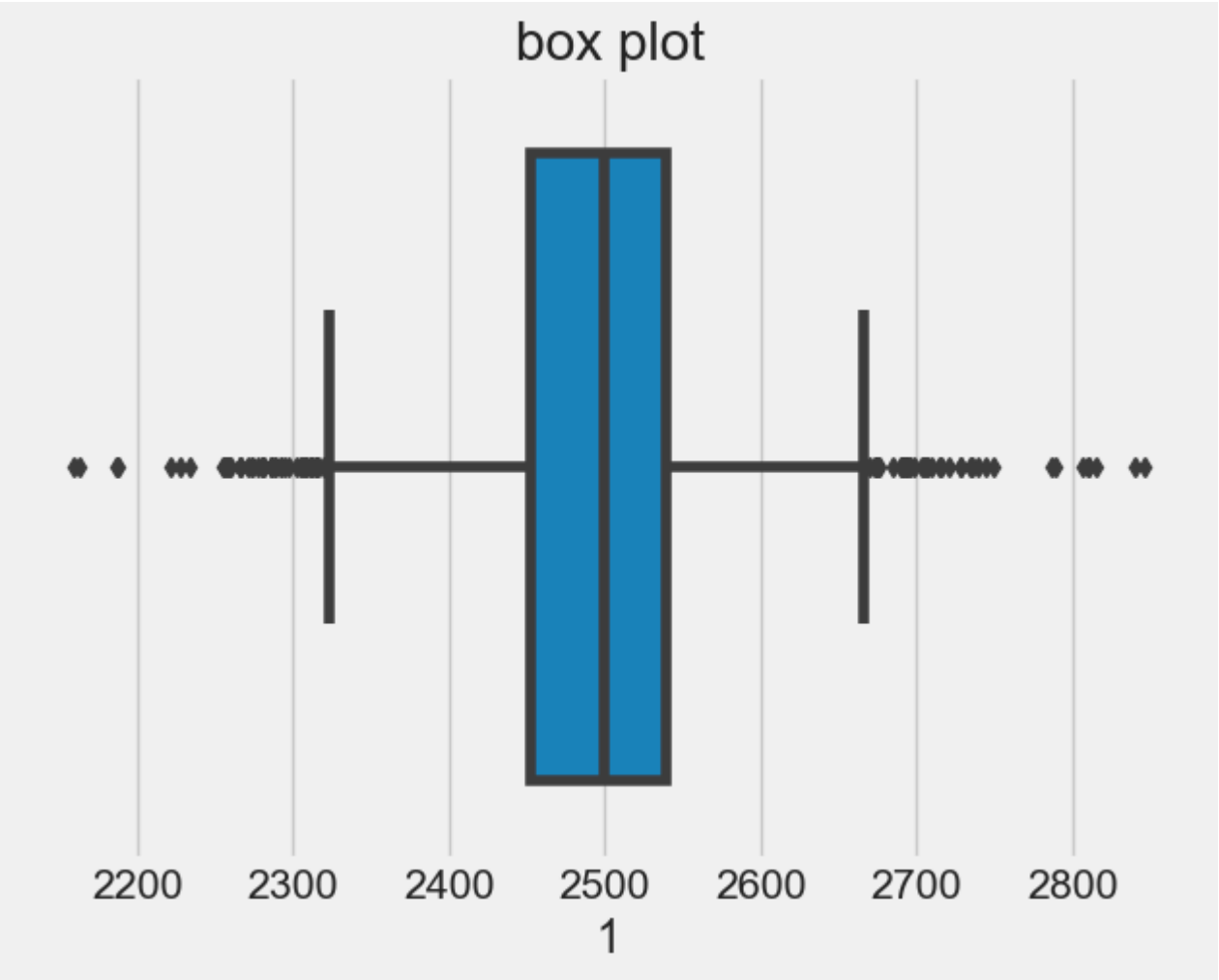
Out[620]:

	count	mean	std	min	25%	50%	75%	max
0	1561.0	3014.452896	73.621787	2743.2400	2966.260000	3011.4900	3056.6500	3356.3500
1	1560.0	2495.850231	80.407705	2158.7500	2452.247500	2499.4050	2538.8225	2846.4400
2	1553.0	2200.547318	29.513152	2060.6600	2181.044400	2201.0667	2218.0555	2315.2667
3	1553.0	1396.376627	441.691640	0.0000	1081.875800	1285.2144	1591.2235	3715.0417
4	1553.0	4.197013	56.355540	0.6815	1.017700	1.3168	1.5257	1114.5366
...	...	...	...	...	...	...	...	...
586	1566.0	0.021458	0.012358	-0.0169	0.013425	0.0205	0.0276	0.1028
587	1566.0	0.016475	0.008808	0.0032	0.010600	0.0148	0.0203	0.0799
588	1566.0	0.005283	0.002867	0.0010	0.003300	0.0046	0.0064	0.0286
589	1566.0	99.670066	93.891919	0.0000	44.368600	71.9005	114.7497	737.3048
Pass/Fail	1567.0	-0.867262	0.498010	-1.0000	-1.000000	-1.0000	-1.0000	1.0000

591 rows × 8 columns

In [621...

```
plt.title("box plot")
plt.xlabel("signal 1")
sns.boxplot(data=df,x='1')
plt.show()
```

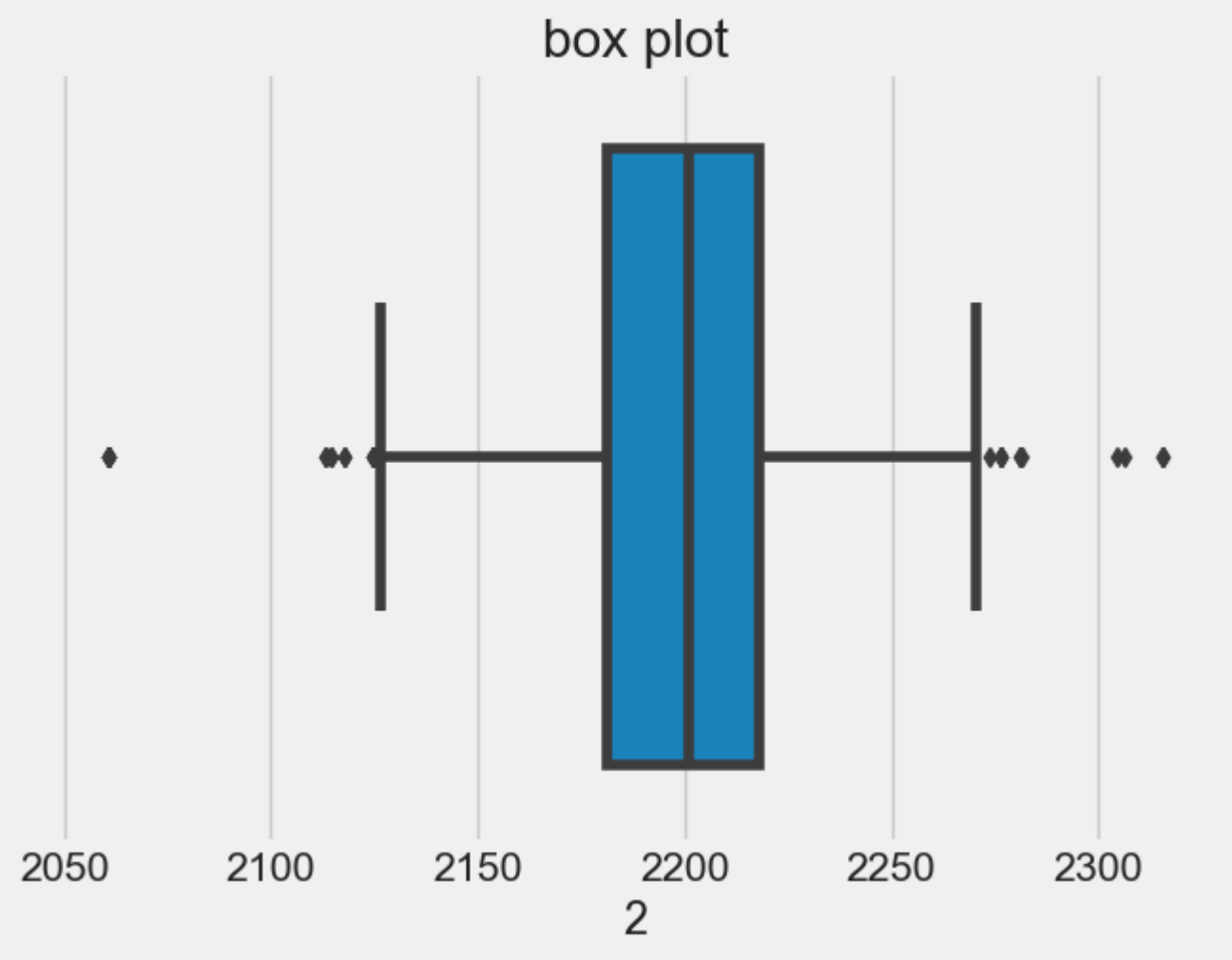


observation for signal 1

- from the summary we can see there are to many outliners may present for each feature and signals
- for signal '1' the mean value is 2495.85 and minimum value is about 2158 max value is 2846 50% of signal value lies in range from 2158 to 2500 some time signal value falls to low upto 80 which does not represent the usual data patterns (outliers)

In [622…

```
plt.title("box plot")
plt.xlabel("signal 1")
sns.boxplot(data=df,x='2')
plt.show()
```



observations for signal 2

- for signal '2' the mean value is 2200 and minimum value is about 2060 ans max value is 2315 50% of the signal value lies in range of 2181 to 2201 some of the signal value fass to low upto 29 which does not represent the usual data patterns (outliers

2. Data cleansing: [15 Marks]

A. Write a for loop which will remove all the features with 20%+ Null values and impute rest with mean of the feature. [5 Marks]

In [623…

```
null_list_gtr20=[]

# creating new data frame new df which has filtered the observation the feature
# which has more than 20% null value present.

newdf=df.copy()
# creating copy and keeping original data intact

for i in range (df.shape[1]-2):
    null_sum=df[df.columns[i+1]].isnull().sum()
    total_sum=df[df.columns[i+1]].isnull().count()
    percent=null_sum/total_sum

#here we are calculating percentage if column has  greater than 20% null value present will be list down.

    if(percent>.20):
        null_list_gtr20.append(df.columns[i+1])
#droping all that column as per null_list_gtr20

for j in null_list_gtr20 :
    newdf.drop(j,axis=1,inplace=True)
```

In [624…

```
newdf.head()
```

Out[624]:

	Time	0	1	2	3	4	5	6	7	8	...	577	582	583	584	585	586	587
0	2008-07-19 11:55:00	3030.93	2564.00	2187.7333	1411.1265	1.3602	100.0	97.6133	0.1242	1.5005	...	14.9509	0.5005	0.0118	0.0035	2.3630	NaN	NaN
1	2008-07-19 12:32:00	3095.78	2465.14	2230.4222	1463.6606	0.8294	100.0	102.3433	0.1247	1.4966	...	10.9003	0.5019	0.0223	0.0055	4.4447	0.0096	0.0201
2	2008-07-19 13:17:00	2932.61	2559.94	2186.4111	1698.0172	1.5102	100.0	95.4878	0.1241	1.4436	...	9.2721	0.4958	0.0157	0.0039	3.1745	0.0584	0.0484
3	2008-07-19 14:43:00	2988.72	2479.90	2199.0333	909.7926	1.3204	100.0	104.2367	0.1217	1.4882	...	8.5831	0.4990	0.0103	0.0025	2.0544	0.0202	0.0149
4	2008-07-19 15:22:00	3032.24	2502.87	2233.3667	1326.5200	1.5334	100.0	100.3967	0.1235	1.5031	...	10.9698	0.4800	0.4766	0.1045	99.3032	0.0202	0.0149

5 rows × 560 columns

In [625...
```
print("Now after droping column number of columns in new dataframe is ", newdf.shape[1], "\nOriginal number of columns were ",df.
print("It has reduced",df.shape[1]-newdf.shape[1],'columns')
```

```
Now after droping column number of columns in new dataframe is  560
Original number of columns were  592
It has reduced 32 columns
```

replacing rest of the null values with mean of the feature

In [626...
```
list=[]
for i in range (newdf.shape[1]-2):
    list.append(newdf.columns[i+1])
```

In [627...
```
newdf[list] = newdf[list].fillna(newdf[list].mean())
```

In [628...
```
# final check is that value is present in that perticular
```

In [629…
```
newdf.head()
```

Out[629]:

	Time	0	1	2	3	4	5	6	7	8	...	577	582	583	584	585	586	5
0	2008-07-19 11:55:00	3030.93	2564.00	2187.7333	1411.1265	1.3602	100.0	97.6133	0.1242	1.5005	...	14.9509	0.5005	0.0118	0.0035	2.3630	0.021458	0.0164
1	2008-07-19 12:32:00	3095.78	2465.14	2230.4222	1463.6606	0.8294	100.0	102.3433	0.1247	1.4966	...	10.9003	0.5019	0.0223	0.0055	4.4447	0.009600	0.0201
2	2008-07-19 13:17:00	2932.61	2559.94	2186.4111	1698.0172	1.5102	100.0	95.4878	0.1241	1.4436	...	9.2721	0.4958	0.0157	0.0039	3.1745	0.058400	0.0484
3	2008-07-19 14:43:00	2988.72	2479.90	2199.0333	909.7926	1.3204	100.0	104.2367	0.1217	1.4882	...	8.5831	0.4990	0.0103	0.0025	2.0544	0.020200	0.0149
4	2008-07-19 15:22:00	3032.24	2502.87	2233.3667	1326.5200	1.5334	100.0	100.3967	0.1235	1.5031	...	10.9698	0.4800	0.4766	0.1045	99.3032	0.020200	0.0149

5 rows × 560 columns

- we can that null value are replace with mean let check the data

In [630…
```
newdf.isnull().any().any()
```

Out[630]:
```
False
```

In [631…
```
newdf.shape
```

Out[631]:
```
(1567, 560)
```

we have successfully replace null values with mean of the feature and have dropped column with null values more then 20%

2 B. Identify and drop the features which are having same value for all the rows. [3 Marks]

In [632…

```
j=0;
for i in range (newdf.shape[1]-2):
    if(newdf[newdf.columns[i]].nunique()==1):
        print("the column name ", newdf.columns[i],"has only",newdf[newdf.columns[i]].nunique()," unique value present")
        j=j+1
```

```
the column name  5 has only 1  unique value present
the column name  13 has only 1  unique value present
the column name  42 has only 1  unique value present
the column name  49 has only 1  unique value present
the column name  52 has only 1  unique value present
the column name  69 has only 1  unique value present
the column name  97 has only 1  unique value present
the column name  141 has only 1  unique value present
the column name  149 has only 1  unique value present
the column name  178 has only 1  unique value present
the column name  179 has only 1  unique value present
the column name  186 has only 1  unique value present
the column name  189 has only 1  unique value present
the column name  190 has only 1  unique value present
the column name  191 has only 1  unique value present
the column name  192 has only 1  unique value present
the column name  193 has only 1  unique value present
the column name  194 has only 1  unique value present
the column name  226 has only 1  unique value present
the column name  229 has only 1  unique value present
the column name  230 has only 1  unique value present
the column name  231 has only 1  unique value present
the column name  232 has only 1  unique value present
the column name  233 has only 1  unique value present
the column name  234 has only 1  unique value present
the column name  235 has only 1  unique value present
the column name  236 has only 1  unique value present
the column name  237 has only 1  unique value present
the column name  240 has only 1  unique value present
the column name  241 has only 1  unique value present
the column name  242 has only 1  unique value present
the column name  243 has only 1  unique value present
the column name  256 has only 1  unique value present
the column name  257 has only 1  unique value present
the column name  258 has only 1  unique value present
the column name  259 has only 1  unique value present
the column name  260 has only 1  unique value present
the column name  261 has only 1  unique value present
the column name  262 has only 1  unique value present
the column name  263 has only 1  unique value present
the column name  264 has only 1  unique value present
the column name  265 has only 1  unique value present
the column name  266 has only 1  unique value present
the column name  276 has only 1  unique value present
```

```
the column name  284 has only 1  unique value present
the column name  313 has only 1  unique value present
the column name  314 has only 1  unique value present
the column name  315 has only 1  unique value present
the column name  322 has only 1  unique value present
the column name  325 has only 1  unique value present
the column name  326 has only 1  unique value present
the column name  327 has only 1  unique value present
the column name  328 has only 1  unique value present
the column name  329 has only 1  unique value present
the column name  330 has only 1  unique value present
the column name  364 has only 1  unique value present
the column name  369 has only 1  unique value present
the column name  370 has only 1  unique value present
the column name  371 has only 1  unique value present
the column name  372 has only 1  unique value present
the column name  373 has only 1  unique value present
the column name  374 has only 1  unique value present
the column name  375 has only 1  unique value present
the column name  378 has only 1  unique value present
the column name  379 has only 1  unique value present
the column name  380 has only 1  unique value present
the column name  381 has only 1  unique value present
the column name  394 has only 1  unique value present
the column name  395 has only 1  unique value present
the column name  396 has only 1  unique value present
the column name  397 has only 1  unique value present
the column name  398 has only 1  unique value present
the column name  399 has only 1  unique value present
the column name  400 has only 1  unique value present
the column name  401 has only 1  unique value present
the column name  402 has only 1  unique value present
the column name  403 has only 1  unique value present
the column name  404 has only 1  unique value present
the column name  414 has only 1  unique value present
the column name  422 has only 1  unique value present
the column name  449 has only 1  unique value present
the column name  450 has only 1  unique value present
the column name  451 has only 1  unique value present
the column name  458 has only 1  unique value present
the column name  461 has only 1  unique value present
the column name  462 has only 1  unique value present
the column name  463 has only 1  unique value present
the column name  464 has only 1  unique value present
```

```
the column name  465 has only 1  unique value present
the column name  466 has only 1  unique value present
the column name  481 has only 1  unique value present
the column name  498 has only 1  unique value present
the column name  501 has only 1  unique value present
the column name  502 has only 1  unique value present
the column name  503 has only 1  unique value present
the column name  504 has only 1  unique value present
the column name  505 has only 1  unique value present
the column name  506 has only 1  unique value present
the column name  507 has only 1  unique value present
the column name  508 has only 1  unique value present
the column name  509 has only 1  unique value present
the column name  512 has only 1  unique value present
the column name  513 has only 1  unique value present
the column name  514 has only 1  unique value present
the column name  515 has only 1  unique value present
the column name  528 has only 1  unique value present
the column name  529 has only 1  unique value present
the column name  530 has only 1  unique value present
the column name  531 has only 1  unique value present
the column name  532 has only 1  unique value present
the column name  533 has only 1  unique value present
the column name  534 has only 1  unique value present
the column name  535 has only 1  unique value present
the column name  536 has only 1  unique value present
the column name  537 has only 1  unique value present
the column name  538 has only 1  unique value present
```

In [633…
```
print("total column present which has value present = ",j)
```

```
total column present which has value present =  116
```

In [634…
```
nunique = newdf.nunique()
cols_to_drop = nunique[nunique == 1].index
newdf.drop(cols_to_drop, axis=1,inplace=True)
```

In [635…
```
newdf.head()
```

Out[635]:

	Time	0	1	2	3	4	6	7	8	9	...	577	582	583	584	585	586	
0	2008-07-19 11:55:00	3030.93	2564.00	2187.7333	1411.1265	1.3602	97.6133	0.1242	1.5005	0.0162	...	14.9509	0.5005	0.0118	0.0035	2.3630	0.021458	0.01
1	2008-07-19 12:32:00	3095.78	2465.14	2230.4222	1463.6606	0.8294	102.3433	0.1247	1.4966	-0.0005	...	10.9003	0.5019	0.0223	0.0055	4.4447	0.009600	0.02
2	2008-07-19 13:17:00	2932.61	2559.94	2186.4111	1698.0172	1.5102	95.4878	0.1241	1.4436	0.0041	...	9.2721	0.4958	0.0157	0.0039	3.1745	0.058400	0.04
3	2008-07-19 14:43:00	2988.72	2479.90	2199.0333	909.7926	1.3204	104.2367	0.1217	1.4882	-0.0124	...	8.5831	0.4990	0.0103	0.0025	2.0544	0.020200	0.01
4	2008-07-19 15:22:00	3032.24	2502.87	2233.3667	1326.5200	1.5334	100.3967	0.1235	1.5031	-0.0031	...	10.9698	0.4800	0.4766	0.1045	99.3032	0.020200	0.01

5 rows × 444 columns

- it has removed columns which has same values from all rows and fillnay we have 444 number of columns

In [636…
```
newdf.shape
```

Out[636]:
```
(1567, 444)
```

2C. Drop other features if required using relevant functional knowledge. Clearly justify the same. [2 Marks]

In [637…
```
newdf['Time'].dtype
```

Out[637]:
```
dtype('O')
```

In [638…
```
newdf=newdf.drop(columns=['Time'],axis=1)
```

In [639…
```
newdf.shape
```

Out[639]:
```
(1567, 443)
```

In [640…
```
newdf.head(3)
```

Out[640]:

	0	1	2	3	4	6	7	8	9	10	...	577	582	583	584	585	586	5
0	3030.93	2564.00	2187.7333	1411.1265	1.3602	97.6133	0.1242	1.5005	0.0162	-0.0034	...	14.9509	0.5005	0.0118	0.0035	2.3630	0.021458	0.0164
1	3095.78	2465.14	2230.4222	1463.6606	0.8294	102.3433	0.1247	1.4966	-0.0005	-0.0148	...	10.9003	0.5019	0.0223	0.0055	4.4447	0.009600	0.0201
2	2932.61	2559.94	2186.4111	1698.0172	1.5102	95.4878	0.1241	1.4436	0.0041	0.0013	...	9.2721	0.4958	0.0157	0.0039	3.1745	0.058400	0.0484

3 rows × 443 columns

- we can remove data time column whic is not require to under stand the patter and behaviour of the signals here

2D. Check for multi-collinearity in the data and take necessary action. [3 Marks]

In [641…
```
#Remove the highly collinear features from data
def remove_collinear_features(x, threshold):
    '''
    Objective:
        Remove collinear features in a dataframe with a correlation coefficient
        greater than the threshold. Removing collinear features can help a model
        to generalize and improves the interpretability of the model.

    Inputs:
        x: features dataframe
        threshold: features with correlations greater than this value are removed

    Output:
        dataframe that contains only the non-highly-collinear features
    '''
```

```
    # Calculate the correlation matrix
    corr_matrix = x.corr()
    iters = range(len(corr_matrix.columns) - 1)
    drop_cols = []

    # Iterate through the correlation matrix and compare correlations
    for i in iters:
        for j in range(i+1):
            item = corr_matrix.iloc[j:(j+1), (i+1):(i+2)]
            col = item.columns
            row = item.index
            val = abs(item.values)

            # If correlation exceeds the threshold
            if val >= threshold:
                # Print the correlated features and the correlation value
                print(col.values[0], "|", row.values[0], "|", round(val[0][0], 2))
                drop_cols.append(col.values[0])

    # Drop one of each pair of correlated columns
    drops = set(drop_cols)
    x = x.drop(columns=drops)

    return x
```

In [642…

```
data = remove_collinear_features(newdf,0.85)
```

27 | 25 | 0.98
30 | 29 | 0.86
36 | 34 | 1.0
50 | 46 | 0.9
54 | 53 | 0.94
60 | 43 | 0.9
70 | 66 | 0.9
96 | 94 | 0.96
98 | 96 | 0.87
101 | 98 | 0.91
104 | 99 | 0.99
105 | 92 | 0.99
106 | 93 | 0.99
123 | 121 | 0.94
124 | 121 | 0.89
124 | 123 | 0.86
127 | 122 | 0.96
140 | 4 | 1.0
147 | 16 | 0.89
148 | 16 | 0.97
148 | 147 | 0.89
152 | 16 | 0.98
152 | 147 | 0.9
152 | 148 | 0.99
154 | 16 | 0.87
154 | 148 | 0.94
154 | 152 | 0.89
164 | 163 | 0.92
165 | 163 | 0.9
165 | 164 | 0.96
174 | 172 | 1.0
196 | 67 | 0.86
197 | 67 | 0.86
197 | 196 | 0.9
199 | 196 | 0.94
204 | 67 | 0.9
204 | 196 | 0.87
205 | 67 | 0.87
205 | 196 | 0.86
206 | 74 | 1.0
207 | 67 | 0.86
207 | 196 | 0.92
207 | 197 | 0.87
207 | 199 | 0.88

207 | 203 | 0.86
207 | 204 | 0.87
207 | 205 | 0.87
209 | 74 | 1.0
209 | 206 | 1.0
249 | 114 | 0.98
252 | 117 | 0.99
270 | 135 | 0.95
271 | 136 | 0.97
272 | 137 | 0.98
273 | 138 | 0.92
274 | 139 | 0.99
275 | 4 | 1.0
275 | 140 | 1.0
277 | 142 | 0.97
278 | 143 | 0.91
279 | 144 | 0.98
280 | 145 | 0.96
281 | 146 | 0.95
282 | 16 | 0.88
282 | 147 | 1.0
282 | 148 | 0.89
282 | 152 | 0.89
283 | 16 | 0.97
283 | 147 | 0.89
283 | 148 | 1.0
283 | 152 | 0.99
283 | 154 | 0.94
283 | 282 | 0.89
285 | 150 | 0.97
286 | 151 | 0.99
287 | 16 | 0.98
287 | 147 | 0.9
287 | 148 | 0.99
287 | 152 | 1.0
287 | 154 | 0.89
287 | 282 | 0.89
287 | 283 | 0.99
288 | 153 | 1.0
289 | 16 | 0.88
289 | 148 | 0.94
289 | 152 | 0.89
289 | 154 | 0.99
289 | 283 | 0.94

