PA Assignment Final

April 30, 2022

Importing libraries

[20]:
```python
# Prerequisites

#Calculations
import numpy as np

#Data processing
import pandas as pd

#Visualization
import seaborn as sns
import matplotlib.pyplot as plt
!pip install plotly
import plotly.express as px
#Collection of functions for scientific computing and advance mathematics
import scipy as sp

#Statistical Models
import statsmodels.api as sm
```

```
Requirement already satisfied: plotly in /opt/anaconda3/lib/python3.8/site-
packages (5.7.0)
Requirement already satisfied: tenacity>=6.2.0 in
/opt/anaconda3/lib/python3.8/site-packages (from plotly) (8.0.1)
Requirement already satisfied: six in /opt/anaconda3/lib/python3.8/site-packages
(from plotly) (1.15.0)
```

Loading the data file

[21]:
```python
data = pd.read_csv('Assignment_PA.csv')
data.head()
```

[21]:
```
     V1    V2       V3       V4    V5  V6   V7      V8  V9  V10  …     V25  \
0    42    50   270900   270944   267  17   44   24220  76  108  …  0.8182
1   645   651  2538079  2538108   108  10   30   11397  84  123  …  0.7931
2   829   835  1553913  1553931    71   8   19    7972  99  125  …  0.6667
3   853   860   369370   369415   176  13   45   18996  99  126  …  0.8444
4  1289  1306   498078   498335  2409  60  260  246930  37  126  …  0.9338
```

```
        V26     V27  V28  V29  V30  V31  V32  V33  Class
0 -0.2913  0.5822    1    0    0    0    0    0      1
1 -0.1756  0.2984    1    0    0    0    0    0      1
2 -0.1228  0.2150    1    0    0    0    0    0      1
3 -0.1568  0.5212    1    0    0    0    0    0      1
4 -0.1992  1.0000    1    0    0    0    0    0      1

[5 rows x 34 columns]
```

The info() method prints information about the DataFrame. The information contains the number of columns, column labels, column data types, memory usage, range index, and the number of cells in each column

[22]:
```python
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1941 entries, 0 to 1940
Data columns (total 34 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   V1      1941 non-null   int64
 1   V2      1941 non-null   int64
 2   V3      1941 non-null   int64
 3   V4      1941 non-null   int64
 4   V5      1941 non-null   int64
 5   V6      1941 non-null   int64
 6   V7      1941 non-null   int64
 7   V8      1941 non-null   int64
 8   V9      1941 non-null   int64
 9   V10     1941 non-null   int64
 10  V11     1941 non-null   int64
 11  V12     1941 non-null   int64
 12  V13     1941 non-null   int64
 13  V14     1941 non-null   int64
 14  V15     1941 non-null   float64
 15  V16     1941 non-null   float64
 16  V17     1941 non-null   float64
 17  V18     1941 non-null   float64
 18  V19     1941 non-null   float64
 19  V20     1941 non-null   float64
 20  V21     1941 non-null   float64
 21  V22     1941 non-null   float64
 22  V23     1941 non-null   float64
 23  V24     1941 non-null   float64
 24  V25     1941 non-null   float64
 25  V26     1941