289 | 287 | 0.89
290 | 155 | 0.95
291 | 156 | 0.99
294 | 159 | 0.99
295 | 160 | 1.0
296 | 161 | 0.99
297 | 162 | 0.99
298 | 163 | 0.99
298 | 164 | 0.94
298 | 165 | 0.92
299 | 163 | 0.92
299 | 164 | 1.0
299 | 165 | 0.96
299 | 298 | 0.95
300 | 163 | 0.9
300 | 164 | 0.97
300 | 165 | 1.0
300 | 298 | 0.93
300 | 299 | 0.97
301 | 166 | 0.96
302 | 167 | 0.98
303 | 168 | 0.96
304 | 169 | 0.98
305 | 170 | 0.96
306 | 171 | 0.99
307 | 172 | 0.96
307 | 174 | 0.96
308 | 173 | 0.96
309 | 172 | 0.96
309 | 174 | 0.96
309 | 307 | 1.0
310 | 175 | 0.96
311 | 176 | 0.98
312 | 177 | 1.0
316 | 180 | 0.88
317 | 181 | 0.96
318 | 182 | 0.98
319 | 183 | 0.98
320 | 184 | 0.99
321 | 185 | 0.99
323 | 187 | 0.99
324 | 188 | 0.98
331 | 195 | 0.95
332 | 67 | 0.88

332 | 196 | 0.96
332 | 197 | 0.91
332 | 199 | 0.9
332 | 205 | 0.89
332 | 207 | 0.91
333 | 67 | 0.87
333 | 196 | 0.87
333 | 197 | 0.98
333 | 207 | 0.87
333 | 332 | 0.9
334 | 198 | 0.99
335 | 196 | 0.93
335 | 197 | 0.86
335 | 199 | 0.96
335 | 205 | 0.85
335 | 207 | 0.9
335 | 332 | 0.96
335 | 333 | 0.86
336 | 67 | 0.87
336 | 196 | 0.91
336 | 197 | 0.9
336 | 199 | 0.88
336 | 205 | 0.88
336 | 207 | 0.9
336 | 332 | 0.94
336 | 333 | 0.9
336 | 335 | 0.93
337 | 201 | 0.93
338 | 74 | 0.87
338 | 202 | 0.99
338 | 203 | 0.86
338 | 206 | 0.87
338 | 209 | 0.87
339 | 202 | 0.88
339 | 203 | 0.98
339 | 338 | 0.91
340 | 67 | 0.95
340 | 196 | 0.85
340 | 204 | 0.99
341 | 67 | 0.91
341 | 196 | 0.86
341 | 197 | 0.85
341 | 204 | 0.87
341 | 205 | 0.99

341 | 207 | 0.87
341 | 332 | 0.89
341 | 333 | 0.86
341 | 335 | 0.85
341 | 336 | 0.89
341 | 340 | 0.89
342 | 74 | 1.0
342 | 206 | 1.0
342 | 209 | 1.0
342 | 338 | 0.87
343 | 67 | 0.87
343 | 196 | 0.9
343 | 197 | 0.88
343 | 199 | 0.87
343 | 205 | 0.88
343 | 207 | 0.98
343 | 332 | 0.94
343 | 333 | 0.89
343 | 335 | 0.92
343 | 336 | 0.92
343 | 341 | 0.88
344 | 208 | 0.96
347 | 74 | 1.0
347 | 206 | 1.0
347 | 209 | 1.0
347 | 338 | 0.87
347 | 342 | 1.0
348 | 210 | 0.95
349 | 211 | 0.99
350 | 212 | 0.99
351 | 213 | 1.0
352 | 214 | 0.98
353 | 215 | 0.98
354 | 216 | 0.97
355 | 217 | 0.99
356 | 218 | 0.95
357 | 219 | 0.98
359 | 221 | 0.98
360 | 222 | 0.99
361 | 223 | 0.98
362 | 224 | 1.0
363 | 225 | 0.96
365 | 227 | 0.97
366 | 228 | 0.97

376 | 238 | 0.97
377 | 239 | 0.95
386 | 248 | 1.0
387 | 114 | 0.98
387 | 249 | 1.0
388 | 250 | 0.97
389 | 251 | 1.0
390 | 117 | 0.99
390 | 252 | 1.0
391 | 253 | 0.99
392 | 254 | 0.99
393 | 255 | 0.99
405 | 267 | 0.99
406 | 268 | 0.97
407 | 269 | 0.95
408 | 135 | 1.0
408 | 270 | 0.94
409 | 136 | 1.0
409 | 271 | 0.97
410 | 137 | 1.0
410 | 272 | 0.97
411 | 138 | 1.0
411 | 273 | 0.92
413 | 4 | 0.94
413 | 140 | 0.94
413 | 275 | 0.94
415 | 142 | 0.99
415 | 277 | 0.97
416 | 143 | 1.0
416 | 278 | 0.91
417 | 144 | 0.99
417 | 279 | 0.97
420 | 16 | 0.9
420 | 147 | 1.0
420 | 148 | 0.9
420 | 152 | 0.91
420 | 282 | 1.0
420 | 283 | 0.9
420 | 287 | 0.91
421 | 16 | 0.96
421 | 147 | 0.89
421 | 148 | 1.0
421 | 152 | 0.98
421 | 154 | 0.95

421 | 282 | 0.88
421 | 283 | 1.0
421 | 287 | 0.98
421 | 289 | 0.95
421 | 420 | 0.9
424 | 151 | 0.98
424 | 286 | 0.97
425 | 16 | 0.94
425 | 147 | 0.87
425 | 148 | 0.96
425 | 152 | 0.98
425 | 154 | 0.86
425 | 282 | 0.87
425 | 283 | 0.96
425 | 287 | 0.97
425 | 289 | 0.86
425 | 420 | 0.88
425 | 421 | 0.95
426 | 153 | 1.0
426 | 288 | 0.99
427 | 16 | 0.89
427 | 148 | 0.95
427 | 152 | 0.91
427 | 154 | 1.0
427 | 283 | 0.95
427 | 287 | 0.91
427 | 289 | 0.99
427 | 421 | 0.97
427 | 425 | 0.88
428 | 155 | 1.0
428 | 290 | 0.96
429 | 156 | 1.0
429 | 291 | 0.99
430 | 159 | 0.87
430 | 164 | 0.88
430 | 294 | 0.89
430 | 299 | 0.87
431 | 430 | 0.9
434 | 163 | 0.88
434 | 164 | 0.9
434 | 165 | 0.86
434 | 298 | 0.89
434 | 299 | 0.89
434 | 300 | 0.86

434 | 430 | 0.95
434 | 431 | 0.93
435 | 164 | 0.91
435 | 165 | 0.87
435 | 298 | 0.86
435 | 299 | 0.9
435 | 300 | 0.86
435 | 430 | 0.95
435 | 431 | 0.93
435 | 434 | 0.99
436 | 164 | 0.9
436 | 165 | 0.88
436 | 298 | 0.86
436 | 299 | 0.9
436 | 300 | 0.87
436 | 430 | 0.95
436 | 431 | 0.93
436 | 434 | 0.99
436 | 435 | 1.0
437 | 166 | 0.99
437 | 301 | 0.95
440 | 169 | 1.0
440 | 304 | 0.98
441 | 170 | 0.99
441 | 305 | 0.95
442 | 171 | 0.97
442 | 306 | 0.96
443 | 172 | 1.0
443 | 174 | 1.0
443 | 307 | 0.96
443 | 309 | 0.96
444 | 173 | 0.99
444 | 308 | 0.95
445 | 172 | 1.0
445 | 174 | 1.0
445 | 307 | 0.96
445 | 309 | 0.96
445 | 443 | 0.99
446 | 175 | 1.0
446 | 310 | 0.95
447 | 176 | 1.0
447 | 311 | 0.98
448 | 177 | 1.0
448 | 312 | 1.0

452 | 180 | 0.99
452 | 316 | 0.86
453 | 181 | 1.0
453 | 317 | 0.96
454 | 182 | 0.99
454 | 318 | 0.97
455 | 183 | 1.0
455 | 319 | 0.98
456 | 184 | 0.97
456 | 320 | 0.96
457 | 185 | 1.0
457 | 321 | 0.99
459 | 187 | 1.0
459 | 323 | 0.99
467 | 195 | 1.0
467 | 331 | 0.95
469 | 67 | 0.87
469 | 196 | 0.9
469 | 197 | 1.0
469 | 207 | 0.88
469 | 332 | 0.92
469 | 333 | 0.99
469 | 335 | 0.88
469 | 336 | 0.91
469 | 341 | 0.86
469 | 343 | 0.89
470 | 198 | 1.0
470 | 334 | 0.98
471 | 199 | 0.94
473 | 201 | 0.87
475 | 202 | 0.87
475 | 203 | 1.0
475 | 338 | 0.88
475 | 339 | 0.99
477 | 67 | 0.92
477 | 196 | 0.88
477 | 204 | 0.87
477 | 205 | 0.99
477 | 207 | 0.88
477 | 332 | 0.91
477 | 335 | 0.87
477 | 336 | 0.89
477 | 340 | 0.88
477 | 341 | 0.99

477 | 343 | 0.89
477 | 469 | 0.85
478 | 74 | 1.0
478 | 206 | 1.0
478 | 209 | 1.0
478 | 338 | 0.87
478 | 342 | 1.0
478 | 347 | 1.0
479 | 67 | 0.85
479 | 196 | 0.91
479 | 197 | 0.87
479 | 199 | 0.88
479 | 203 | 0.88
479 | 204 | 0.88
479 | 205 | 0.86
479 | 207 | 1.0
479 | 332 | 0.89
479 | 333 | 0.86
479 | 335 | 0.88
479 | 336 | 0.89
479 | 340 | 0.85
479 | 341 | 0.86
479 | 343 | 0.97
479 | 469 | 0.87
479 | 475 | 0.85
479 | 477 | 0.87
490 | 218 | 0.98
490 | 356 | 0.93
491 | 219 | 1.0
491 | 357 | 0.97
493 | 221 | 1.0
493 | 359 | 0.98
494 | 222 | 1.0
494 | 360 | 1.0
495 | 223 | 1.0
495 | 361 | 0.97
497 | 225 | 0.99
497 | 363 | 0.96
520 | 248 | 1.0
520 | 386 | 1.0
522 | 250 | 0.99
522 | 388 | 0.96
523 | 251 | 1.0
523 | 389 | 1.0

524 | 117 | 0.98
524 | 252 | 1.0
524 | 390 | 1.0
525 | 253 | 1.0
525 | 391 | 0.99
526 | 254 | 1.0
526 | 392 | 0.99
527 | 255 | 1.0
527 | 393 | 0.98
539 | 267 | 1.0
539 | 405 | 0.99
540 | 268 | 1.0
540 | 406 | 0.97
541 | 269 | 0.97
541 | 407 | 0.91
545 | 543 | 0.99
552 | 549 | 1.0
553 | 550 | 0.98
554 | 551 | 1.0
555 | 549 | 0.88
555 | 552 | 0.88
556 | 550 | 1.0
556 | 553 | 0.98
557 | 551 | 1.0
557 | 554 | 1.0
560 | 559 | 0.89
561 | 559 | 0.98
566 | 564 | 0.98
567 | 565 | 0.99
568 | 564 | 1.0
568 | 566 | 0.98
569 | 565 | 0.94
569 | 567 | 0.93
574 | 572 | 0.99
575 | 573 | 0.98
576 | 572 | 0.99
576 | 574 | 0.99
577 | 572 | 0.86
577 | 573 | 0.96
577 | 574 | 0.85
577 | 575 | 0.93
577 | 576 | 0.86
584 | 583 | 0.99
585 | 583 | 1.0

```
585 | 584 | 1.0
588 | 587 | 0.97
```

In [643…
```
data.shape
```

Out[643]:
```
(1567, 234)
```

2E. Make all relevant modifications on the data using both functional/logical reasoning/assumptions. [2 Marks]

- Here initial we have observe the data which is signals from different sensors
- Target variable has imbalance data which may biased towards result 'pass' for now we are not treating it keeping it for later part of analysis
- We have successfully removed the obserations which has more than 20% of null value in feature column because such reading may cause noise in our model and can make model more complex we can set any percentage here we have set to 20%
- Reset of the null value we have replace with mean of the feature as ask because we dont want to loose usefull signals here we would have replace with median also because too many outliers are there. but for signal processing we may
- We have successfully removed the column in which sensors signal are repeating thru out rows which can be a malfunction of the sensor or will not represent any behaviour pattern so better to remove it.
- dimentionality reduction using principle component analysis will be keep for later part observe the difference between two procedures.
- SMOTE can be use for oversmapling to balnce the data wil be executed later
- we have removed highly correlated columns if correlation coeffients reaches to threashhold value .85 to reduce the complexity of the model

3. Data analysis & visualisation: [5 Marks]

3A. Perform a detailed univariate Analysis with appropriate detailed comments after each analysis. [2 Marks]

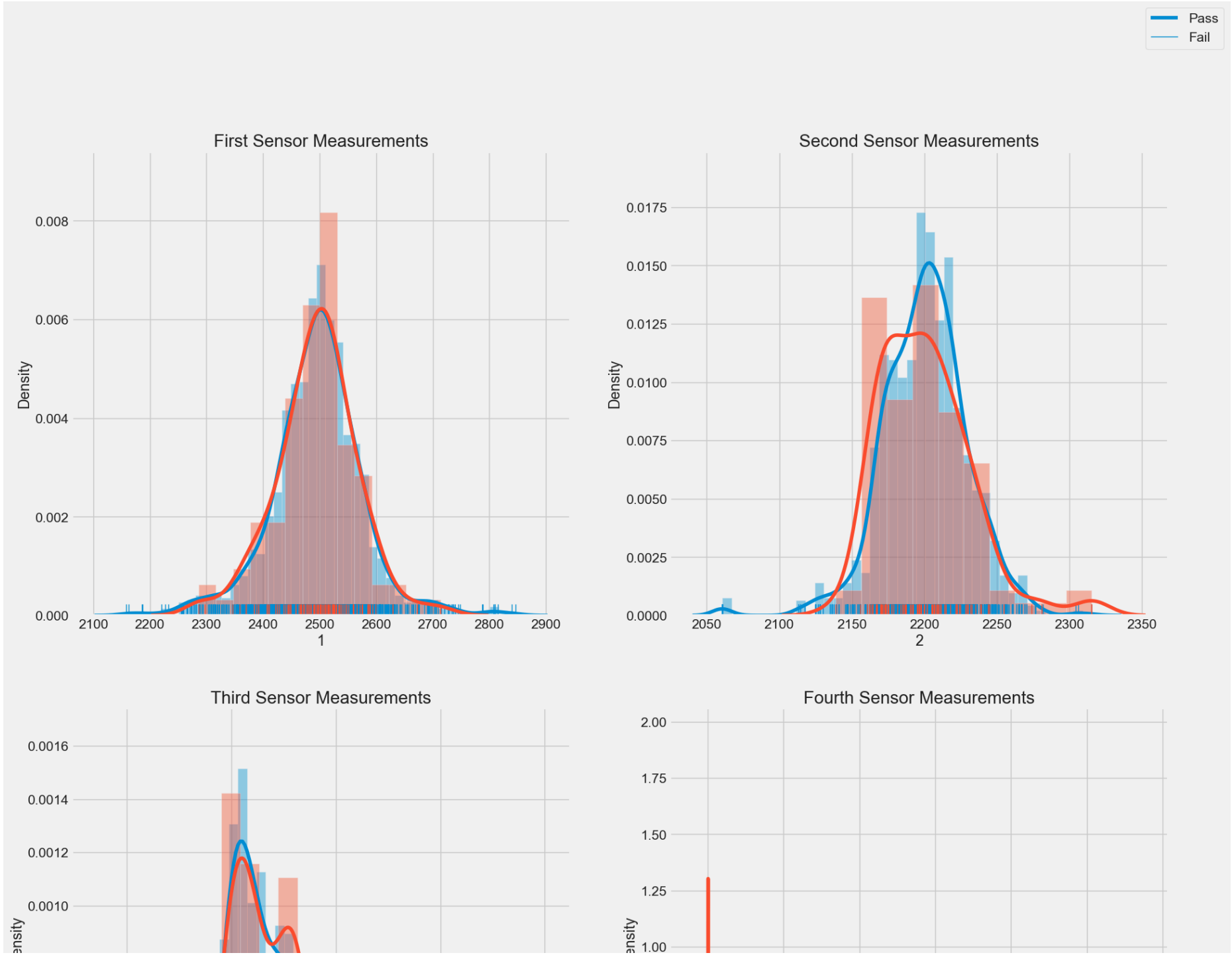
In [644…
```
unique_vals = data['Pass/Fail'].unique()  # [0, 1, 2]
unique_vals
```

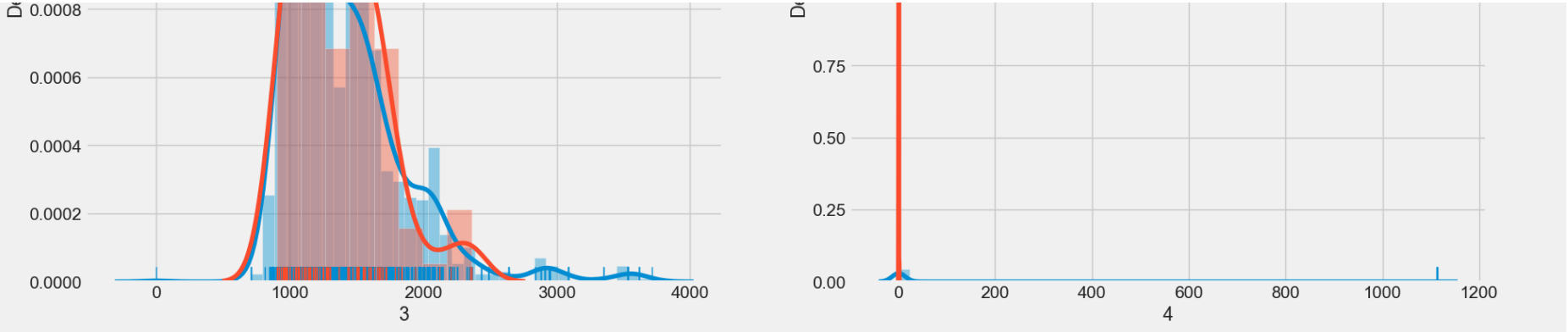
Out[644]:
```
array([-1,  1], dtype=int64)
```

In [645…
```
targets = [data.loc[data['Pass/Fail'] == val] for val in unique_vals]
```

In [646…

```
fig = plt.figure(figsize=(20,20))

plt.subplot(2, 2, 1)
for target in targets:
    sns.distplot(target['1'], hist=True, rug=True)
plt.title('First Sensor Measurements', fontsize = 20)

plt.subplot(2, 2, 2)
for target in targets:
    sns.distplot(target['2'], hist=True, rug=True)
plt.title('Second Sensor Measurements', fontsize = 20)

plt.subplot(2, 2, 3)
for target in targets:
    sns.distplot(target['3'], hist=True, rug=True)
plt.title('Third Sensor Measurements', fontsize = 20)

plt.subplot(2, 2, 4)
for target in targets:
    sns.distplot(target['4'], hist=True, rug=True)
plt.title('Fourth Sensor Measurements', fontsize = 20)

#sns.add_legend()
#plt.legend()
fig.legend(labels=['Pass','Fail'])
plt.show()
```

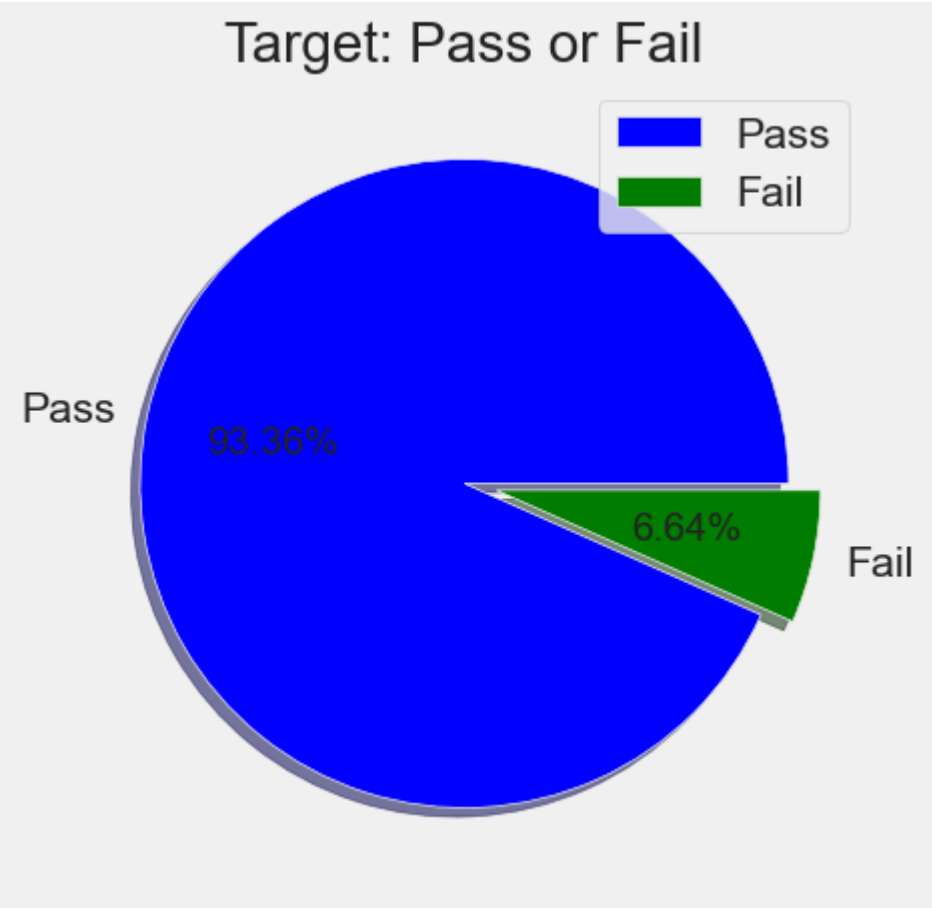

Pass
Fail
First Sensor Measurements
Density
0.008
0.006
0.004
0.002
0.000
2100
2200
2300
2400
2500
2600
2700
2800
2900
1
Second Sensor Measurements
Density
0.0175
0.0150
0.0125
0.0100
0.0075
0.0050
0.0025
0.0000
2050
2100
2150
2200
2250
2300
2350
2
Third Sensor Measurements
0.0016
0.0014
0.0012
0.0010
Fourth Sensor Measurements
2.00
1.75
1.50
1.25
1.00

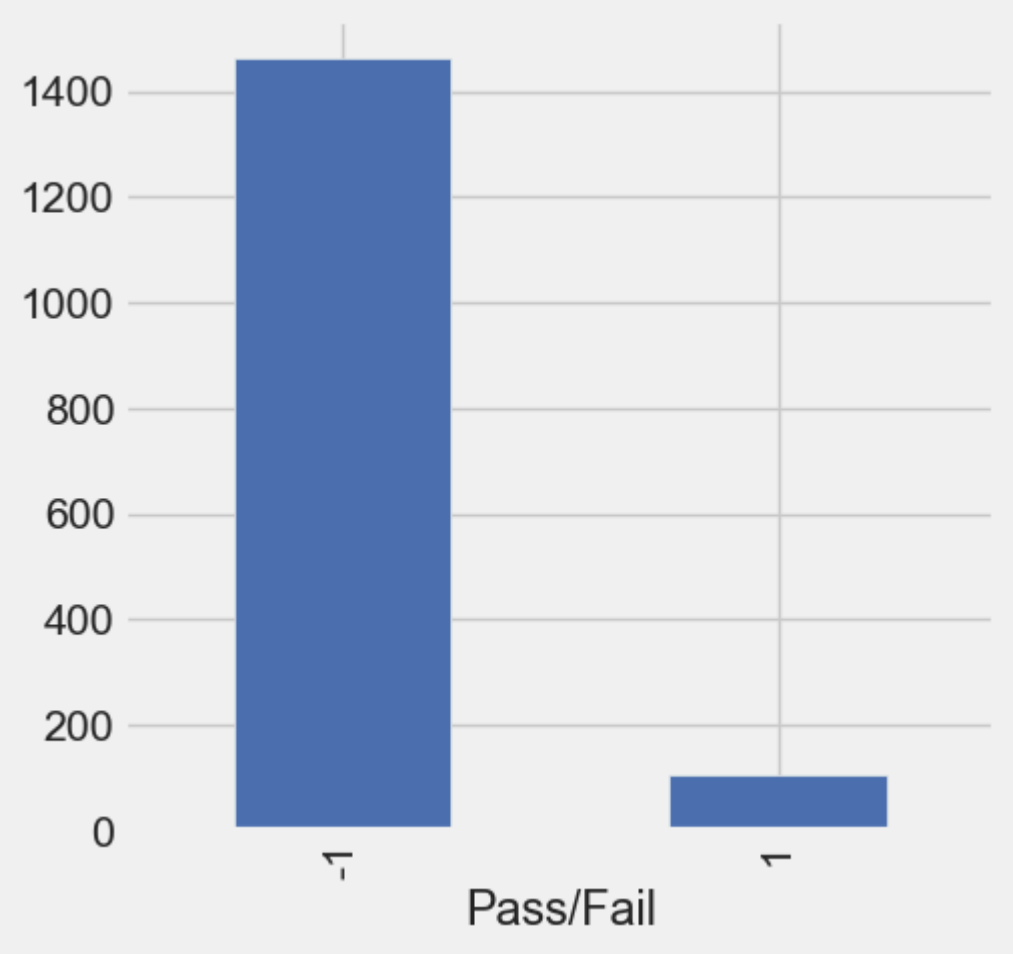


In [ ]:

In [647…

```
# pie chart
# We have highly imbalanced class with only 6.6% failures and 93.4% pass

labels = ['Pass', 'Fail']
size = data['Pass/Fail'].value_counts()
colors = ['blue', 'green']
explode = [0, 0.1]

plt.style.use('seaborn-deep')
plt.rcParams['figure.figsize'] = (5, 5)
plt.pie(size, labels =labels, colors = colors, explode = explode, autopct = "%.2f%%", shadow = True)
plt.axis('off')
plt.title('Target: Pass or Fail', fontsize = 20)
plt.legend()
plt.show()
data['Pass/Fail'].value_counts().plot(kind="bar");
```


Target: Pass or Fail
Pass
Fail
Pass
93.36%
6.64%
Fail



In [648…
```
data['Pass/Fail'].value_counts()
```

Out[648]:
```
Pass/Fail
-1    1463
 1     104
Name: count, dtype: int64
```

- As per univariate analysis the most of the signal falls under pass result categories.
- Which represents that is highly imbalance data.
- pass signal has about 94% and fails has 7% which shows there is requirement of synthetic data
- some signal represent normal distribustion as per dist plot

- and some does not are bimodel or skewed

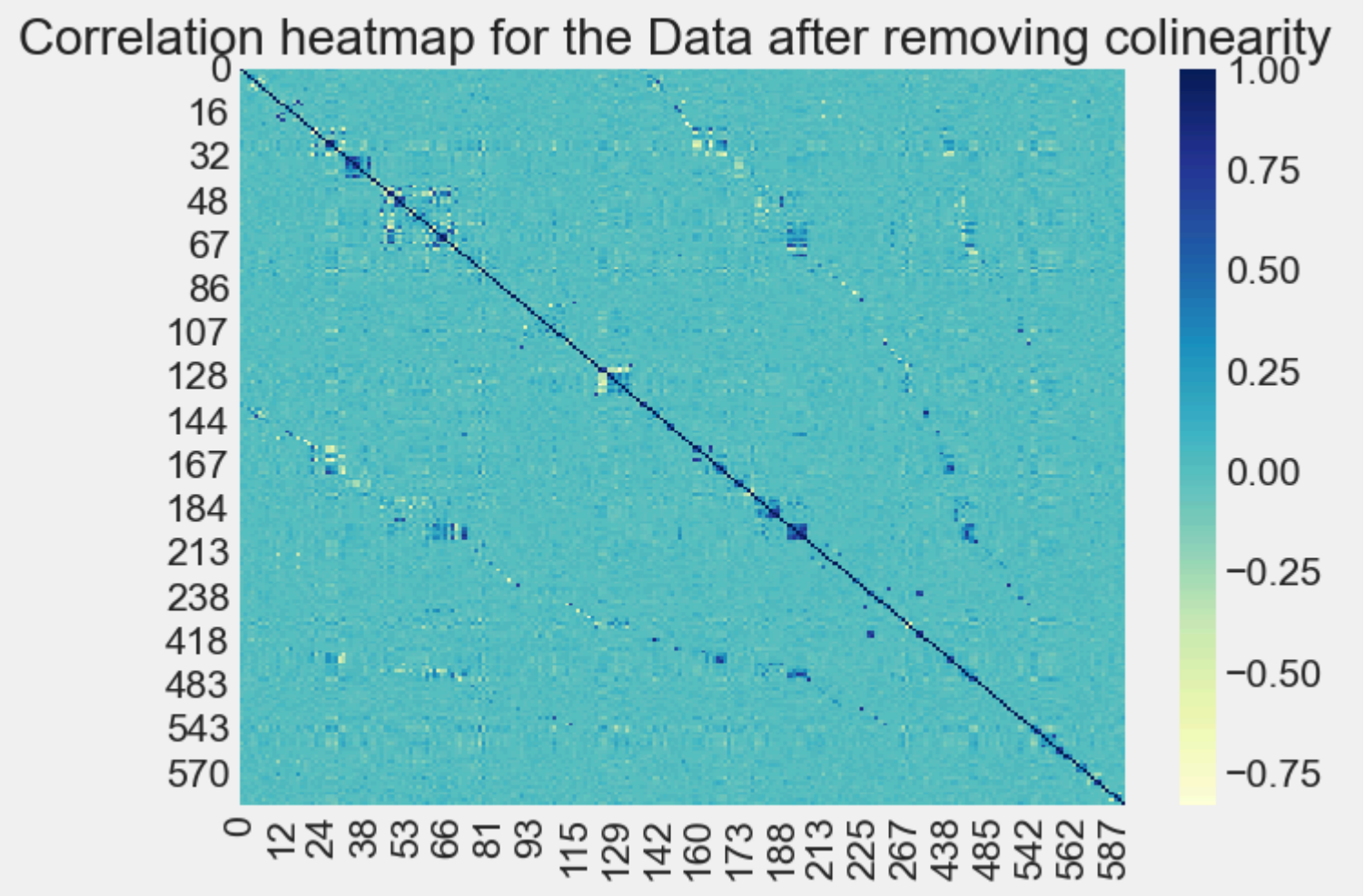
3 B. Perform bivariate and multivariate analysis with appropriate detailed comments after each analysis. [3 Marks]

In [649…

```
plt.rcParams['figure.figsize'] = (7, 5)
sns.heatmap(data.corr(), cmap = "YlGnBu")
plt.title('Correlation heatmap for the Data after removing colinearity ', fontsize = 20)
```

Out[649]:

```
Text(0.5, 1.0, 'Correlation heatmap for the Data after removing colinearity ')
```



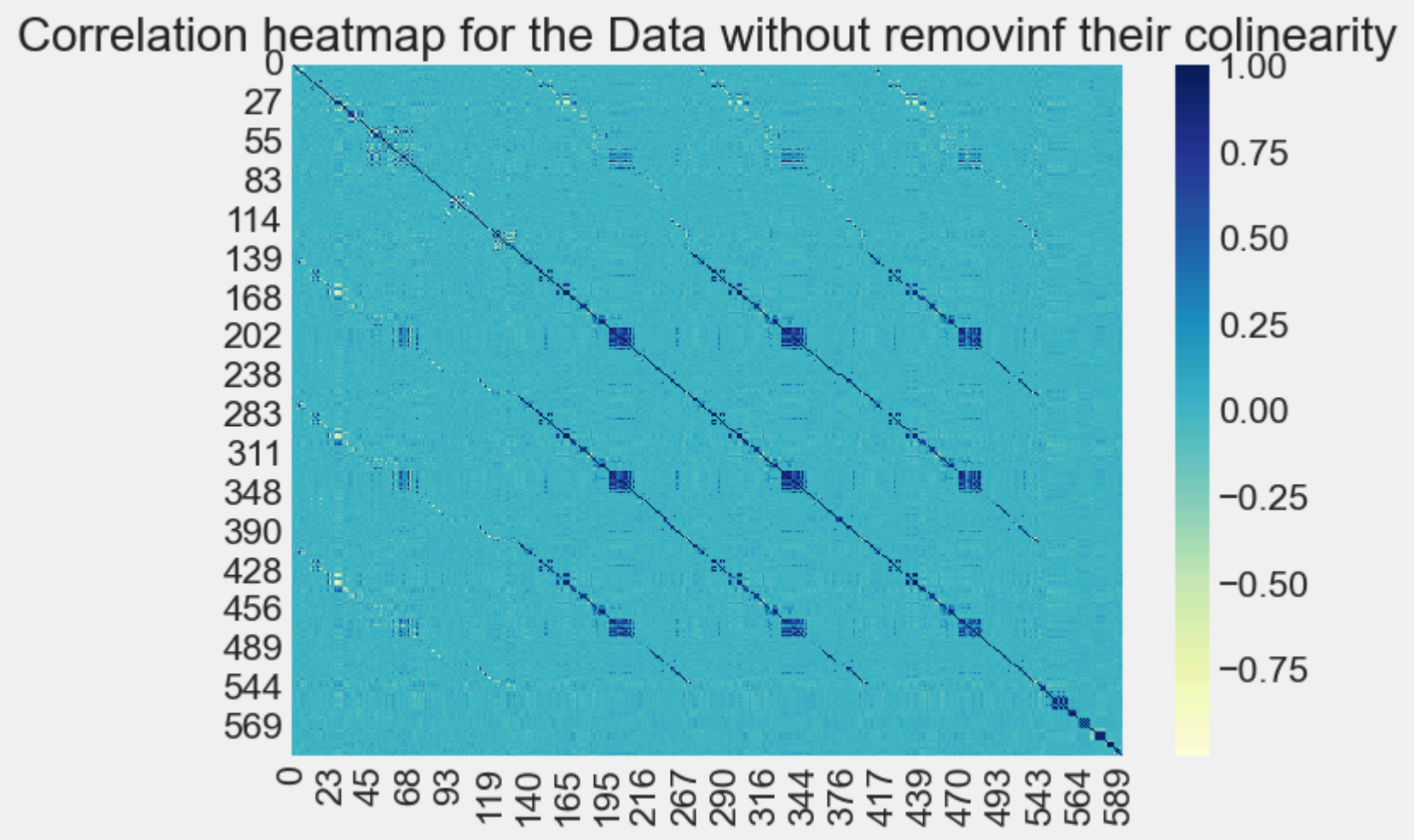
- lets see scatter plot for some of the sensors

In [650…

```
plt.rcParams['figure.figsize'] = (7, 5)
sns.heatmap(newdf.corr(), cmap = "YlGnBu")
plt.title('Correlation heatmap for the Data without removinf their colinearity', fontsize = 20)
```

Out[650]:

```
Text(0.5, 1.0, 'Correlation heatmap for the Data without removinf their colinearity')
```

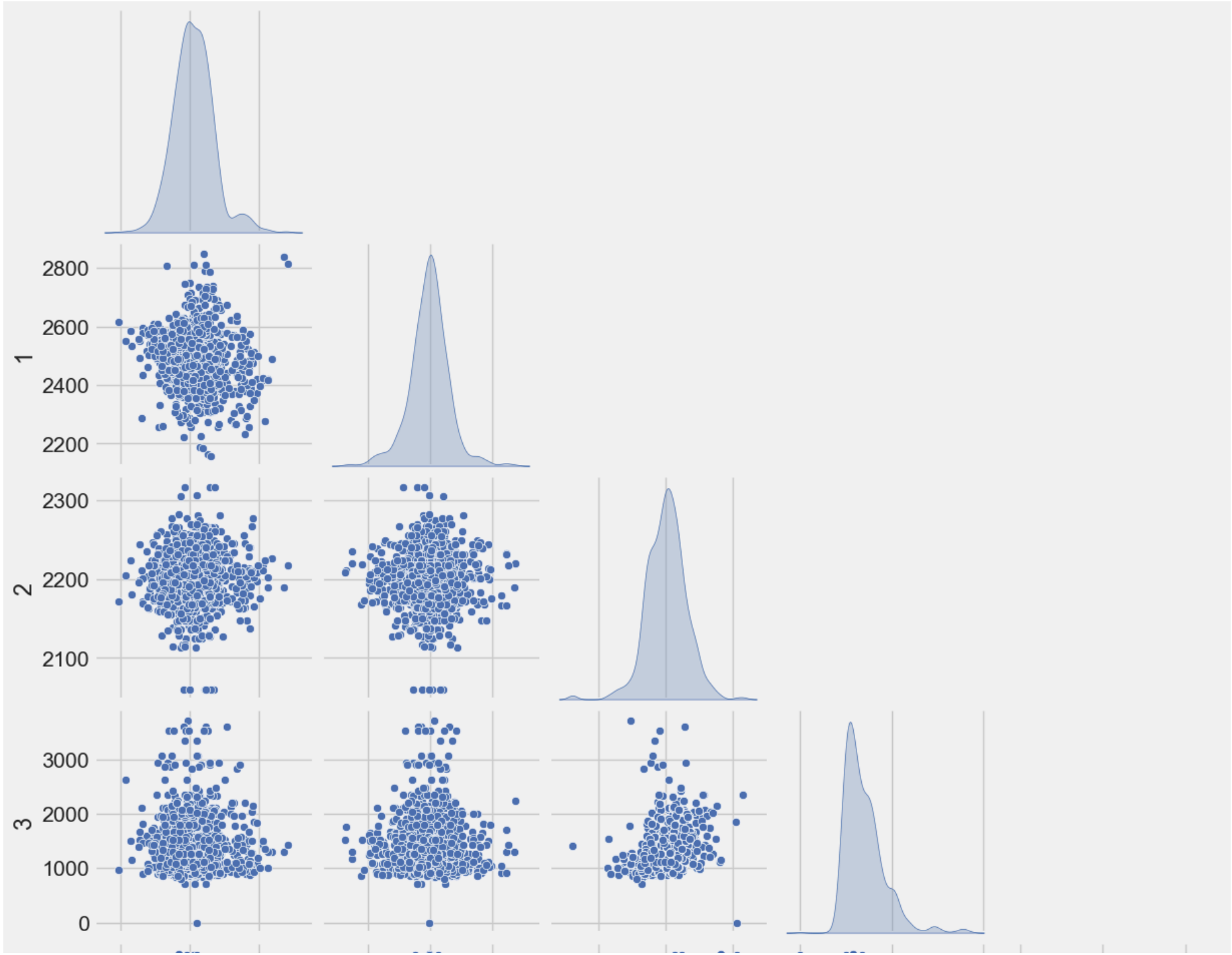


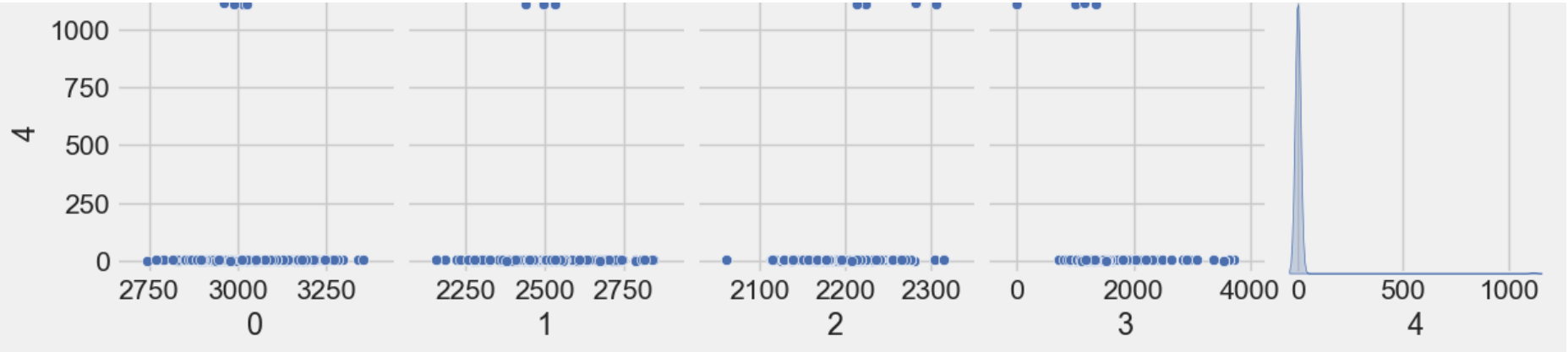
In [651…

```
pair_plot_data=data.iloc[:,0:5]
plt.figure(figsize=(10,5))
sns.pairplot(data=pair_plot_data,diag_kind='kde',corner=True)
```

Out[651]:

```
<seaborn.axisgrid.PairGrid at 0x23cb3f39bd0>

<Figure size 1000x500 with 0 Axes>
```


1
2
3
2800
2600
2400
2200
2300
2200
2100
3000
2000
1000
0



- from correlation heat map bivariate analysis we can say that most of the signal has most of the signal has correalation between +.10 to -.40
- because we have removed most of the colinearity of the feature
- very few feature show very high positve as well as negative correlation.
- in original data where removal of feature was not perfomed based on their colinearity some signal has high positive as well as negative correlations
- from scatter plot most of the signal shows no correlation
- and their distribution is also varies some has normal dist, some are skewed, and some are bimodel
- signal '4' has high spikes at arround value zero slighly variation almost neglegible with other signal it almost showing zero value

4. Data pre-processing: [10 Marks]

4 A. Segregate predictors vs target attributes. [2 Marks]

In [652…
```
data.shape
```

Out[652]:
```
(1567, 234)
```

In [653…
```
X=data.iloc[:,0:233]
Y=data.iloc[:,233]
test_size=0.30
seed=7
#getting the shapes of new data sets x and y
X_train,X_test,y_train,y_test=train_test_split(X,Y,test_size=0.30,random_state=1)
```

```
print("shape of x:", X.shape)
print("shape of y:", Y.shape)
type(X_train)
```

```
shape of x: (1567, 233)
shape of y: (1567,)
pandas.core.frame.DataFrame
```

Out[653]:

4 B. Check for target balancing and fix it if found imbalanced. [3 Marks]

In [654…

```
data['Pass/Fail'].value_counts()
```

Out[654]:

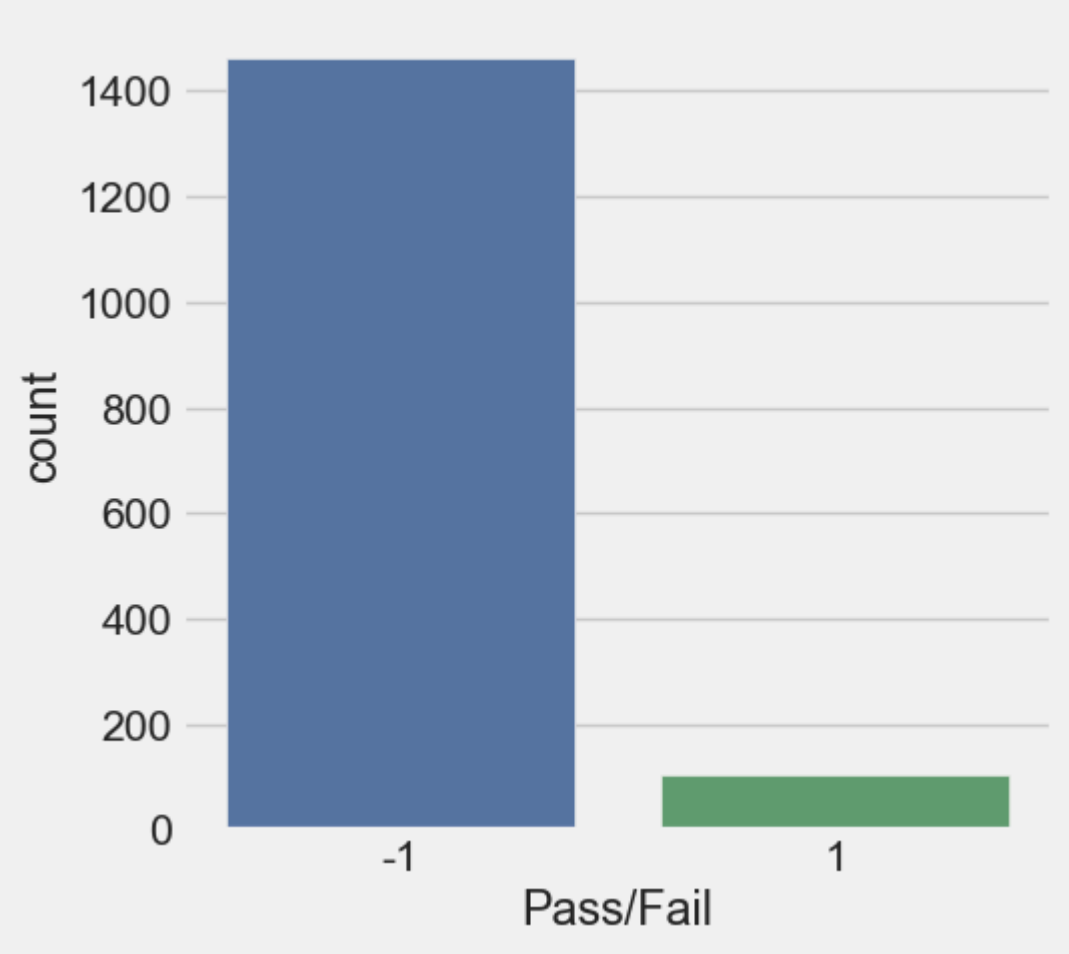
```
Pass/Fail
-1    1463
 1     104
Name: count, dtype: int64
```

In [655…

```
plt.figure(figsize=(5,5))
sns.countplot(x=Y)
```

Out[655]:

```
<Axes: xlabel='Pass/Fail', ylabel='count'>
```



In [656…

```
# data shows highly imbalance
```

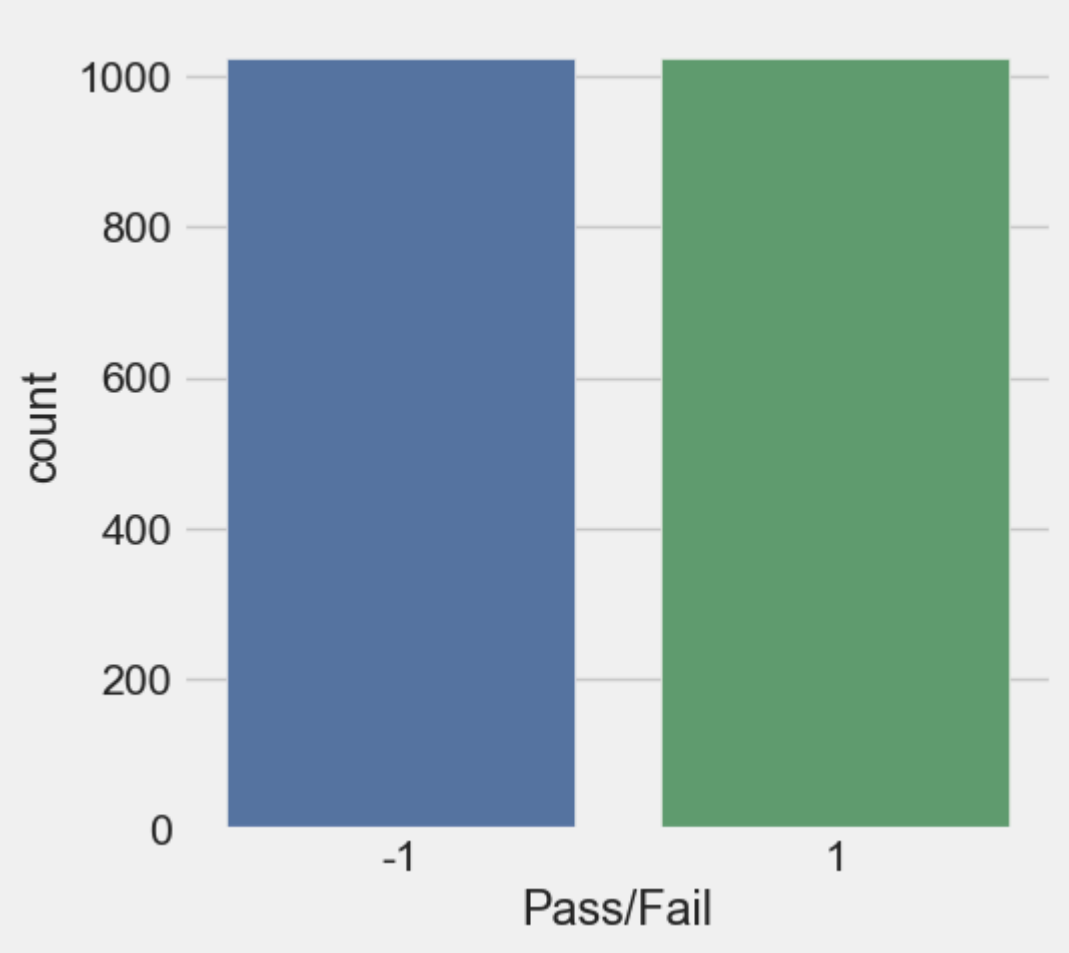
In [657…

```
from imblearn.over_sampling import SMOTE
sm=SMOTE(sampling_strategy=1, k_neighbors=5,random_state=1)
X_train_res, y_train_res  = sm.fit_resample(X_train, y_train)

print(X_train_res.shape)
print(y_train_res.shape)
```

```
(2048, 233)
(2048,)
```

In [658…
```
plt.figure(figsize=(5,5))
sns.countplot(x=y_train_res)
```

Out[658]:
```
<Axes: xlabel='Pass/Fail', ylabel='count'>
```



Though this algorithm is quite useful, it has few drawbacks associated with it.

The synthetic instances generated are in the same direction i.e. connected by an artificial line its diagonal instances. This in turn complicates the decision surface generated by few classifier algorithms. SMOTE tends to create a large no. of noisy data points in feature space.

4 C. Perform train-test split and standardise the data or vice versa if required. [3 Marks]

please note we have alredy perfom train test split as above

In [659…
```
sc = StandardScaler()
X_train_os = sc.fit_transform(X_train_res)
X_test_os = sc.transform(X_test)
```

4 D. Check if the train and test data have similar statistical characteristics when compared with original data. [2 Marks]

- lets perform statistical anlysis that is 5point summary for train and test data and original data
- here we use original data which is obtain after colenear fiture and collumn with similar data

In [660…
```
data.describe()
```

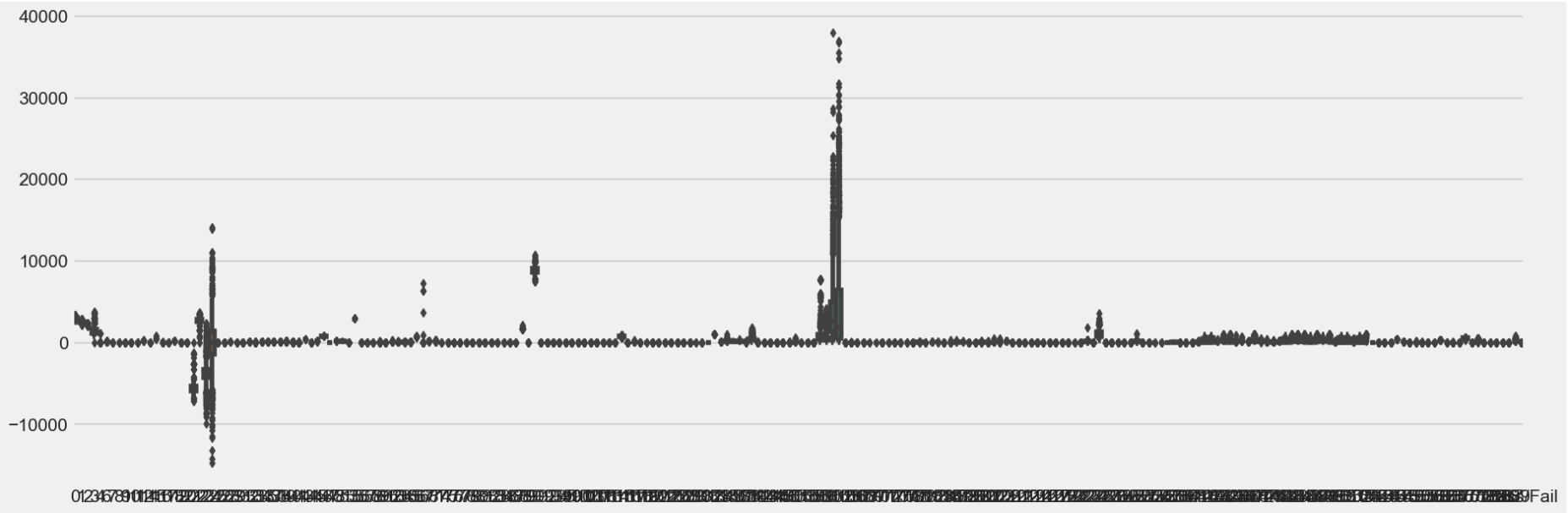
Out[660]:

	0	1	2	3	4	6	7	8	9	10	...	57
count	1567.000000	1567.000000	1567.000000	1567.000000	1567.000000	1567.000000	1567.000000	1567.000000	1567.000000	1567.000000	...	1567.00000
mean	3014.452896	2495.850231	2200.547318	1396.376627	4.197013	101.112908	0.121822	1.462862	-0.000841	0.000146	...	530.52362
std	73.480613	80.227793	29.380932	439.712852	56.103066	6.209271	0.008936	0.073849	0.015107	0.009296	...	17.49973
min	2743.240000	2158.750000	2060.660000	0.000000	0.681500	82.131100	0.000000	1.191000	-0.053400	-0.034900	...	317.19640
25%	2966.665000	2452.885000	2181.099950	1083.885800	1.017700	97.937800	0.121100	1.411250	-0.010800	-0.005600	...	530.70270
50%	3011.840000	2498.910000	2200.955600	1287.353800	1.317100	101.492200	0.122400	1.461600	-0.001300	0.000400	...	532.39820
75%	3056.540000	2538.745000	2218.055500	1590.169900	1.529600	104.530000	0.123800	1.516850	0.008400	0.005900	...	534.35640
max	3356.350000	2846.440000	2315.266700	3715.041700	1114.536600	129.252200	0.128600	1.656400	0.074900	0.053000	...	589.50820

8 rows × 234 columns

In [661…
```
plt.figure(figsize=(20,7))
sns.boxplot(data)
```

Out[661]:
```
<Axes: >
```



In [662…

```
pd.DataFrame(X_train_os).describe()
```

Out[662]:

	0	1	2	3	4	5	6	7	8	
count	2.048000e+03	2.048000e+03	2.048000e+03	2.048000e+03	2.048000e+03	2.048000e+03	2.048000e+03	2.048000e+03	2.048000e+03	2.048000
mean	7.268491e-15	1.000068e-15	1.008395e-14	2.775558e-17	-3.469447e-18	-2.220446e-15	2.775558e-17	-2.213507e-15	-6.938894e-18	-1.38777
std	1.000244e+00	1.000244e+00	1.000244e+00	1.000244e+00	1.000244e+00	1.000244e+00	1.000244e+00	1.000244e+00	1.000244e+00	1.000244
min	-3.121846e+00	-4.905133e+00	-5.278793e+00	-1.788041e+00	-5.381020e-02	-3.612093e+00	-2.157503e+01	-4.455161e+00	-3.824206e+00	-4.221082
25%	-6.823224e-01	-5.645747e-01	-6.975679e-01	-7.014650e-01	-4.423329e-02	-4.854499e-01	-1.710410e-01	-6.431852e-01	-5.980097e-01	-5.55966
50%	-1.087828e-01	2.439124e-02	-1.377074e-02	-2.364089e-01	-3.882245e-02	3.541931e-04	2.149459e-02	-8.283764e-03	7.349063e-03	5.02197
75%	5.757058e-01	5.535281e-01	5.863220e-01	5.180836e-01	-3.447138e-02	4.731242e-01	2.597016e-01	6.761124e-01	5.313942e-01	6.07424
max	3.767279e+00	4.539654e+00	3.979068e+00	6.191149e+00	2.614547e+01	5.179593e+00	9.592028e-01	2.952057e+00	5.760762e+00	4.535717

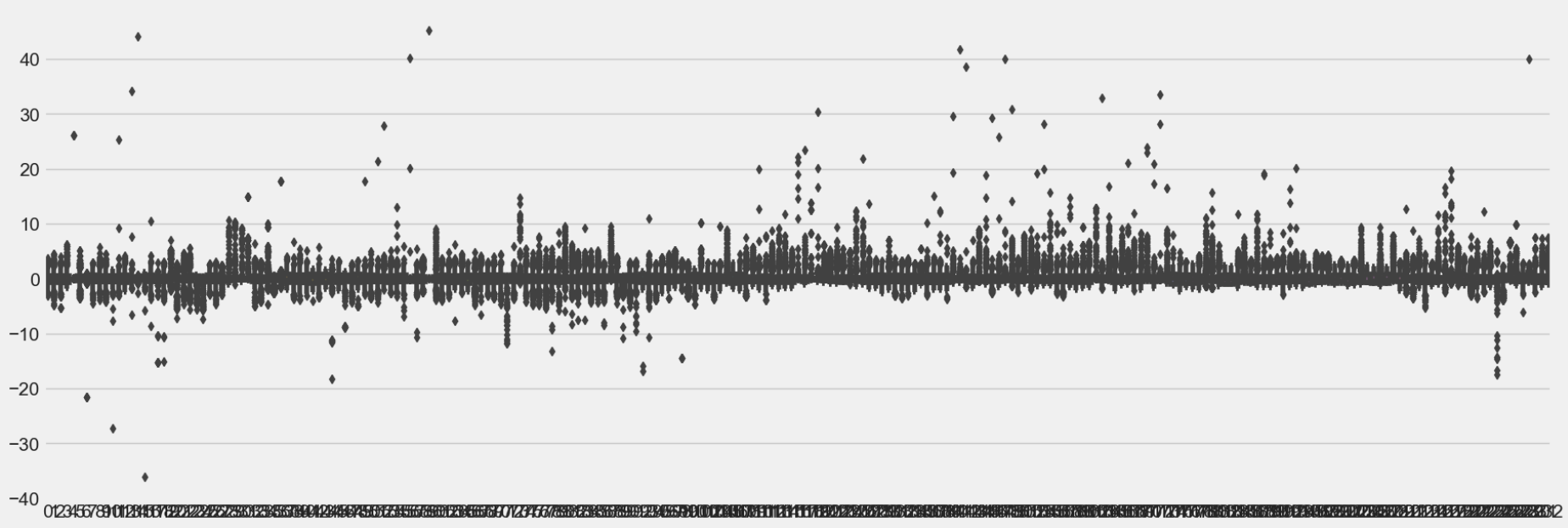
8 rows × 233 columns

In [663…

```
plt.figure(figsize=(20,7))
sns.boxplot(X_train_os)
```

Out[663]:

```
<Axes: >
```



In [664…

```
pd.DataFrame(data=X_test_os).describe()
```

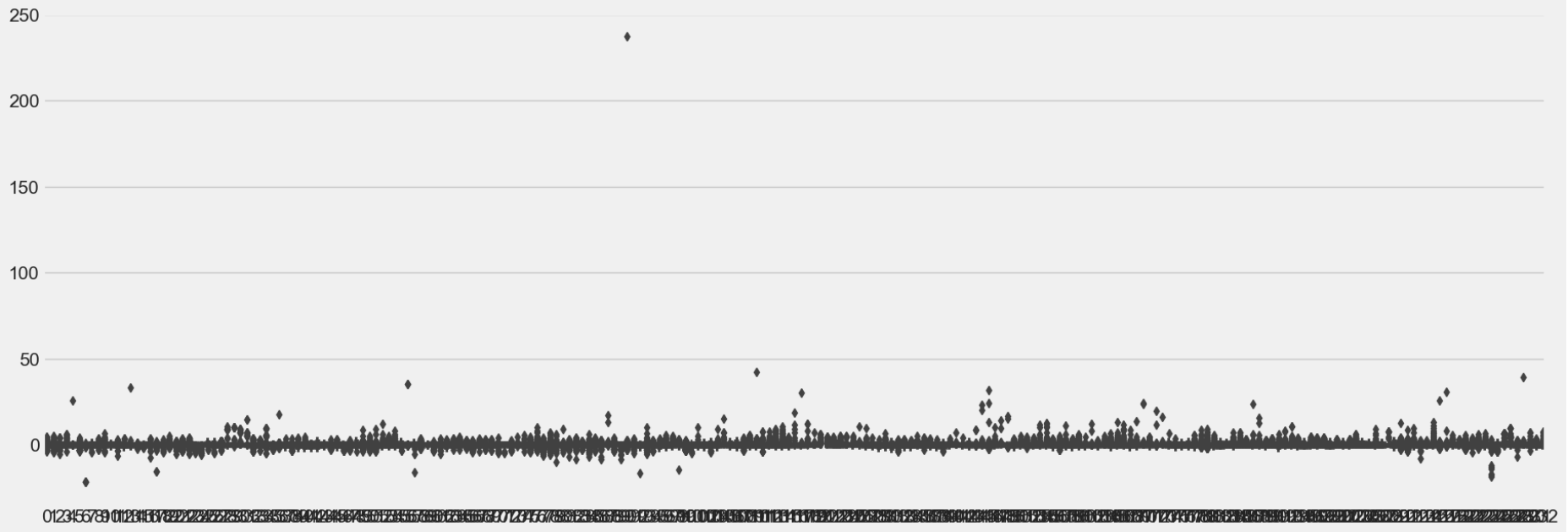
Out[664]:

	0	1	2	3	4	5	6	7	8	9	...	223	22
count	471.000000	471.000000	471.000000	471.000000	471.000000	471.000000	471.000000	471.000000	471.000000	471.000000	...	471.000000	471.00000
mean	0.013681	-0.026480	0.033869	0.096674	0.017290	-0.132894	-0.131497	-0.176113	0.102726	-0.001303	...	-0.133942	-0.17687
std	1.060428	1.199157	1.127277	1.232473	1.204304	1.133279	2.016284	1.180037	1.080532	1.287898	...	0.981946	1.80378
min	-3.737219	-4.509141	-5.278793	-3.738529	-0.052865	-3.590524	-21.575032	-4.303961	-2.882892	-4.473077	...	-1.684252	-18.40268
25%	-0.600692	-0.591999	-0.712244	-0.763861	-0.046782	-0.716427	-0.188701	-0.983923	-0.608049	-0.819160	...	-0.864274	-0.09077
50%	-0.069871	0.038311	0.055387	-0.240117	-0.039335	0.000232	0.040880	-0.204048	0.097936	-0.000179	...	-0.237613	0.04337
75%	0.553012	0.604894	0.694832	0.644540	-0.034260	0.518093	0.288121	0.700766	0.751626	0.718004	...	0.093882	0.21621
max	4.789147	5.066289	4.399313	6.452582	26.096929	4.092263	1.135803	2.415694	3.781014	6.602069	...	6.939796	3.29232

8 rows × 233 columns

In [665…
```
plt.figure(figsize=(20,7))
sns.boxplot(X_test_os)
```

Out[665]:
```
<Axes: >
```



- when statistical data is comapred with original data it is evident their it is change due to application of standard scaler
- the effect of one dominating feature can be seen almost reduced in satndardiztion of data.
- where as train and test appears to be expanded or shrinked view of each other

5. Model training, testing and tuning: [20 Marks]

5 A. Use any Supervised Learning technique to train a model. [2 Marks]

RANDOM FOREST - (OVERSAMPLE DATA)

In [666…
```
#model = LogisticRegression()
model_Ran_Os = RandomForestClassifier()
```

```
# model = KNeighborsClassifier(n_neighbors=7)
model_Ran_Os.fit(X_train_os, y_train_res)
#scores_prediction = model.decision_function(x_train)
y_pred_Ran_Os = model_Ran_Os.predict(X_test_os)
print("Accuracy on train :",model_Ran_Os.score(X_train_os,y_train_res)*100)
print("Accuracy on test  :",model_Ran_Os.score(X_test_os,y_test)*100)
```

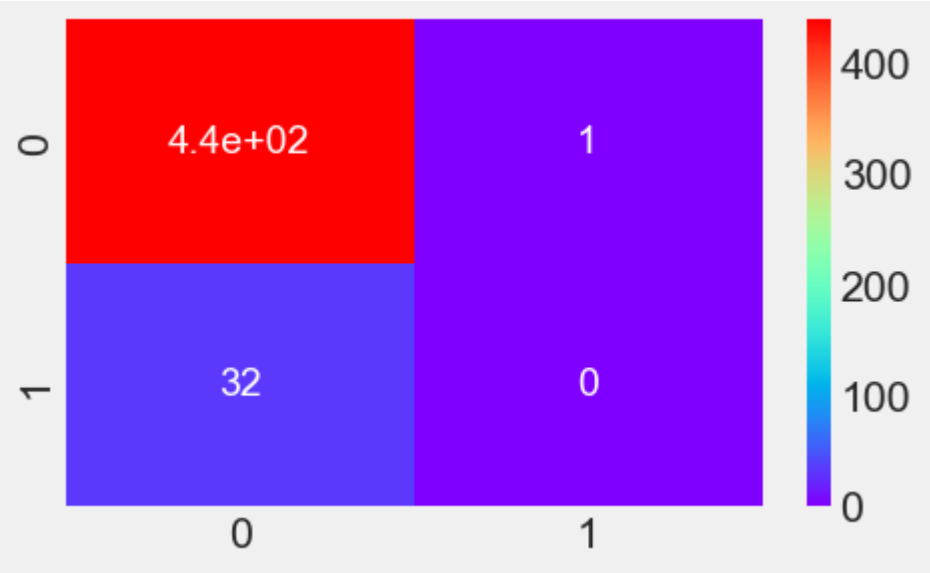
```
Accuracy on train : 100.0
Accuracy on test  : 92.99363057324841
```

In [667…

```
# printing the confusion matrix
cm = confusion_matrix(y_test, y_pred_Ran_Os)
plt.figure(figsize=(5,3))
sns.heatmap(cm, annot = True, cmap = 'rainbow')
```

Out[667]:

```
<Axes: >
```



In [668…

```
print(classification_report(y_test,y_pred_Ran_Os))
```

```
              precision    recall  f1-score   support

          -1       0.93      1.00      0.96       439
           1       0.00      0.00      0.00        32

    accuracy                           0.93       471
   macro avg       0.47      0.50      0.48       471
weighted avg       0.87      0.93      0.90       471
```

logistic Regression-(oversample data)

In [669…

```
model_Lr_Os = LogisticRegression()
#model = RandomForestClassifier()
#model= KNeighborsClassifier(n_neighbors=7)
model_Lr_Os.fit(X_train_os, y_train_res)
#scores_prediction = model.decision_function(x_train)
y_pred_Lr_Os = model_Lr_Os.predict(X_test_os)
print("Accuracy on train :",model_Lr_Os.score(X_train_os,y_train_res)*100)
print("Accuracy on test  :",model_Lr_Os.score(X_test_os,y_test)*100)
```

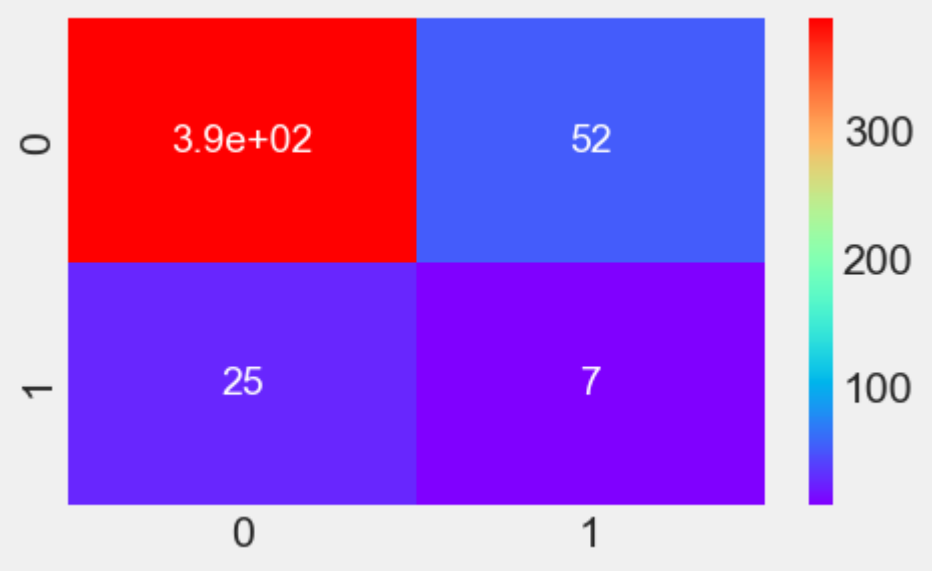
```
Accuracy on train : 99.12109375
Accuracy on test  : 83.65180467091295
```

In [670…

```
# printing the confusion matrix
cm = confusion_matrix(y_test, y_pred_Lr_Os)
plt.figure(figsize=(5,3))
sns.heatmap(cm, annot = True, cmap = 'rainbow')
```

Out[670]:

```
<Axes: >
```



In [671…

```
print(classification_report(y_test,y_pred_Lr_Os))
```

```
              precision    recall  f1-score   support

          -1       0.94      0.88      0.91       439
           1       0.12      0.22      0.15        32

    accuracy                           0.84       471
   macro avg       0.53      0.55      0.53       471
weighted avg       0.88      0.84      0.86       471
```

KNNeighbour Classifier(Over sample Data)

In [672…

```
#model = LogisticRegression()
#model = RandomForestClassifier()
model_Knn_Os = KNeighborsClassifier(n_neighbors=5)

model_Knn_Os.fit(X_train_os, y_train_res)
#scores_prediction = model.decision_function(x_train)
y_pred_Knn_Os = model_Knn_Os.predict(X_test_os)
print("Accuracy on train :",model_Knn_Os.score(X_train_os,y_train_res)*100)
print("Accuracy on test  :",model_Knn_Os.score(X_test_os,y_test)*100)
```

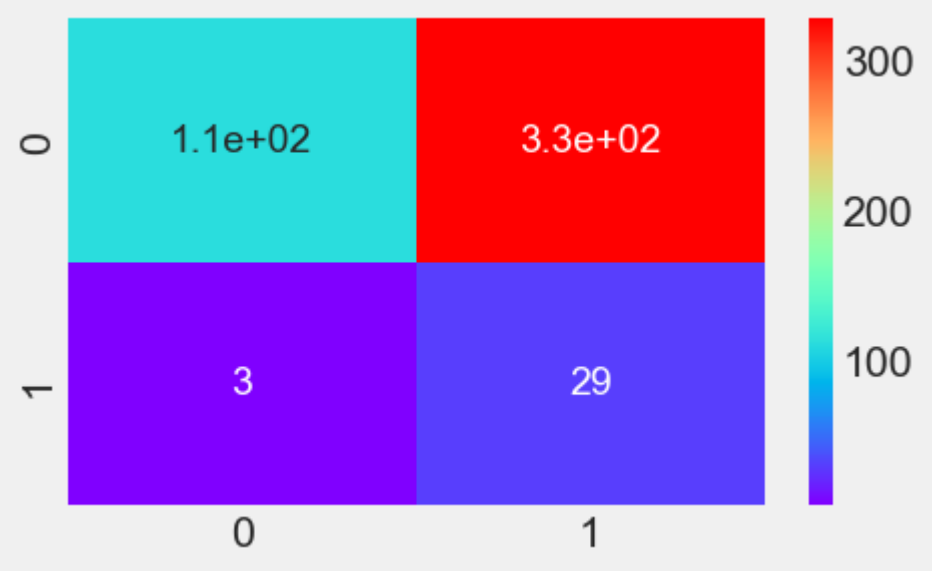
```
Accuracy on train : 68.994140625
Accuracy on test  : 29.723991507431
```

In [673…

```
# printing the confusion matrix
cm = confusion_matrix(y_test, y_pred_Knn_Os)
plt.figure(figsize=(5,3))
sns.heatmap(cm, annot = True, cmap = 'rainbow')
```

Out[673]:

```
<Axes: >
```



In [674…

```
print(classification_report(y_test,y_pred_Knn_Os))
```

```
              precision    recall  f1-score   support

          -1       0.97      0.25      0.40       439
           1       0.08      0.91      0.15        32

    accuracy                           0.30       471
   macro avg       0.53      0.58      0.28       471
weighted avg       0.91      0.30      0.38       471
```

5B. Use cross validation techniques. [3 Marks]

Hint: Use all CV techniques that you have learnt in the course.

Logistic regression - Cross Validation - (Normal data)

In [675…

```
num_fold = 19
seed=7
kfold=KFold(n_splits=num_fold,shuffle=True)
model_Lr_Cv = LogisticRegression()
#model = RandomForestClassifier()
# model = KNeighborsClassifier(n_neighbors=5)
results=cross_val_score(model_Lr_Cv,X,Y,cv=kfold)
print(results)
print("Accuracy: %.3f%% (%.3f%%)" % (results.mean()*100.0,results.std()*100.0))
```

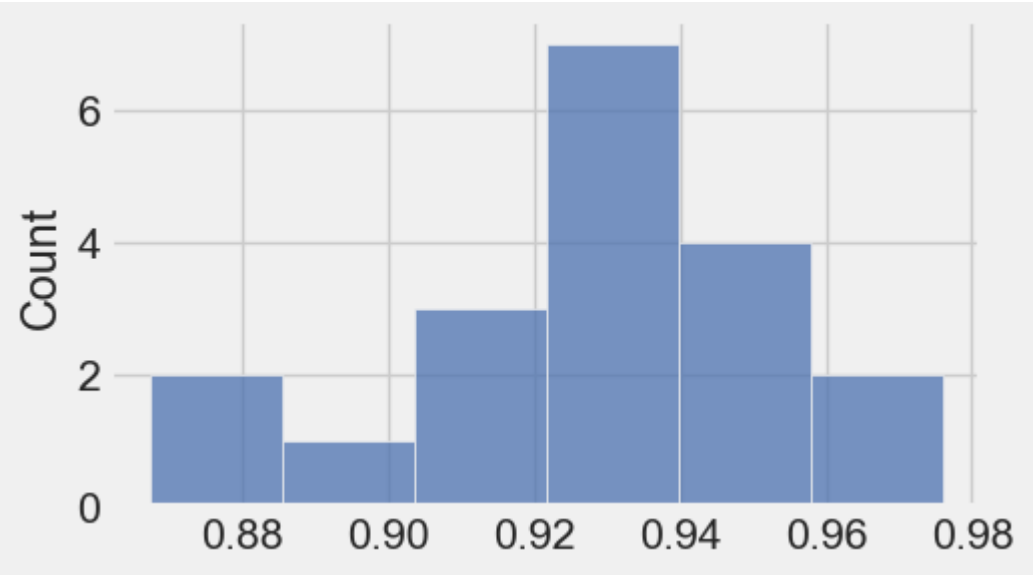
```
[0.96385542 0.91566265 0.95180723 0.95180723 0.92771084 0.97590361
 0.91566265 0.86746988 0.95180723 0.91463415 0.87804878 0.93902439
 0.92682927 0.92682927 0.95121951 0.92682927 0.93902439 0.92682927
 0.90243902]
Accuracy: 92.913% (2.650%)
```

In [676…

```
plt.figure(figsize=(5,3))
sns.histplot(results)
```

Out[676]:

```
<Axes: ylabel='Count'>
```



In [ ]:

Randomforest - Cross Validation - (over sampled data)

In [678…

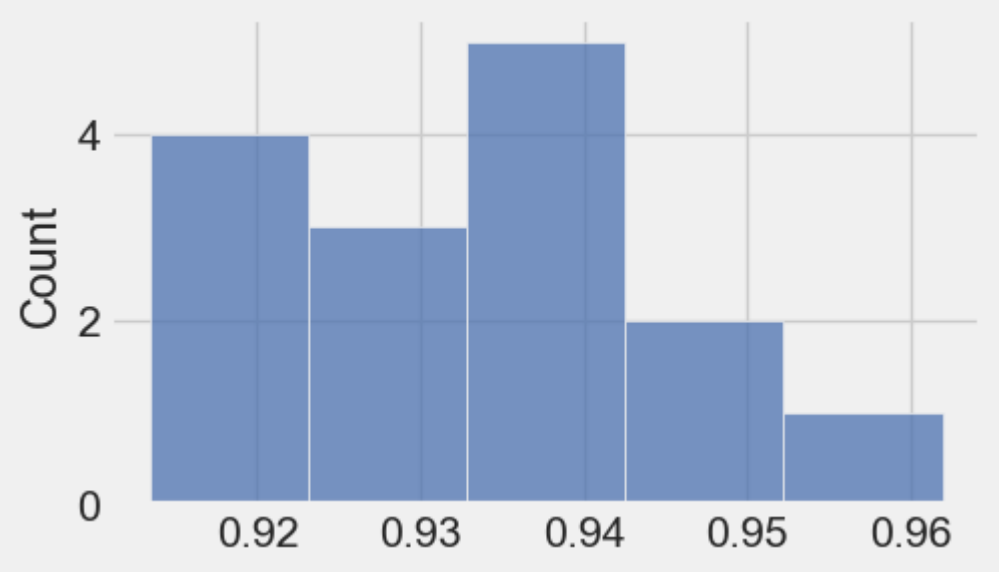
```
num_fold = 15
seed=7
kfold=KFold(n_splits=num_fold,shuffle=True)
# model = LogisticRegression()
model_Ran_Cv = RandomForestClassifier()
# model = KNeighborsClassifier(n_neighbors=5)
results=cross_val_score(model_Ran_Cv,X,Y,cv=kfold)
print(results)
print("Accuracy: %.3f%% (%.3f%%)" % (results.mean()*100.0,results.std()*100.0))
```

```
[0.91428571 0.93333333 0.96190476 0.91428571 0.91428571 0.93333333
 0.92380952 0.94230769 0.94230769 0.93269231 0.94230769 0.95192308
 0.93269231 0.91346154 0.95192308]
Accuracy: 93.366% (1.488%)
```

- with cross validation with k fold we can expact accuracy ranging from 90% to 97% with 95% of confidence level

In [679…
```
plt.figure(figsize=(5,3))
sns.histplot(x=results)
```

Out[679]:
```
<Axes: ylabel='Count'>
```



- lets try leeave one out cross validation sampling method

Logistic regression (LOOCV) Leave One Out Cross Validation- oversample balance data

In [680…
```
loocv=LeaveOneOut()
#num_fold = 100
#seed=7
#kfold=KFold(n_splits=num_fold,shuffle=True)
model_Lr_Looc=LogisticRegression()
#model=RandomForestClassifier()
results=cross_val_score(model_Lr_Looc,X,Y,cv=loocv)
#print(results)
print("Accuracy: %.3f%% (%.3f%%)" % (results.mean()*100.0,results.std()*100.0))
```

```
Accuracy: 93.172% (25.223%)
```

- clearly data leak has happened

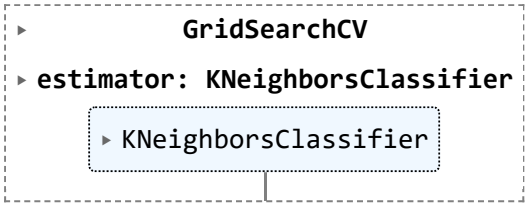
5C. Apply hyper-parameter tuning techniques to get the best accuracy. [3 Marks]

Suggestion: Use all possible hyper parameter combinations to extract the best accuracies.

Kneighbours classifier (oversample) (grid search)

In [681…
```
knn_clf_Gs=KNeighborsClassifier()
list=np.arange(1,9)
param_grid ={'n_neighbors': list,
             'algorithm': ('auto','ball_tree','kd_tree','brute')}
Knn_gs=GridSearchCV(knn_clf_Gs,param_grid,cv=10)
Knn_gs.fit(X_train_os,y_train_res)
```

Out[681]:



In [682…
```
Knn_gs.best_params_
```

Out[682]:
```
{'algorithm': 'auto', 'n_neighbors': 2}
```

In [683…
```
gs.cv_results_['params']
```

Out[683]:
```
[{'algorithm': 'auto', 'n_neighbors': 1},
 {'algorithm': 'auto', 'n_neighbors': 2},
 {'algorithm': 'auto', 'n_neighbors': 3},
 {'algorithm': 'auto', 'n_neighbors': 4},
 {'algorithm': 'auto', 'n_neighbors': 5},
 {'algorithm': 'auto', 'n_neighbors': 6},
 {'algorithm': 'auto', 'n_neighbors': 7},
 {'algorithm': 'auto', 'n_neighbors': 8},
 {'algorithm': 'ball_tree', 'n_neighbors': 1},
 {'algorithm': 'ball_tree', 'n_neighbors': 2},
 {'algorithm': 'ball_tree', 'n_neighbors': 3},
 {'algorithm': 'ball_tree', 'n_neighbors': 4},
 {'algorithm': 'ball_tree', 'n_neighbors': 5},
 {'algorithm': 'ball_tree', 'n_neighbors': 6},
 {'algorithm': 'ball_tree', 'n_neighbors': 7},
 {'algorithm': 'ball_tree', 'n_neighbors': 8},
 {'algorithm': 'kd_tree', 'n_neighbors': 1},
 {'algorithm': 'kd_tree', 'n_neighbors': 2},
 {'algorithm': 'kd_tree', 'n_neighbors': 3},
 {'algorithm': 'kd_tree', 'n_neighbors': 4},
 {'algorithm': 'kd_tree', 'n_neighbors': 5},
 {'algorithm': 'kd_tree', 'n_neighbors': 6},
 {'algorithm': 'kd_tree', 'n_neighbors': 7},
 {'algorithm': 'kd_tree', 'n_neighbors': 8},
 {'algorithm': 'brute', 'n_neighbors': 1},
 {'algorithm': 'brute', 'n_neighbors': 2},
 {'algorithm': 'brute', 'n_neighbors': 3},
 {'algorithm': 'brute', 'n_neighbors': 4},
 {'algorithm': 'brute', 'n_neighbors': 5},
 {'algorithm': 'brute', 'n_neighbors': 6},
 {'algorithm': 'brute', 'n_neighbors': 7},
 {'algorithm': 'brute', 'n_neighbors': 8}]
```

In [684…
```
list_of_scores=Knn_gs.cv_results_['mean_test_score']
list_of_scores
```

Out[684]:
```
array([0.75829986, 0.78662123, 0.66991153, 0.6816308 , 0.62061454,
       0.6313582 , 0.59083931, 0.59962936, 0.75829986, 0.78662123,
       0.66991153, 0.6816308 , 0.62061454, 0.6313582 , 0.59083931,
       0.59962936, 0.75829986, 0.78662123, 0.66991153, 0.6816308 ,
       0.62061454, 0.6313582 , 0.59083931, 0.59962936, 0.75829986,
       0.78662123, 0.66991153, 0.6816308 , 0.62061454, 0.6313582 ,
       0.59083931, 0.59962936])
```

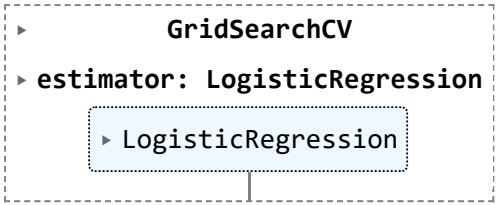
In [685…
```
list_of_scores.max()
```

Out[685]:
```
0.7866212338593974
```

logistic regression (oversample) (grid search CV)

In [686…
```
Log_clf=LogisticRegression()
list=np.arange(1,9)
param_grid={"C":np.logspace(-3,3,7)}
gs_lr=GridSearchCV(Log_clf,param_grid,cv=10)
gs_lr.fit(X_train_os,y_train_res)
```

Out[686]:



In [687…
```
gs_lr.best_params_
```

Out[687]:
```
{'C': 10.0}
```

In [688…
```
gs_lr.cv_results_['params']
```

Out[688]:
```
[{'C': 0.001},
 {'C': 0.01},
 {'C': 0.1},
 {'C': 1.0},
 {'C': 10.0},
 {'C': 100.0},
 {'C': 1000.0}]
```

In [689…
```
list_of_scores=gs_lr.cv_results_['mean_test_score']
list_of_scores
```

Out[689]:
```
array([0.87159971, 0.9091846 , 0.93066475, 0.94189861, 0.94483022,
       0.94483022, 0.94385701])
```

In [690…
```
gs_lr.cv_results_['mean_test_score'].max()
```

Out[690]: 0.9448302247728358

In [691…
```
y_pred_lr_gs = gs_lr.predict(X_test_os)
#print("Accuracy of logistic regression oversample greed search:",(y_test_res,y_pred_lr_gs))
print(classification_report(y_test,y_pred_lr_gs))
```

```
              precision    recall  f1-score   support

          -1       0.94      0.90      0.92       439
           1       0.14      0.22      0.17        32

    accuracy                           0.85       471
   macro avg       0.54      0.56      0.54       471
weighted avg       0.89      0.85      0.87       471
```

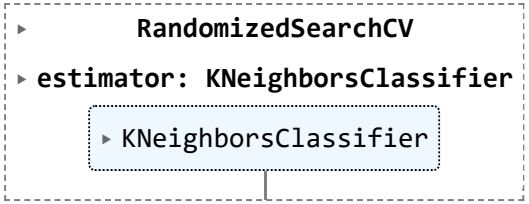
Knneighbors Clssifier (random search CV)(oversampled data)

In [692…
```
# specify parameters and distributions to samle from
param_dist= {'n_neighbors': list,
             'algorithm' : ('auto', 'ball_tree', 'kd_tree', 'brute'),
              #'weights':[1,2,3,4,5],
               #'p':2,
               #'metric':['Euclidean'],
               #'metric_params': None,
                 #'n_jobs':2,
              'weights': ['uniform', 'distance'],
              'p': [1, 2]}
```

In [693…
```
from sklearn.model_selection import RandomizedSearchCV
sample=10
randomCV=RandomizedSearchCV(knn_clf_Gs,param_distributions=param_dist,n_iter=sample)
```

In [694…
```
randomCV.fit(X_train_os,y_train_res)
```

Out[694]:



In [695…
```
randomCV.best_params_
```

Out[695]:
```
{'weights': 'distance', 'p': 2, 'n_neighbors': 1, 'algorithm': 'auto'}
```

In [696…
```
randomCV.cv_results_['mean_test_score']
```

Out[696]:
```
array([0.62208122, 0.5927855 , 0.74610054, 0.71681317, 0.58106506,
       0.65723418, 0.61084978, 0.61084978, 0.58106506, 0.66992546])
```

In [697…
```
randomCV.cv_results_['mean_test_score'].max()
```

Out[697]:
```
0.7461005426680185
```

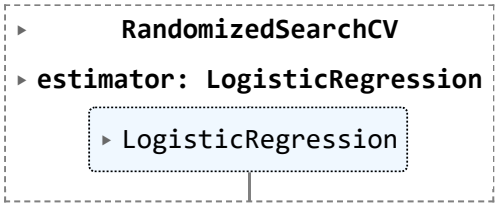
LogisticRegression (random search CV) (oversample Data)

In [698…
```
param_dist_lr={
    'C' : np.logspace(-4, 4, 20),
    'solver' : ['lbfgs','newton-cg','liblinear','sag','saga'],
    'max_iter' : [100, 1000,2500, 5000]}
```

In [699…
```
from sklearn.model_selection import RandomizedSearchCV
sample=10
randomCV_lr=RandomizedSearchCV(Log_clf,param_distributions=param_dist_lr,n_iter=sample)
```

In [700…
```
randomCV_lr.fit(X_train_os,y_train_res)
```

Out[700]:



In [701…
```
randomCV_lr.best_params_
```

Out[701]:
```
{'solver': 'lbfgs', 'max_iter': 100, 'C': 10000.0}
```

In [702…
```
randomCV_lr.cv_results_['mean_test_score']
```

Out[702]:
```
array([0.8564494 , 0.93457451, 0.9233383 , 0.93017592, 0.91943467,
       0.93212833, 0.93213191, 0.93408551, 0.93457451, 0.93750611])
```

In [703…
```
randomCV_lr.cv_results_['mean_test_score'].max()
```

Out[703]:
```
0.9375061124694376
```

5 D. Use any other technique/method which can enhance the model performance. [4 Marks]

- Hint: Dimensionality reduction, attribute removal,

lets perform PCA for dimentinality reduction

In [704…
```
data_new.shape
```

Out[704]:
```
(1567, 233)
```

In [705…
```
#Scaling the data before applying PCA
from scipy.stats import zscore
data_new=data.iloc[:,:233].apply(zscore)
data_new.head()
```

Out[705]:

	0	1	2	3	4	6	7	8	9	10	...	565	570	571	57
0	0.224309	0.849725	-0.436273	0.033555	-0.050580	-0.563790	0.266269	0.509826	1.128417	-0.381543	...	0.000000	0.190142	0.034410	-0.22601
1	1.107136	-0.382910	1.017137	0.153067	-0.060045	0.198217	0.322244	0.456999	0.022582	-1.608247	...	0.000000	0.256816	1.205944	-0.26113
2	-1.114158	0.799102	-0.481289	0.686213	-0.047906	-0.906210	0.255074	-0.260907	0.327183	0.124204	...	6.463483	0.257279	-0.263745	-0.19982
3	-0.350312	-0.198875	-0.051547	-1.106948	-0.051290	0.503246	-0.013602	0.343218	-0.765408	-0.370782	...	0.235997	0.002548	-0.278290	-0.22161
4	0.242143	0.087526	1.117387	-0.158919	-0.047492	-0.115382	0.187905	0.545044	-0.149584	-0.790444	...	0.000000	0.085279	-0.270290	-0.22740

5 rows × 233 columns

In [706…]:
```
data_new.isnull().any().any()
```

Out[706]:
```
False
```

In [707…]:
```
X = data_new.iloc[:,:233]
Y = data["Pass/Fail"]

# getting the shapes of new data sets x and y
print("shape of x:", X.shape)
print("shape of y:", Y.shape)
```

```
shape of x: (1567, 233)
shape of y: (1567,)
```

In [708…]:
```
# PCA
# Step 1 - Create covariance matrix

cov_matrix = np.cov(X.T)
print('Covariance Matrix \n%s', cov_matrix)
```

```
Covariance Matrix
%s [[ 1.00063857 -0.14393166  0.00475868 ...  0.01845493 -0.02589702
   0.00417663]
 [-0.14393166  1.00063857  0.00577089 ... -0.00940915  0.01727747
   0.04482545]
 [ 0.00475868  0.00577089  1.00063857 ... -0.02551161 -0.02936364
  -0.03291098]
 ...
 [ 0.01845493 -0.00940915 -0.02551161 ...  1.00063857  0.16801987
  -0.48686973]
 [-0.02589702  0.01727747 -0.02936364 ...  0.16801987  1.00063857
   0.39106264]
 [ 0.00417663  0.04482545 -0.03291098 ... -0.48686973  0.39106264
   1.00063857]]
```

In [709…

```python
# Step 2- Get eigen values and eigen vector
eig_vals, eig_vecs = np.linalg.eig(cov_matrix)
print('Eigen Vectors \n%s', eig_vecs)
print('\n Eigen Values \n%s', eig_vals)
```

```
Eigen Vectors
%s [[-0.0415156   0.01988424 -0.01532755 ...  0.00531248 -0.06137769
  -0.00737086]
 [ 0.0044108   0.00150383  0.02144423 ...  0.00545263 -0.00961127
   0.01879874]
 [ 0.01073471  0.00380943  0.00500601 ...  0.08345242  0.086805
   0.12560322]
 ...
 [-0.0448755   0.02825638  0.01477176 ...  0.0098863   0.02413497
   0.01287882]
 [ 0.01676349 -0.02500915 -0.01455108 ... -0.05240327 -0.03487063
  -0.00283998]
 [ 0.02538257 -0.0411165   0.01217098 ... -0.00669265  0.06798295
   0.03088317]]

 Eigen Values
%s [8.74809339e+00 6.66962302e+00 5.50955678e+00 5.22584832e+00
 4.35773117e+00 3.97072403e+00 3.81897083e+00 3.62197791e+00
 3.53379893e+00 3.31502557e+00 3.19430133e+00 3.07774981e+00
 2.88243194e+00 2.85804781e+00 2.71289610e+00 2.57643757e+00
 2.53918611e+00 2.45459984e+00 2.01545843e-04 1.13289694e-03
 2.37289211e+00 2.34520123e+00 2.31227101e+00 2.24293432e+00
 2.21223382e+00 2.15206988e+00 2.13775542e+00 2.07410321e+00
 2.05703566e+00 2.01912315e+00 1.99225195e+00 1.94230979e+00
 1.92180993e+00 1.88534222e+00 1.77905056e+00 1.84540223e+00
 1.83079465e+00 1.83438983e+00 1.75068791e+00 1.73571581e+00
 1.70816153e+00 1.69779384e+00 1.67449827e+00 1.66410548e+00
 1.63049305e+00 1.60686663e+00 1.57330047e+00 1.56254820e+00
 1.54209626e+00 1.52814895e+00 1.49778541e+00 1.49145497e+00
 1.46916193e+00 1.46171385e+00 6.12476174e-03 1.41705481e+00
 1.39367024e+00 1.36757239e+00 1.34234361e+00 1.30786944e+00
 1.31596080e+00 1.29028864e+00 1.28242129e+00 1.24780621e+00
 1.24603225e+00 2.76141174e-03 1.21924114e+00 1.20983002e+00
 1.20792385e+00 1.19597731e+00 1.17815219e+00 1.15073294e+00
 1.16285023e+00 1.16265989e+00 1.12676957e+00 1.11790798e+00
 1.11975125e+00 1.10472968e+00 1.08430168e+00 1.07927807e+00
 1.06684855e+00 1.04208758e+00 1.03546112e+00 1.02763262e+00
 1.01874587e+00 4.72128787e-03 1.76218944e-02 1.62327721e-02
 1.00095472e+00 9.81877326e-01 9.71198984e-01 9.63049155e-01
 9.57107931e-01 9.50182430e-01 9.39556502e-01 9.36414361e-01
 3.89419500e-02 2.25301364e-02 2.94142362e-02 2.84642450e-02
 2.62806840e-02 2.66320395e-02 9.24931199e-01 9.19293190e-01
 9.01839373e-01 5.00808131e-02 9.08111187e-01 8.86693569e-01
 8.85263596e-01 8.66799657e-01 8.79235234e-01 8.51465221e-01
```

```
5.79238946e-02 6.40754463e-02 6.29941704e-02 6.90299266e-02
8.31145469e-01 8.28247593e-01 8.21302055e-01 8.11939615e-01
7.98935470e-01 7.85874788e-01 7.31139847e-02 7.58512485e-02
8.50493562e-02 1.09552995e-01 9.06717097e-02 9.26635348e-02
9.77760390e-02 9.87420312e-02 7.74065015e-01 7.71176129e-01
7.59720546e-01 7.55333238e-01 1.20875903e-01 1.24800856e-01
1.26737249e-01 1.37729925e-01 1.40402564e-01 1.46768852e-01
7.32224918e-01 7.40184335e-01 7.48026371e-01 7.46650170e-01
7.13128671e-01 7.17349932e-01 6.95689831e-01 1.48940625e-01
1.53567185e-01 1.55219270e-01 1.58307759e-01 1.65561156e-01
1.64938051e-01 1.76458113e-01 1.81689071e-01 1.85455652e-01
7.20703447e-01 6.89981314e-01 6.39631892e-01 6.78537450e-01
6.58024994e-01 6.61063442e-01 6.68158227e-01 6.70817220e-01
6.33163526e-01 6.25288186e-01 6.15964890e-01 1.92687952e-01
2.02166881e-01 2.00055143e-01 6.10415924e-01 6.00242805e-01
2.44950458e-01 2.11839448e-01 2.14727779e-01 2.25145869e-01
5.83307752e-01 5.76206998e-01 5.71460451e-01 5.67935828e-01
2.31607160e-01 2.40981926e-01 2.51973996e-01 2.56875976e-01
2.68747821e-01 2.74806930e-01 2.81686471e-01 2.88174800e-01
2.18438614e-01 2.95580865e-01 2.64796984e-01 3.14748960e-01
5.29477447e-01 5.21057439e-01 5.46380357e-01 5.59021432e-01
5.11678046e-01 4.78335518e-01 5.05164903e-01 4.74546699e-01
4.90601527e-01 4.60133520e-01 4.51927374e-01 4.42716114e-01
4.32666468e-01 4.27660870e-01 4.17027836e-01 4.12283930e-01
3.35633901e-01 3.37984452e-01 5.48526165e-01 2.29187844e-01
5.95520147e-01 5.39532991e-01 3.92794697e-01 3.03784601e-01
3.06561530e-01 5.00667143e-01 4.66474808e-01 5.16453222e-01
3.49148523e-01 4.03962796e-01 3.87945581e-01 3.83035011e-01
3.74174767e-01 3.68304547e-01 1.95539044e-01 3.10006295e-01
3.28040134e-01 3.55685119e-01 3.52930213e-01 3.61707269e-01
3.62197814e-01]
```

In [710…

```
tot = sum(eig_vals)
var_exp = [( i /tot ) * 100 for i in sorted(eig_vals, reverse=True)]
cum_var_exp = np.cumsum(var_exp)
print("Cumulative Variance Explained", cum_var_exp)
```

```
Cumulative Variance Explained [  3.75215051   6.61282292   8.97593056  11.21735261  13.08642996
  14.78951583  16.42751313  17.98101807  19.4967021   20.9185519
  22.28862179  23.60870149  24.84500728  26.07085445  27.23444451
  28.33950605  29.42859004  30.48139407  31.4991528   32.5050346
  33.49679229  34.45881073  35.4076614   36.33070711  37.2476132
  38.13721817  39.0195027   39.88552614  40.74002424  41.57310161
  42.39738638  43.20602975  43.99754244  44.78433179  45.56957913
  46.33263289  47.08352159  47.82798861  48.5606373   49.28883917
  50.00704933  50.72080191  51.42013776  52.10933999  52.78414533
  53.45433891  54.11576044  54.77119981  55.41361594  56.05331687
  56.68345609  57.31040074  57.91819065  58.51595068  59.10251705
  59.67826253  60.24269215  60.8036513   61.35706985  61.90711401
  62.44231139  62.97674791  63.49969343  64.01860242  64.53669384
  65.04966126  65.5549833   66.05374216  66.55241938  67.045981
  67.52926448  68.00953773  68.48902038  68.96285071  69.42791927
  69.89083314  70.34841585  70.79537832  71.23949863  71.68026122
  72.11721218  72.54653233  72.96766998  73.38422758  73.79728963
  74.20780342  74.6153468   75.0183326   75.4199707   75.81668355
  76.2109782   76.60047677  76.98728529  77.36759761  77.74729661
  78.12440997  78.49618958  78.86139209  79.21787924  79.57312347
  79.92538867  80.27363823  80.61631017  80.95338024  81.28538498
  81.61615064  81.94200288  82.26597336  82.58680984  82.90705606
  83.22452901  83.53858809  83.84770548  84.15538451  84.461253
  84.75964178  85.05558212  85.34661407  85.63433473  85.92091491
  86.20445207  86.486686    86.76103094  87.03260153  87.3007943
  87.5649882   87.8268021   88.08425264  88.33967758  88.58986449
  88.83700581  89.08211129  89.32570502  89.56547528  89.800744
  90.03509236  90.26650381  90.49360233  90.71708942  90.93860171
  91.15806588  91.37473649  91.58947797  91.79990219  92.00506539
  92.20860353  92.40867955  92.60603572  92.79987219  92.98975785
  93.17533311  93.35876142  93.5376291   93.71446207  93.88772602
  94.05619986  94.22259385  94.38688165  94.5473692   94.70533894
  94.86068945  95.01582955  95.16838669  95.31976222  95.46951574
  95.61448088  95.75843784  95.89913775  96.03413693  96.16710193
  96.29858944  96.4288859   96.55566368  96.67926492  96.80008325
  96.91795088  97.03321968  97.14679393  97.25697078  97.36504511
  97.47010697  97.57346669  97.67280547  97.77110658  97.86767404
  97.96136469  98.05346372  98.14432392  98.23103545  98.31684124
  98.40071002  98.48335593  98.56289984  98.64082822  98.71651298
  98.78752393  98.85826762  98.9261675   98.9927427   99.0586093
  99.12249153  99.18544225  99.24566241  99.30473624  99.3590952
  99.41262362  99.46446858  99.51145703  99.55380854  99.59574572
  99.6354901   99.67438016  99.71085874  99.74339215  99.77475151
  99.80435919  99.83184183  99.8588607   99.88370487  99.90518507
```

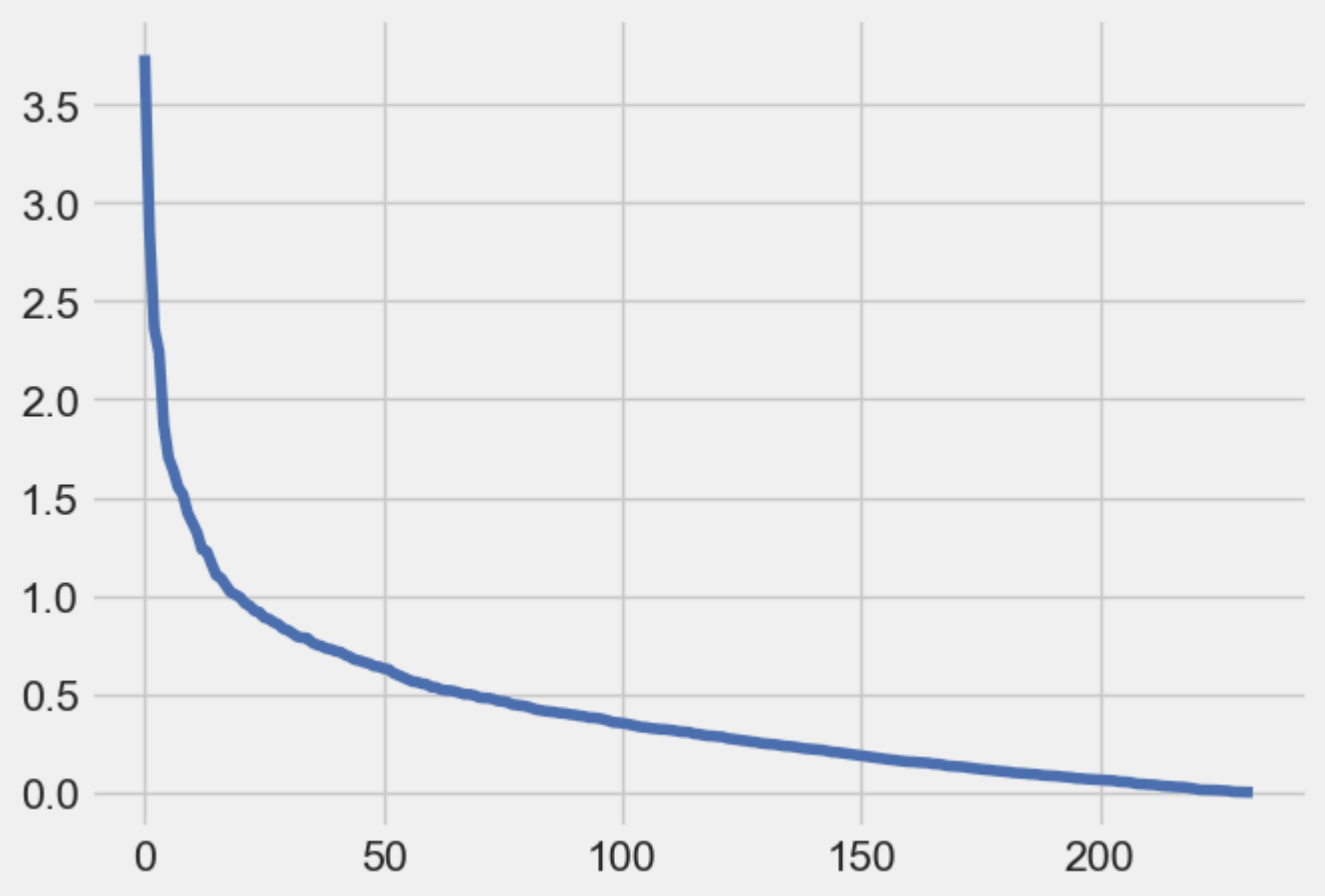
```
  99.92188769  99.93450377  99.94671238  99.95813515  99.96940721
  99.97907063  99.98662885  99.99359126  99.99621823  99.99824324
  99.99942764  99.99991355 100.        ]
```

In [711…

```
plt.plot(var_exp)
```

Out[711]:

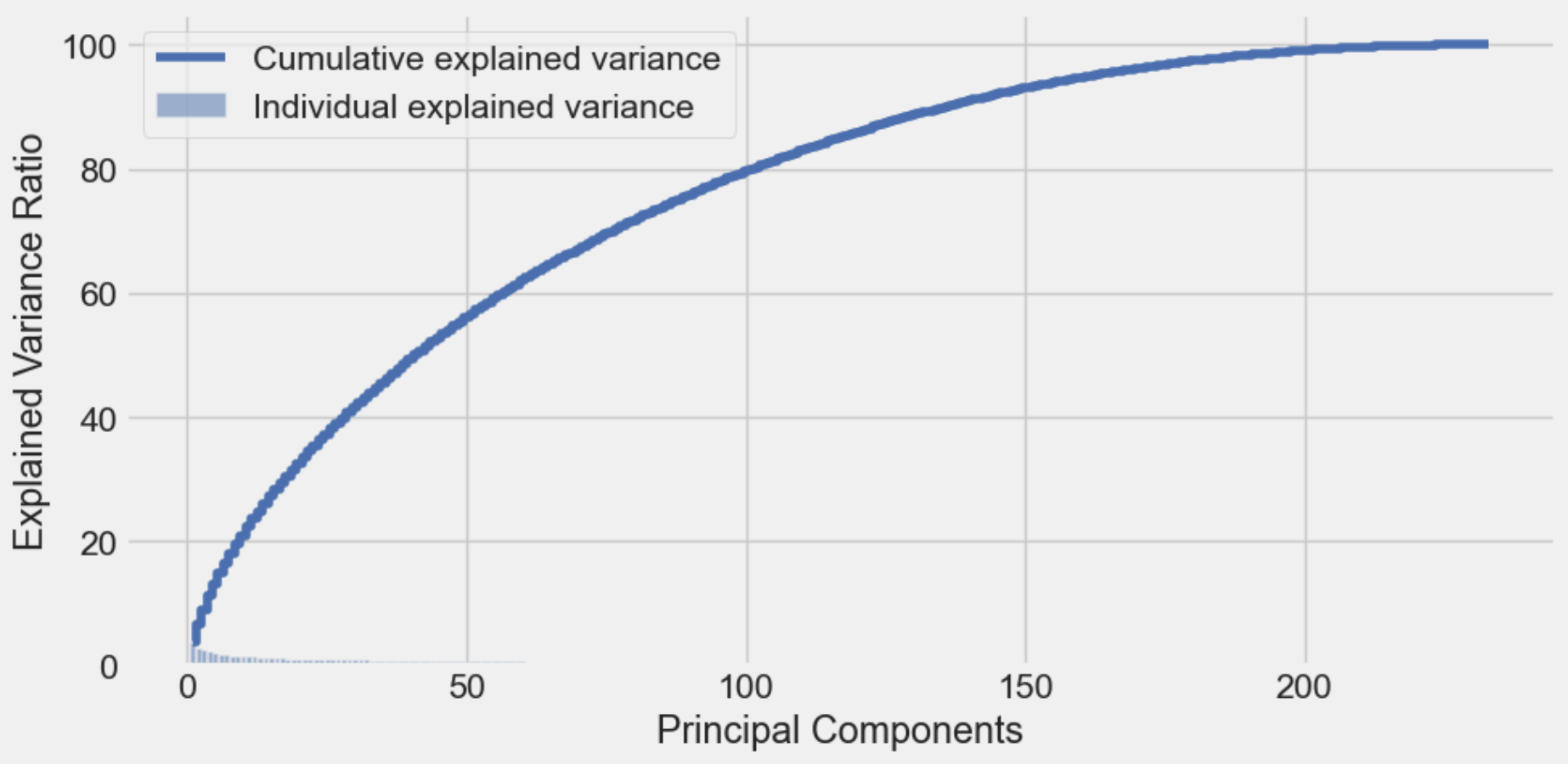
```
[<matplotlib.lines.Line2D at 0x23cfbe6a710>]
```



In [712…

```
# Ploting
plt.figure(figsize=(10 , 5))
plt.bar(range(1, eig_vals.size + 1), var_exp, alpha = 0.5, align = 'center', label = 'Individual explained variance')
plt.step(range(1, eig_vals.size + 1), cum_var_exp, where='mid', label = 'Cumulative explained variance')
plt.ylabel('Explained Variance Ratio')
plt.xlabel('Principal Components')
plt.legend(loc = 'best')
```

```
plt.tight_layout()
plt.show()
```



In [713…
```
len(cum_var_exp)
```

Out[713]:
```
233
```

In [714…
```
# Using scikit learn PCA here. It does all the above steps and maps data to PCA dimensions in one shot
from sklearn.decomposition import PCA

# NOTE - we are generating only 130 PCA dimensions (dimensionality reduction from 306 to 130)
# For 130 components we are getting approximately 90% of the variance
pca = PCA(n_components=130)
data_reduced = pca.fit_transform(X)
data_reduced.transpose()
```

Out[714]:
```
array([[-4.14517481e+00, -2.63846969e+00, -1.48799139e+00, ...,
         2.17746449e+00,  3.30544490e+00,  3.18099300e+00],
       [-1.59393350e+00, -2.18160439e+00,  1.03604846e+00, ...,
        -1.51092817e+00, -8.02083907e-01,  2.63768111e+00],
       [ 7.18446588e-02,  1.15096391e-01, -2.73419856e-02, ...,
         9.78135819e-01, -7.58307641e-01, -1.39180010e+00],
       ...,
       [-8.79829160e-01,  6.87900562e-01,  3.17385916e-02, ...,
         5.72880408e-01, -1.09738011e+00,  2.09033317e-01],
       [ 3.62503685e-02,  1.40877927e+00, -8.71797860e-01, ...,
         2.63088099e-03, -1.29979610e-01, -1.46271790e-01],
       [ 5.46674045e-01, -5.88502111e-01,  6.44388853e-01, ...,
        -9.14603137e-01, -1.03062152e+00,  1.40335860e-02]])
```

In [715…
```
pca.components_
```

Out[715]:
```
array([[ 0.04151562, -0.00441068, -0.01073478, ...,  0.04487553,
        -0.01676348, -0.02538262],
       [-0.01988421, -0.00150389, -0.00380946, ..., -0.02825637,
         0.02500919,  0.04111651],
       [-0.01532609,  0.02144464,  0.005006  , ...,  0.01477176,
        -0.01455088,  0.01217076],
       ...,
       [-0.0058807 , -0.00756068,  0.16195315, ...,  0.06632525,
         0.04640314, -0.02674515],
       [-0.04368576,  0.03587762, -0.04183317, ...,  0.00023617,
         0.05273033,  0.00991881],
       [ 0.13778285,  0.08037527, -0.03098421, ..., -0.04895232,
        -0.01493495, -0.0347743 ]])
```

In [716…
```
df_comp = pd.DataFrame(pca.components_,columns=[X])
df_comp.head()
```

Out[716]:

	0	1	2	3	4	6	7	8	9	10	...	565	570	571	57
0	0.041516	-0.004411	-0.010735	0.072049	0.024202	-0.086338	-0.021677	-0.071148	0.028368	-0.018258	...	-0.001796	0.014131	-0.095104	0.02197
1	-0.019884	-0.001504	-0.003809	-0.020478	0.012892	0.023380	-0.006248	-0.005313	-0.020699	0.001152	...	-0.001926	0.019060	-0.026545	0.00478
2	-0.015326	0.021445	0.005006	0.023912	-0.001445	-0.034854	0.013869	-0.009929	-0.006946	0.027645	...	0.005725	-0.005126	0.026211	0.00584
3	0.014355	-0.030386	-0.056962	0.012280	-0.029122	-0.007962	0.030600	0.011077	0.012031	-0.023789	...	-0.004819	-0.012114	0.042671	0.00033
4	-0.016028	0.009064	-0.005684	0.021747	0.012014	-0.044571	-0.029597	0.047517	0.012508	0.001354	...	0.021526	0.003134	0.044285	0.02490

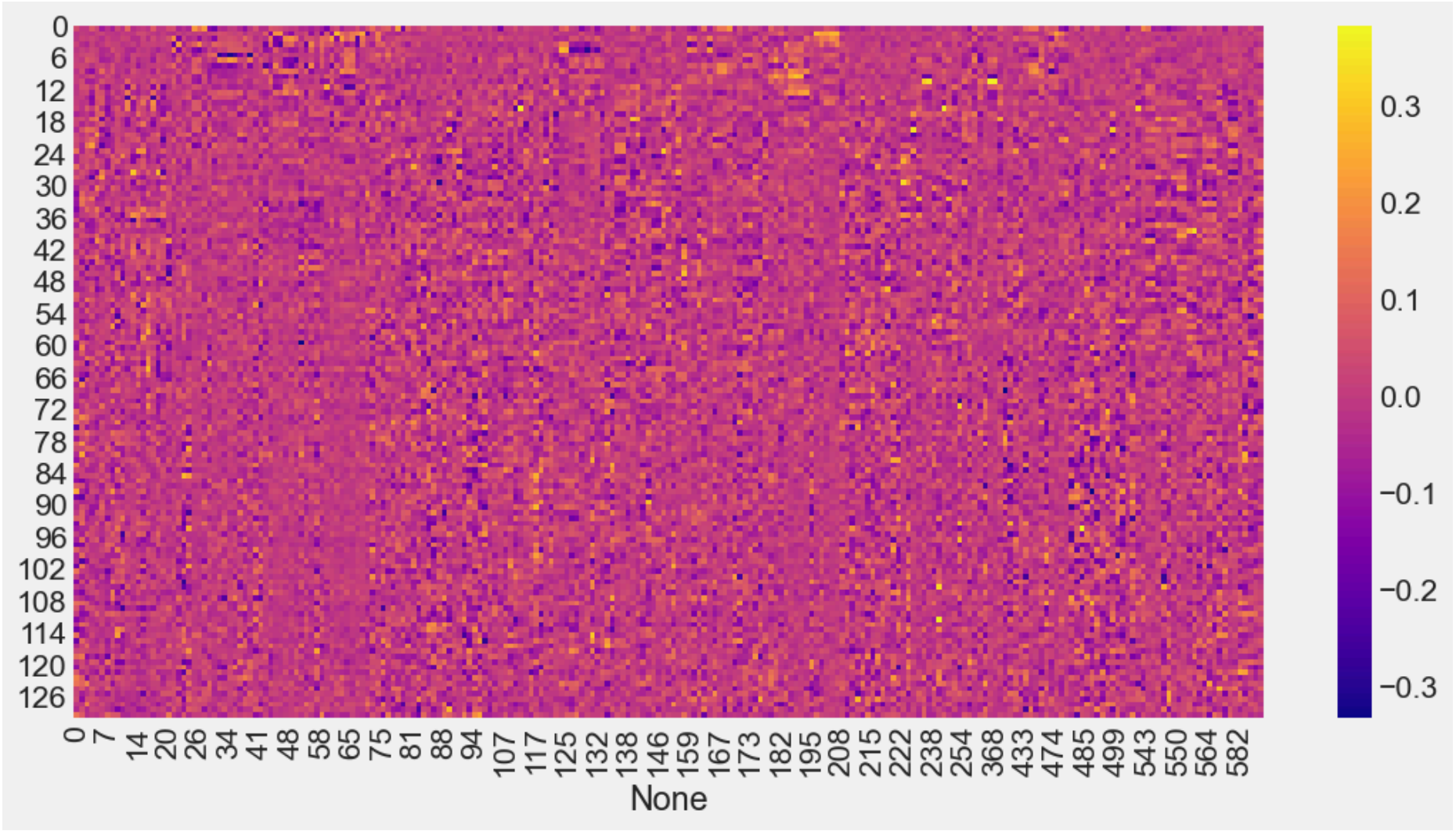
5 rows × 233 columns

In [717…

```
plt.figure(figsize=(12,6))
sns.heatmap(df_comp,cmap='plasma',)
```

Out[717]:

```
<Axes: xlabel='None'>
```



In [718…
```
data_reduced.shape
```

Out[718]:
```
(1567, 130)
```

In [719…
```
df_red2 = pd.DataFrame(data_reduced)
df_red2.head()
```

Out[719]:

	0	1	2	3	4	5	6	7	8	9	...	120	121	122	12
0	-4.145175	-1.593933	0.071845	0.105783	-1.213989	1.160279	1.689056	0.664378	0.755242	-5.016076	...	-0.664840	0.308873	0.122663	0.06825
1	-2.638470	-2.181604	0.115096	-0.254892	-1.681164	0.715334	0.100907	0.984506	1.647267	-2.448497	...	0.444078	0.956545	-1.397056	0.39347
2	-1.487991	1.036048	-0.027342	-2.090404	-2.167080	-0.237045	-0.146342	-0.709647	2.181600	0.095759	...	-1.059031	-0.720311	-1.378906	-0.54728
3	-4.992506	1.911749	0.822977	-2.588636	-2.129381	-1.587287	-1.568638	-3.104166	3.070694	-0.864159	...	0.009308	0.083541	1.608518	-0.03023
4	-2.527951	5.138807	-1.816094	-3.921205	0.837393	-2.057796	-0.183271	1.464817	1.864978	-2.869996	...	0.029598	3.167577	-1.486342	-0.16039

5 rows × 130 columns

In [720…
```
df_red3 = df_red2.copy()
df_red4 = df_red3
df_red4["Pass/Fail"] = data["Pass/Fail"]
```

In [721…
```
df_red4.head()
```

Out[721]:

	0	1	2	3	4	5	6	7	8	9	...	121	122	123	12
0	-4.145175	-1.593933	0.071845	0.105783	-1.213989	1.160279	1.689056	0.664378	0.755242	-5.016076	...	0.308873	0.122663	0.068254	1.29538
1	-2.638470	-2.181604	0.115096	-0.254892	-1.681164	0.715334	0.100907	0.984506	1.647267	-2.448497	...	0.956545	-1.397056	0.393473	0.58840
2	-1.487991	1.036048	-0.027342	-2.090404	-2.167080	-0.237045	-0.146342	-0.709647	2.181600	0.095759	...	-0.720311	-1.378906	-0.547284	0.78270
3	-4.992506	1.911749	0.822977	-2.588636	-2.129381	-1.587287	-1.568638	-3.104166	3.070694	-0.864159	...	0.083541	1.608518	-0.030238	0.06889
4	-2.527951	5.138807	-1.816094	-3.921205	0.837393	-2.057796	-0.183271	1.464817	1.864978	-2.869996	...	3.167577	-1.486342	-0.160395	-0.89347

5 rows × 131 columns

In [722…
```
df_red4.shape
```

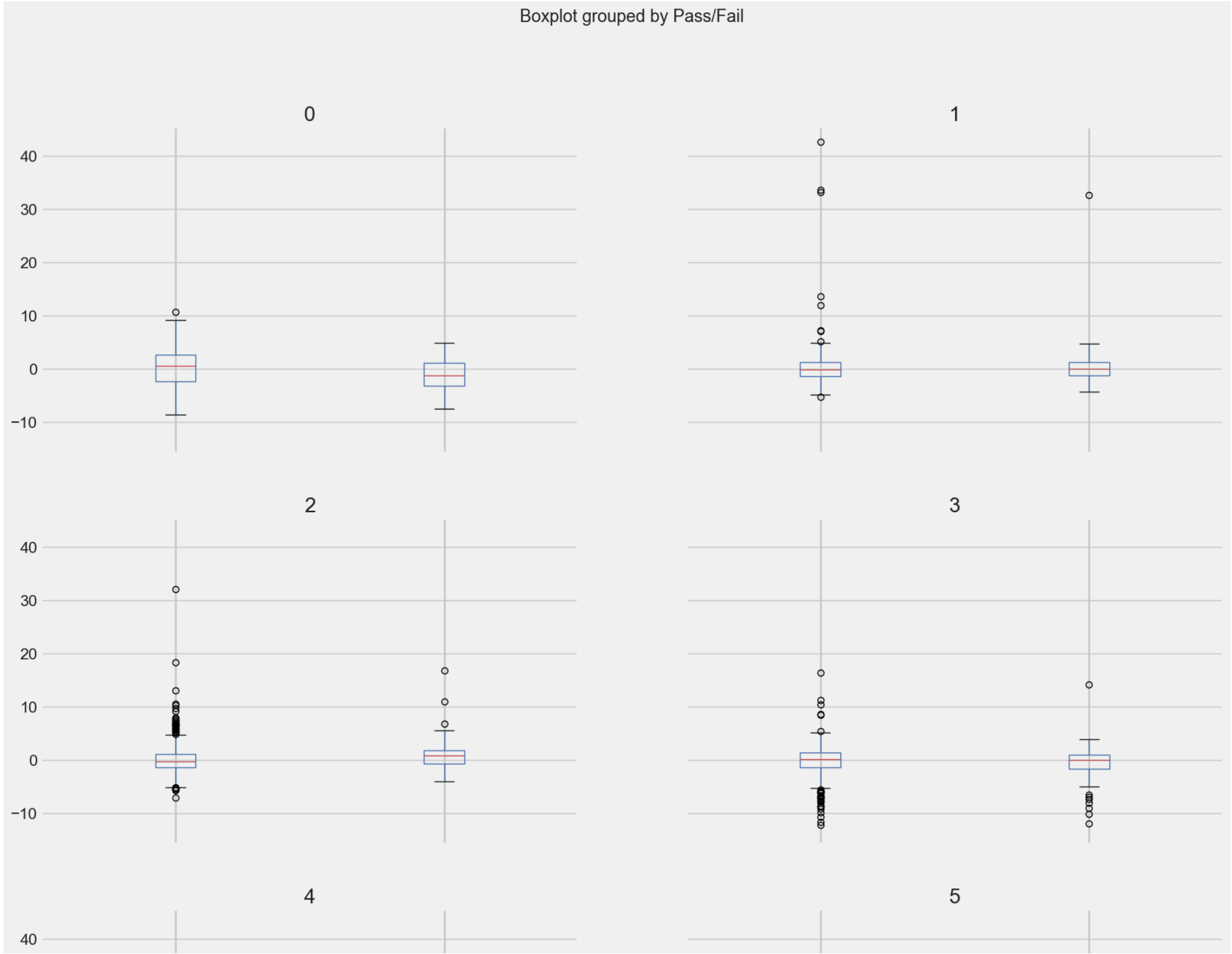
Out[722]:
```
(1567, 131)
```

In [723…
```
#Sample boxplot shows that there are outliers in the data, let us fix them
df_red4.boxplot(column = [df_red4.columns[0],
```

```
                            df_red4.columns[1],
                            df_red4.columns[2],
                            df_red4.columns[3],
                            df_red4.columns[4],
                            df_red4.columns[5],
                           ]
                            , by = 'Pass/Fail', figsize=(20,20))
```

Out[723]:

```
array([[<Axes: title={'center': '0'}, xlabel='Pass/Fail'>,
        <Axes: title={'center': '1'}, xlabel='Pass/Fail'>],
       [<Axes: title={'center': '2'}, xlabel='Pass/Fail'>,
        <Axes: title={'center': '3'}, xlabel='Pass/Fail'>],
       [<Axes: title={'center': '4'}, xlabel='Pass/Fail'>,
        <Axes: title={'center': '5'}, xlabel='Pass/Fail'>]], dtype=object)
```


Boxplot grouped by Pass/Fail
0
1
2
3
4
5
40
30
20
10
0
−10

In [724…

```
#Create a copy of the dataset for maintain data after outlier removal
#Here after identifying outliers we replace with median
pd_data = df_red4.copy()
#pd_data.head()

#pd_data2 = pd_data.drop(columns=['name'],axis=1)
#pd_data2 = pd_data2.apply(replace,axis=1)
from scipy import stats

#Define a function to remove outliers on max side
def outlier_removal_max(var):
    var = np.where(var > var.quantile(0.75)+ stats.iqr(var),var.quantile(0.50),var)
    return var

#Define a function to remove outliers on min side
def outlier_removal_min(var):
    var = np.where(var < var.quantile(0.25) - stats.iqr(var),var.quantile(0.50),var)
    return var

#Loop over the columns and remove the outliers on min and max side
for column in pd_data:
    pd_data[column] = outlier_removal_max(pd_data[column])
    pd_data[column] = outlier_removal_min(pd_data[column])
```

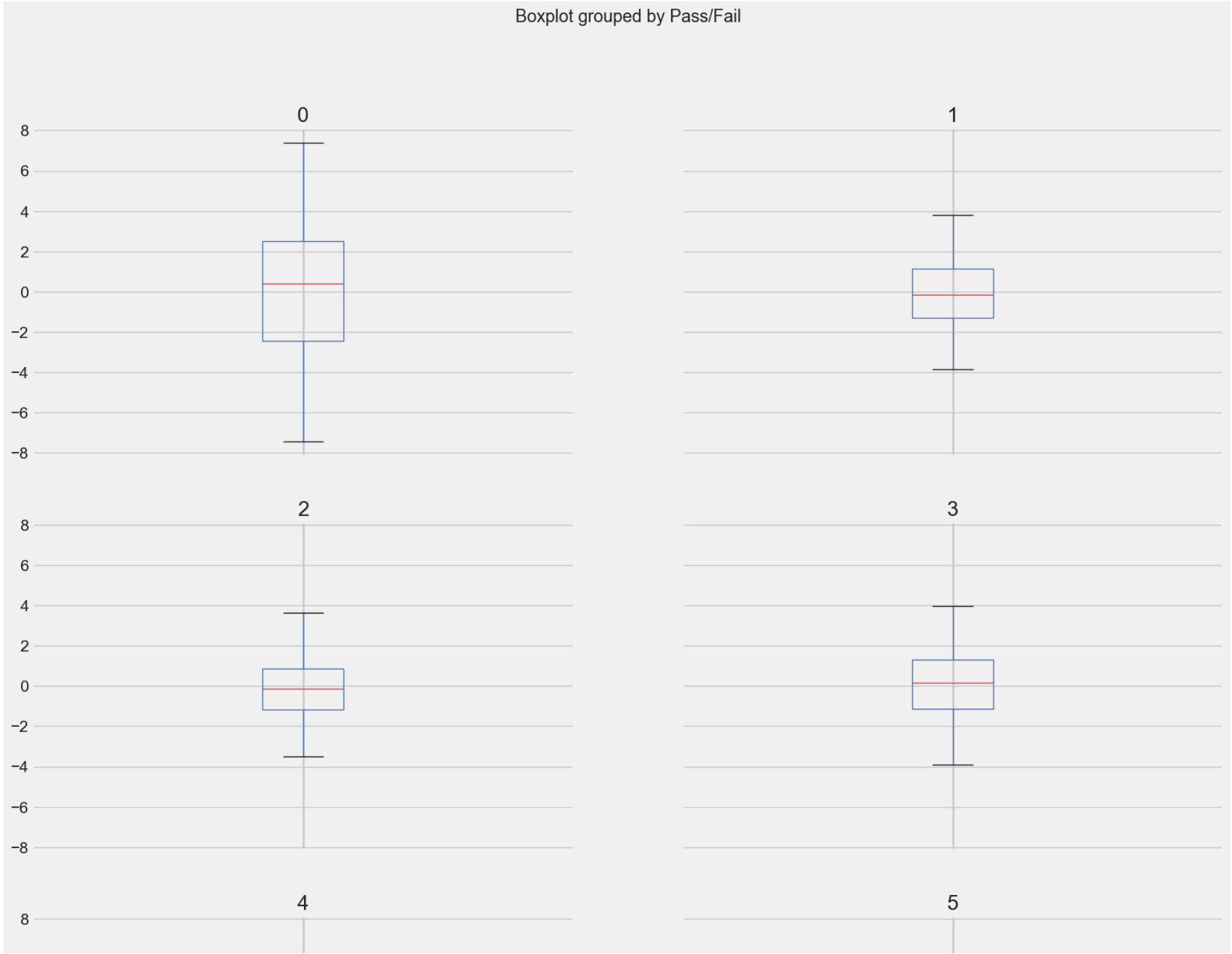
In [725…

```
#Sample boxplot shows that outliers are fixed, but we are loosing observations belonging to failure
#class (Pass/Fail = 1) So we should not remove outliers here
pd_data.boxplot( column =[df_red4.columns[0],
```

```
                            df_red4.columns[1],
                            df_red4.columns[2],
                            df_red4.columns[3],
                            df_red4.columns[4],
                            df_red4.columns[5],
                           ],by = 'Pass/Fail', figsize=(20,20))
```

Out[725]:

```
array([[<Axes: title={'center': '0'}, xlabel='Pass/Fail'>,
        <Axes: title={'center': '1'}, xlabel='Pass/Fail'>],
       [<Axes: title={'center': '2'}, xlabel='Pass/Fail'>,
        <Axes: title={'center': '3'}, xlabel='Pass/Fail'>],
       [<Axes: title={'center': '4'}, xlabel='Pass/Fail'>,
        <Axes: title={'center': '5'}, xlabel='Pass/Fail'>]], dtype=object)
```


Boxplot grouped by Pass/Fail
0
1
2
3
4
5
8
6
4
2
0
−2
−4
−6
−8

In [726…

```
# separating the dependent and independent data

X = df_red4.iloc[:, df_red4.columns != 'Pass/Fail']
Y = df_red4.iloc[:, df_red4.columns == 'Pass/Fail']

# getting the shapes of new data sets x and y
print("shape of X:", X.shape)
print("shape of Y:", Y.shape)
```

```
shape of X: (1567, 130)
shape of Y: (1567, 1)
```

In [ ]:

5 E. Display and explain the classification report in detail. [3 Marks]

lets apply and split dat into train and test fit our model on PCA applied data

Logistic regression-PCA-data

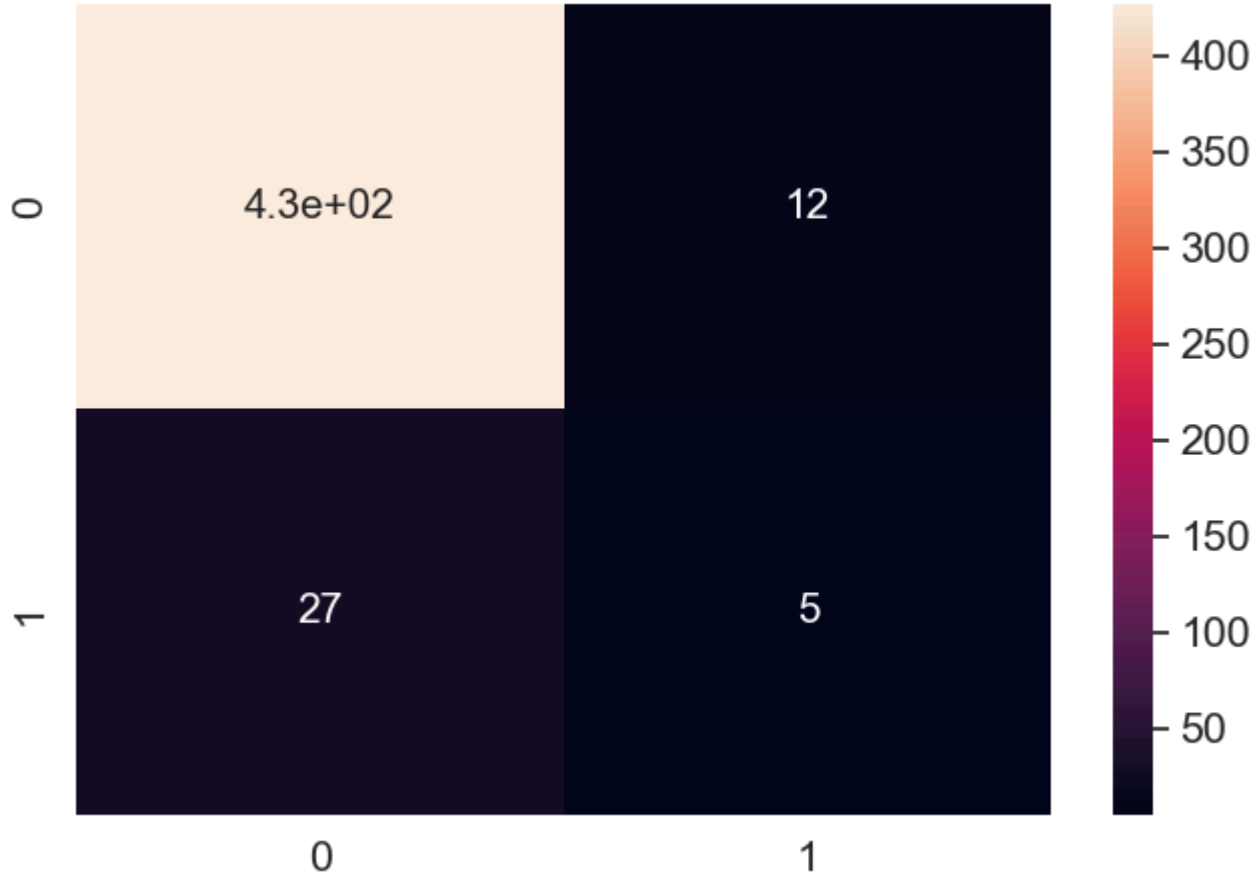
In [727…

```
X_train_pca, X_test_pca, y_train_pca, y_test_pca = train_test_split(X, Y, test_size = 0.3, random_state = 1)
```

In [728…
```
model = LogisticRegression()
model.fit(X_train_pca, y_train_pca)
y_pred_Pca_lr = model.predict(X_test_pca)
```

In [729…
```
cm = confusion_matrix(y_test_pca, y_pred_Pca_lr)
plt.rcParams['figure.figsize'] = (7, 5)
sns.set(style = 'dark', font_scale = 1.4)
sns.heatmap(cm, annot = True, annot_kws = {"size": 15})
```

Out[729]: <Axes: >



In [730…
```
y_test_pca.shape
```

Out[730]: (471, 1)

In [731…
```
y_pred_Pca_lr.shape
```

Out[731]: (471,)

In [732…
```
from sklearn.metrics import classification_report, confusion_matrix

#grid_predictions = grid.predict(X_train_pca)
# print classification report
print(classification_report(y_test_pca,y_pred_Pca_lr))
```

```
              precision    recall  f1-score   support

          -1       0.94      0.97      0.96       439
           1       0.29      0.16      0.20        32

    accuracy                           0.92       471
   macro avg       0.62      0.56      0.58       471
weighted avg       0.90      0.92      0.91       471
```

In [733…
```
print("Accuracy: ", model.score(X_train_pca,y_train_pca)*100)
print("Accuracy: ", model.score(X_test_pca,y_test_pca)*100)
```

```
Accuracy:  95.07299270072993
Accuracy:  91.71974522292994
```

RandomForest with _PCA

In [734…
```
model_Ran_Pca = RandomForestClassifier(n_estimators=100, random_state=1,verbose=0)
model_Ran_Pca.fit(X_train_pca, y_train_pca)
```

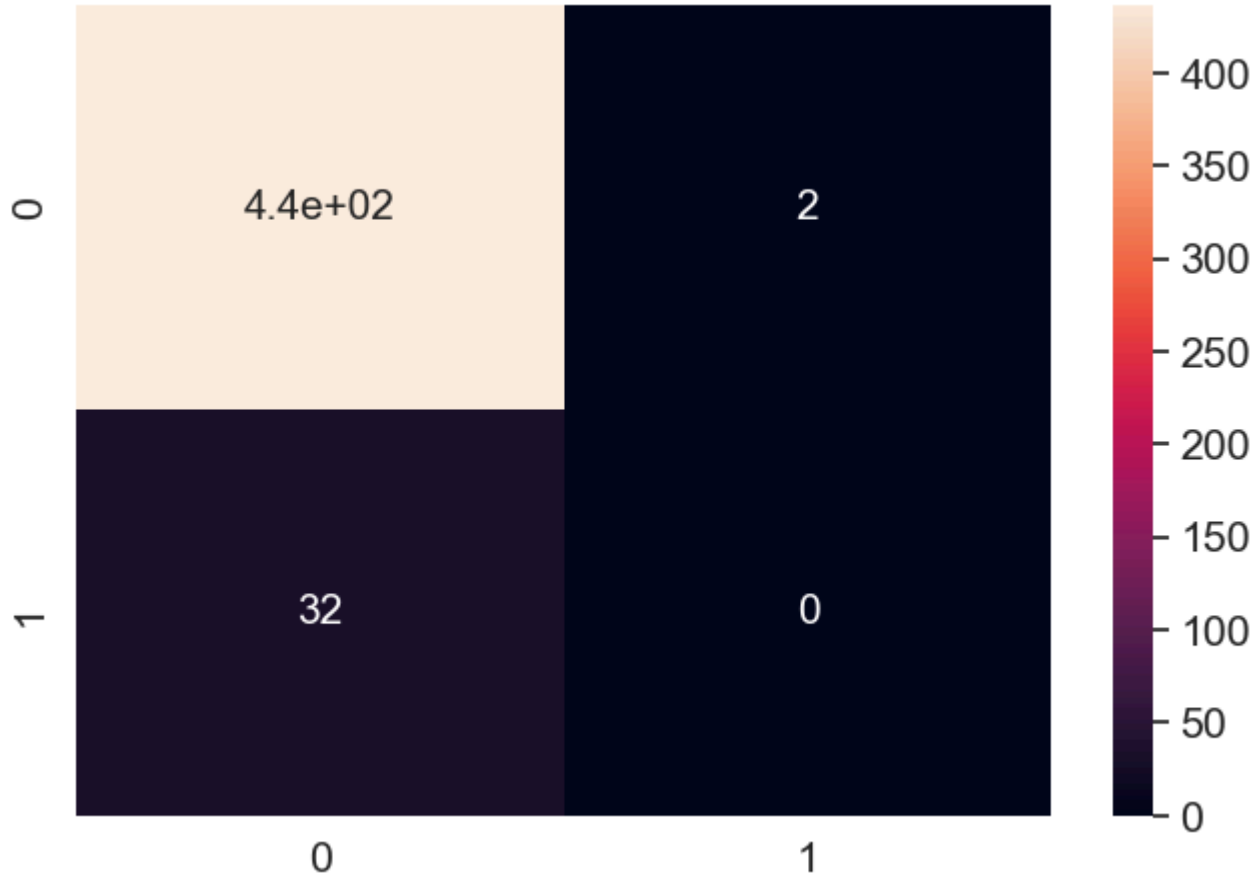
Out[734]:
```
RandomForestClassifier
RandomForestClassifier(random_state=1)
```

In [735…
```
y_pred_Pca_Ran = model_Ran_Pca.predict(X_test_pca)
```

In [736…

```
cm = confusion_matrix(y_test_pca, y_pred_Pca_Ran)
plt.rcParams['figure.figsize'] = (7, 5)
sns.set(style = 'dark', font_scale = 1.4)
sns.heatmap(cm, annot = True, annot_kws = {"size": 15})
```

Out[736]: <Axes: >



In [737…

```
print(classification_report(y_test_pca,y_pred_Pca_Ran))
```

```
              precision    recall  f1-score   support

          -1       0.93      1.00      0.96       439
           1       0.00      0.00      0.00        32

    accuracy                           0.93       471
   macro avg       0.47      0.50      0.48       471
weighted avg       0.87      0.93      0.90       471
```

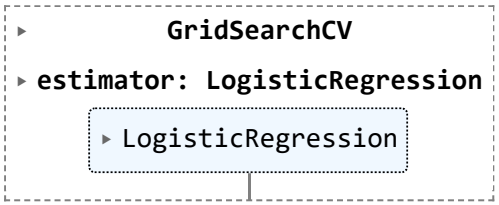
In [738…
```
print("Accuracy: ", model_Ran_Pca.score(X_train_pca,y_train_pca)*100)
print("Accuracy: ", model_Ran_Pca.score(X_test_pca,y_test_pca)*100)
```

```
Accuracy:  100.0
Accuracy:  92.78131634819533
```

logistic regression - without PCA -grid Search

In [739…
```
Log_clf=LogisticRegression()
list=np.arange(1,9)
param_grid={"C":np.logspace(-3,3,7)}
gs_lr=GridSearchCV(Log_clf,param_grid,cv=10)
gs_lr.fit(X_train_os,y_train_res)
```

Out[739]:



In [740…
```
y_pred_lr_gs = gs_lr.predict(X_test_os)
#print("Accuracy of logistic regression oversample greed search:",(y_test_res,y_pred_lr_gs))
print(classification_report(y_test,y_pred_lr_gs))
```

```
              precision    recall  f1-score   support

          -1       0.94      0.90      0.92       439
           1       0.14      0.22      0.17        32

    accuracy                           0.85       471
   macro avg       0.54      0.56      0.54       471
weighted avg       0.89      0.85      0.87       471
```

In [741…

```
print("Accuracy: ", gs_lr.score(X_train_os,y_train_res)*100)
print("Accuracy: ", gs_lr.score(X_test_os,y_test)*100)
```

```
Accuracy:  100.0
Accuracy:  85.35031847133759
```

- summary:
- classification report for PCA perform: data after dimensionality reduction logistic regression model was fitted recall was 97% out of all passes signal how much pass predicted was correct , Accuracy on train is 95% and accuracy on test in 92% the total prediction made passes or fails againts all actual signal. show there is some diference in model performance on accuracy on train and test data.
- calssification report with oversampledat with grid search CV recall was 90% out of all passes signal how much pass predicted was correct , Accuracy on train is 100% and accuracy on test in 85% the total prediction made passes or fails againts all actual signal. show there is more diference in model performance on accuracy on train and test data.on train data model performed well but on test it performed poor

5 F. Apply the above steps for all possible models that you have learnt so far. [5 Marks]

For this question we will perform all steps on undersampled data

In [742…

```
# Under Sampling - Check how many failure observations are there
# We have 104 such observations

failed_tests = np.array(df_red4[df_red4['Pass/Fail'] == 1].index)
no_failed_tests = len(failed_tests)
```

```
print(no_failed_tests)
```

```
104
```

In [743…

```
# Check how many pass observations are there
# We have 1,463 such observations

normal_indices = df_red4[df_red4['Pass/Fail'] == -1]
no_normal_indices = len(normal_indices)

print(no_normal_indices)
```

```
1463
```

In [744…

```
# Get 104 random observations from the pass class as well

random_normal_indices = np.random.choice(no_normal_indices, size = no_failed_tests, replace = True)
random_normal_indices = np.array(random_normal_indices)

print(len(random_normal_indices))
```

```
104
```

In [745…

```
#Getting a 50-50 representation from both pass and fail classes
under_sample = np.concatenate([failed_tests, random_normal_indices])
print(len(under_sample))
```

```
208
```

In [746…

```
# creating the undersample data

undersample_data = df_red4.iloc[under_sample, :]

# splitting the undersample dataset into x and y sets

X = undersample_data.iloc[:, undersample_data.columns != 'Pass/Fail']
Y = undersample_data.iloc[:, undersample_data.columns == 'Pass/Fail']

print(X.shape)
print(Y.shape)
```

```
(208, 130)
(208, 1)
```

In [747…

```
from sklearn.model_selection import train_test_split

X_train_us, X_test_us, y_train_us, y_test_us = train_test_split(X, Y, test_size = 0.3, random_state = 1)

print(X_train_us.shape)
print(y_train_us.shape)
print(X_test_us.shape)
print(y_test_us.shape)
```

```
(145, 130)
(145, 1)
(63, 130)
(63, 1)
```

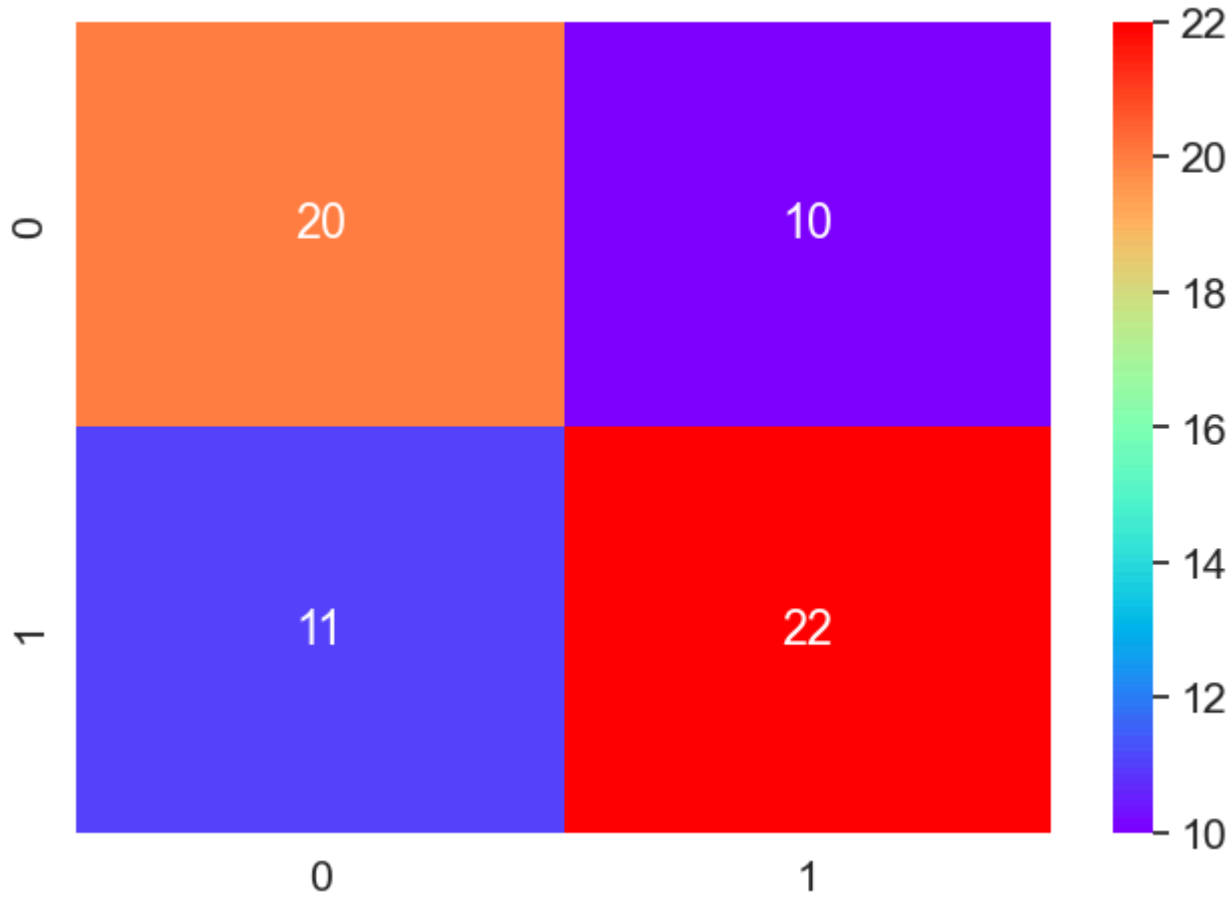
Random Forest - PCA - Undersampled

In [748…

```
model = RandomForestClassifier(n_estimators=100, random_state=1,verbose=0 )
model.fit(X_train_us, y_train_us)
#scores_prediction = model.decision_function(x_train)
y_pred_Pca_Un = model.predict(X_test_us)

# evaluating the model

# printing the confusion matrix
cm = confusion_matrix(y_test_us, y_pred_Pca_Un)
sns.heatmap(cm, annot = True, cmap = 'rainbow')
```

Out[748]:

```
<Axes: >
```



In [749…

```
print("Accuracy: ", model.score(X_train_us,y_train_us)*100)
print("Accuracy: ", model.score(X_test_us,y_test_us)*100)
```

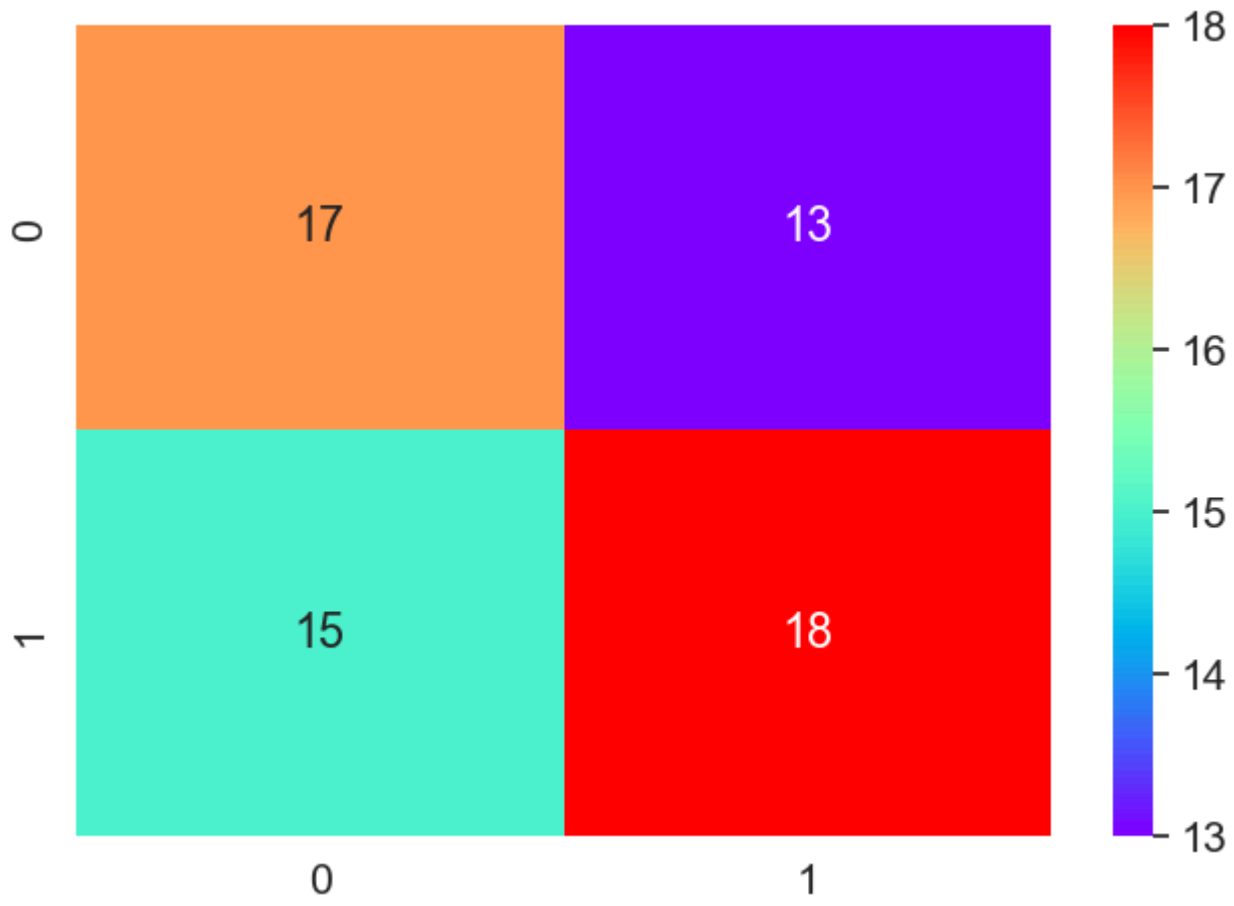
```
Accuracy:  100.0
Accuracy:  66.66666666666666
```

Logistic Regression - PCA - Undersample

In [750…

```
lr = LogisticRegression(random_state=1)
lr.fit(X_train_us, y_train_us)
y_pred_Pca_lr_Un = lr.predict(X_test_us)
cm = confusion_matrix(y_test_us, y_pred_Pca_lr_Un)
sns.heatmap(cm, annot = True, cmap = 'rainbow')
```

Out[750]: <Axes: >



In [751…
```
print("Accuracy: ", lr.score(X_train_us,y_train_us)*100)
print("Accuracy: ", lr.score(X_test_us,y_test_us)*100)
```

```
Accuracy:  100.0
Accuracy:  55.55555555555556
```

Lasso - PCA -Undersampled

In [752…
```
lasso = Lasso(alpha=0.1,random_state=1)
lasso.fit(X_train_us,y_train_us)
```

```
#print ("Lasso model:", (lasso.coef_))

y_pred_Pca_lass_Un = lasso.predict(X_test_us)

#Convert the sign of the predicted values as the classifier
y_pred2 = np.sign(y_pred_Pca_lass_Un)
```

In [753…

```
list=[y_test_us]
actual_cost = list
actual_cost = np.asarray(actual_cost)
y_pred_lass = lasso.predict(X_test_us)
```

In [754…

```
print("Accuracy: ", lasso.score(X_train_us,y_train_us)*100)
print("Accuracy: ", lasso.score(X_test_us, y_test_us)*100)
```

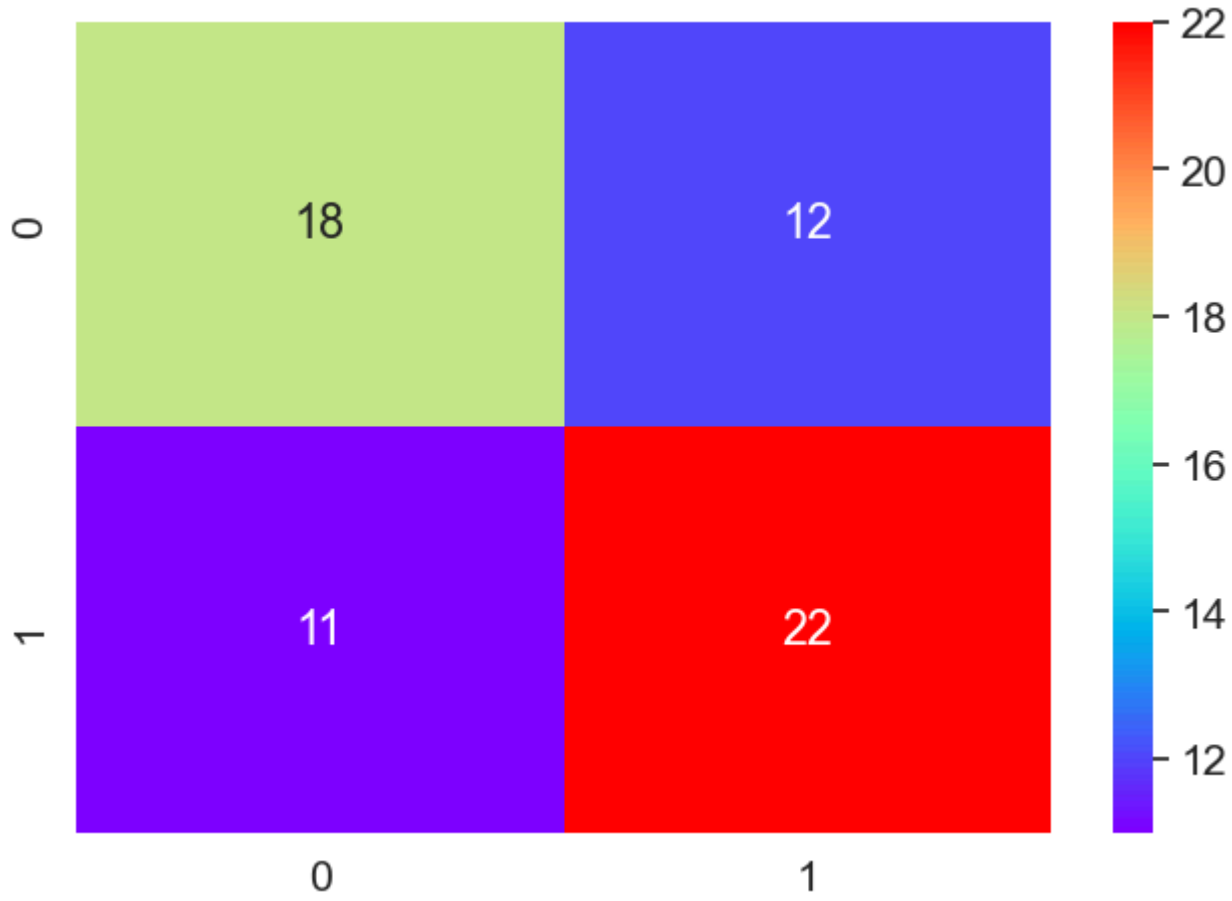
```
Accuracy:  31.547193408331232
Accuracy:  9.481626515503583
```

In [755…

```
cm = confusion_matrix(y_test_us, y_pred2)
sns.heatmap(cm, annot = True, cmap = 'rainbow')
```

Out[755]:

```
<Axes: >
```



In [756…

```
print(classification_report(y_test_us,y_pred2))
```

```
              precision    recall  f1-score   support

          -1       0.62      0.60      0.61        30
           1       0.65      0.67      0.66        33

    accuracy                           0.63        63
   macro avg       0.63      0.63      0.63        63
weighted avg       0.63      0.63      0.63        63
```

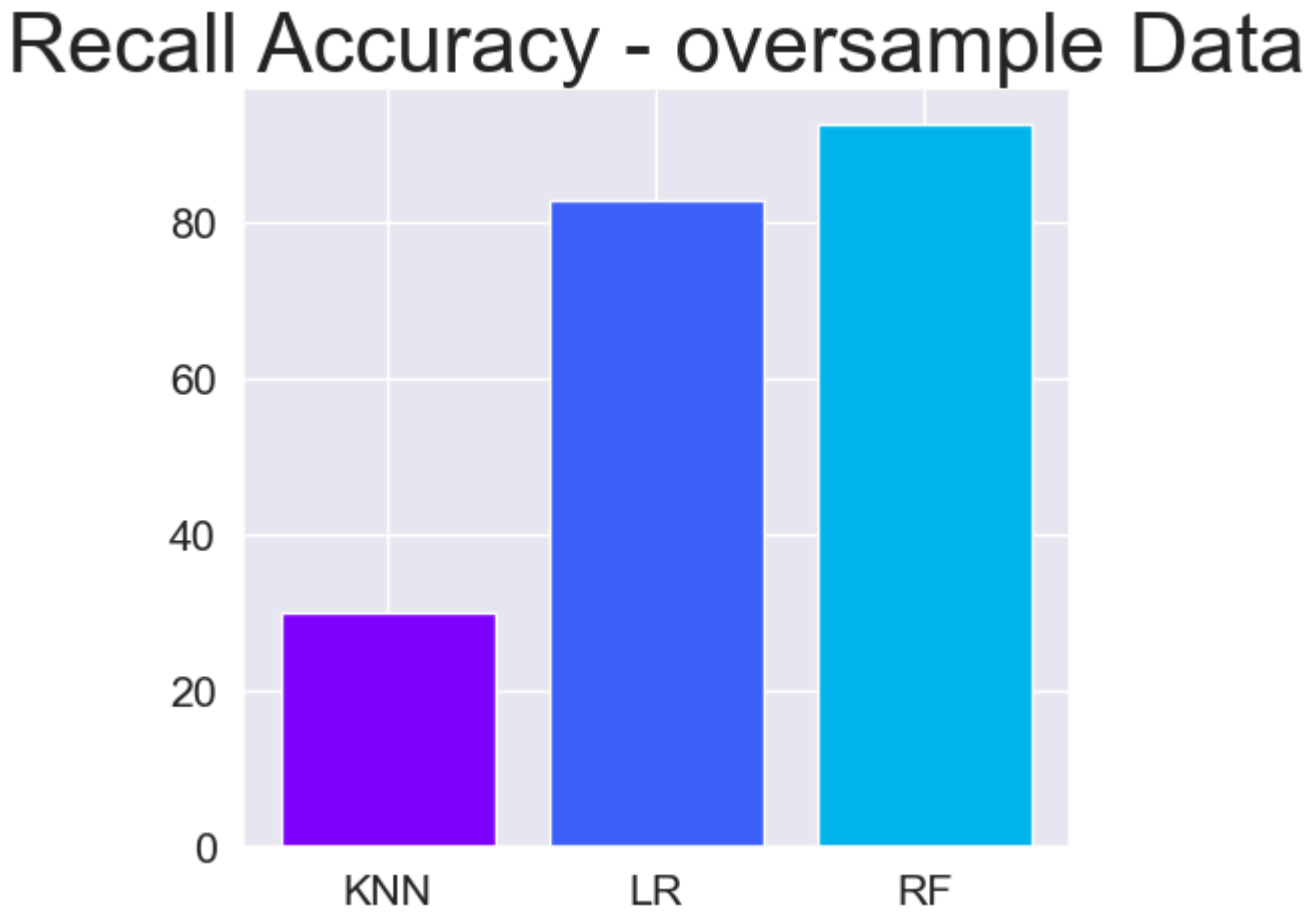
6. Post Training and Conclusion: [5 Marks]

6A. Display and compare all the models designed with their train and test accuracies. [1 Marks]

Accuracy for oversample (without PCA) (without Greed Search)

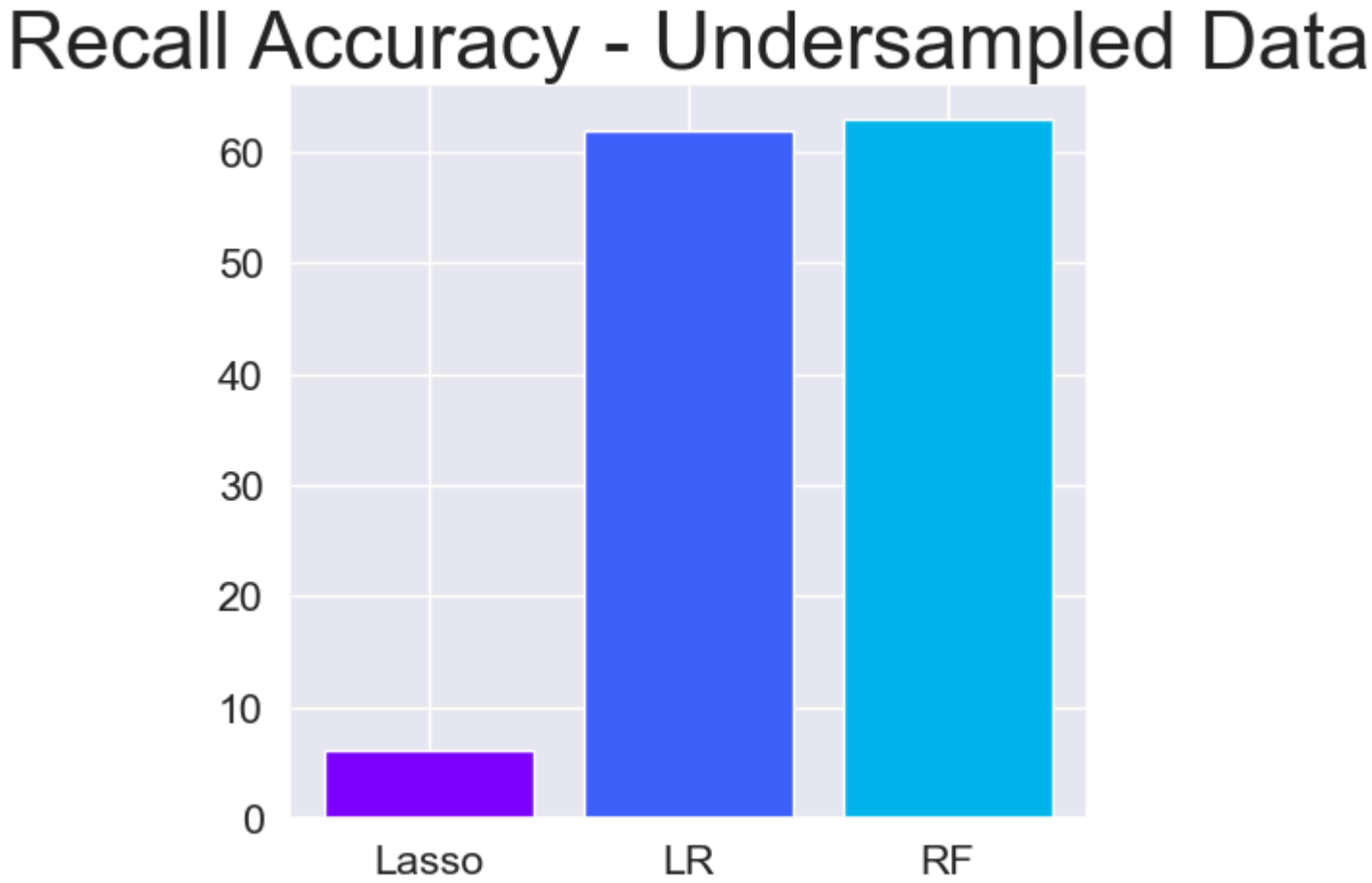
In [757…

```
plt.figure(figsize=(5,5))
Recall = np.array([92.7,83,30])
label = np.array(['RF','LR','KNN'])
indices = np.argsort(Recall)
color = plt.cm.rainbow(np.linspace(0, 1, 9))

plt.rcParams['figure.figsize'] = (18, 7)
plt.bar(range(len(indices)), Recall[indices], color = color)
plt.xticks(range(len(indices)), label[indices])
plt.title('Recall Accuracy - oversample Data', fontsize = 30)
plt.grid()
plt.tight_layout()
plt.show()
```



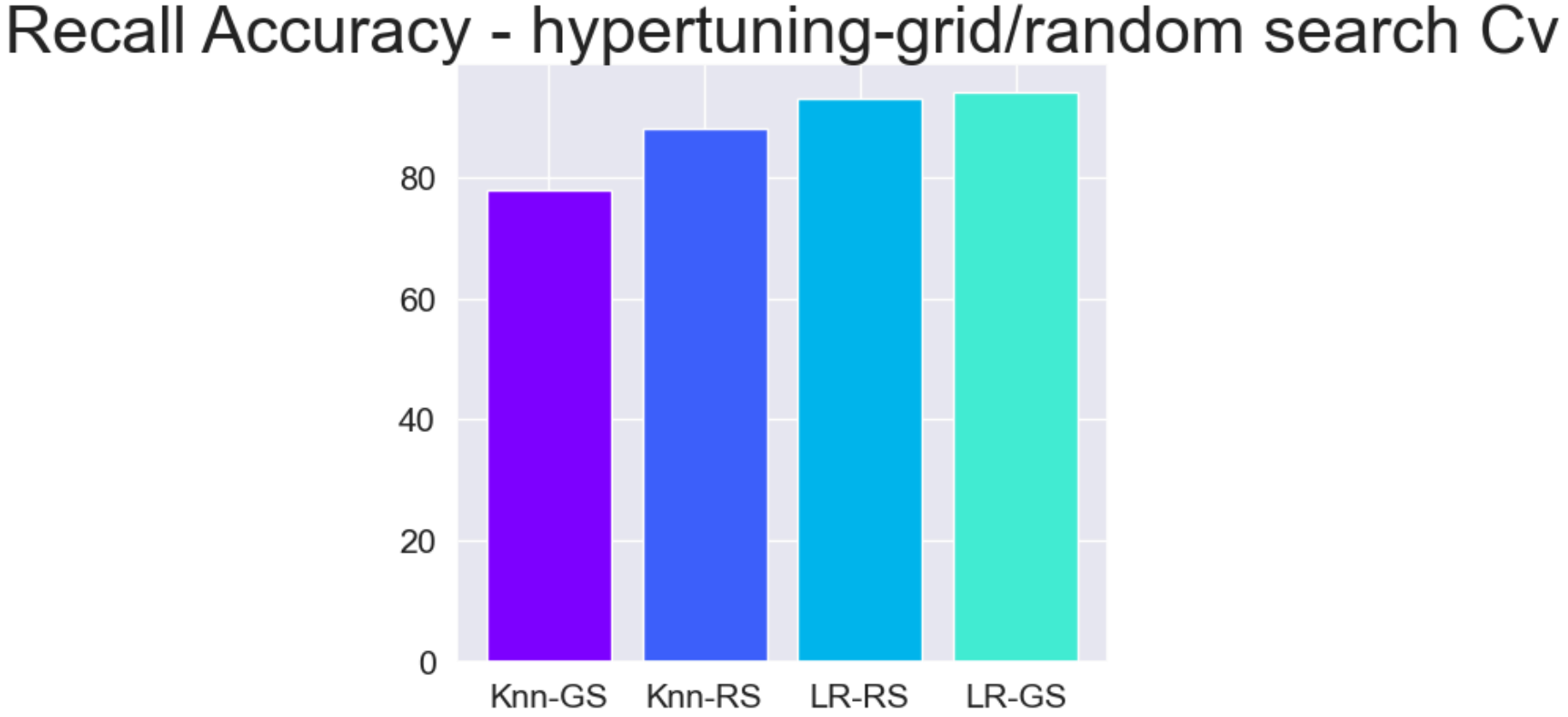
In [758…

```
Recall = np.array([62,6,63])
label = np.array(['LR','Lasso','RF'])
indices = np.argsort(Recall)
color = plt.cm.rainbow(np.linspace(0, 1, 9))

plt.rcParams['figure.figsize'] = (5, 5)
plt.bar(range(len(indices)), Recall[indices], color = color)
plt.xticks(range(len(indices)), label[indices])
plt.title('Recall Accuracy - Undersampled Data', fontsize = 30)
plt.grid()
plt.tight_layout()
plt.show()
```



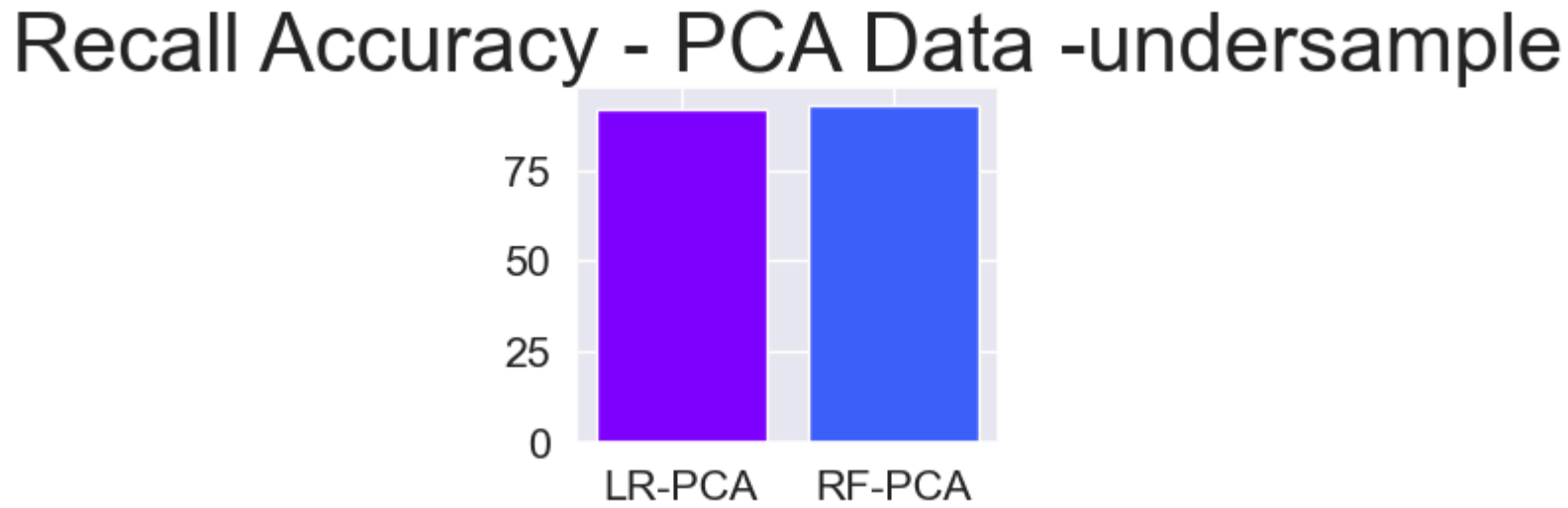
In [759…

```
Recall = np.array([78,94,88,93])
label = np.array(['Knn-GS','LR-GS','Knn-RS','LR-RS'])
indices = np.argsort(Recall)
color = plt.cm.rainbow(np.linspace(0, 1, 9))

plt.rcParams['figure.figsize'] = (5, 5)
plt.bar(range(len(indices)), Recall[indices], color = color)
plt.xticks(range(len(indices)), label[indices])
plt.title('Recall Accuracy - hypertuning-grid/random search Cv', fontsize = 30)
plt.grid()
plt.tight_layout()
plt.show()
```



In [760…

```
Recall = np.array([92,93])
label = np.array(['LR-PCA','RF-PCA'])
indices = np.argsort(Recall)
color = plt.cm.rainbow(np.linspace(0, 1, 9))

plt.rcParams['figure.figsize'] = (3, 3)
plt.bar(range(len(indices)), Recall[indices], color = color)
plt.xticks(range(len(indices)), label[indices])
plt.title('Recall Accuracy - PCA Data -undersample', fontsize = 30)
plt.grid()
plt.tight_layout()
plt.show()
```



6 B. Select the final best trained model along with your detailed comments for selecting this model. [1 Marks]

In [761…
```
print(classification_report(y_test_pca,y_pred_Pca_Ran))
```

```
              precision    recall  f1-score   support

          -1       0.93      1.00      0.96       439
           1       0.00      0.00      0.00        32

    accuracy                           0.93       471
   macro avg       0.47      0.50      0.48       471
weighted avg       0.87      0.93      0.90       471
```

In [762…
```
print(classification_report(y_test_pca,y_pred_Pca_lr))
```

```
              precision    recall  f1-score   support

          -1       0.94      0.97      0.96       439
           1       0.29      0.16      0.20        32

    accuracy                           0.92       471
   macro avg       0.62      0.56      0.58       471
weighted avg       0.90      0.92      0.91       471
```

- 1) though accuracy in random foreset is good with PCA (dimentionareduction performed on data)
- 2) LogisticRegression with PCA with outlier removed perfomed well on train as well as test data
- recall for model 2 has recall .97 And F1 score is 0.96
- we able to identity predicted all pass signals correctly fro actuall pass signals
- F1 score is high which shows balance between recall as well as precision and acuuracies
- logistic regrattion also performed well with hyperparameter tunning with random seach method
- also logistic regression with oversampled data has high recall/accuracy score
- we can choose logisticRegression as best model

6C. Pickle the selected model for future use. [2 Marks]

In [763…

```
data_pickel={
    'classification':['RF','LR','knn','LR','RF','LR','Knn','LR','Knn','LR','LR','Lasso','RF'],
    'Technique':['Oversampling','Oversampling','oversampling','cross-validation-kfold','cross-validtion-kfold',
                 'Grid search','Grid Search','randomsearch','randomsearch','PCA','undersample-PCA','undersample-PCA','undersample
    'score':[93,88,30,61,62,78,95,88,93,92,62,6,63]}
pd.DataFrame(data=data_pickel)
```

Out[763]:

	classification	Technique	score
0	RF	Oversampling	93
1	LR	Oversampling	88
2	knn	oversampling	30
3	LR	cross-validation-kfold	61
4	RF	cross-validtion-kfold	62
5	LR	Grid search	78
6	Knn	Grid Search	95
7	LR	randomsearch	88
8	Knn	randomsearch	93
9	LR	PCA	92
10	LR	undersample-PCA	62
11	Lasso	undersample-PCA	6
12	RF	undersample-PCA	63

- here with pickeling data for future use

6D. Write your conclusion on the results. [1 Marks]

- We have tried multiple models Logistic Regression, Random Forest,(with and without Grid Search) for the imbalanced target
- Across methods Lasso performed the worst while Logistic regression/random foreest performed the best in terms of recall accuracy in PCA
- We saw that for imbalanced classes accuracy and recall are invertially proportional to each other. Better recall models have lower accuracy and vice versa.
- We have tried two sampling techniques -first one using SMOTE (oversampling) and second one
- using random based method (undersampling), Oversampling gave better results than undersampling in
- terms of accuracy. Recall score was similar for both undersampling and oversampling.

- We did scaling on both the datasets and took PCA with n_components as 130 (90% variance coverage). However PCA did not improve either accuracy or recall probably as we were loosing information due to dropping dimensions.
- We tried K-fold cross validation within itself with value 15
- The best recall value at 88% algorithm, the best part was no sampling was required as the algorithm took care of sampling as well as outliers.

==========**END OF PROJECT**==========

In [ ]: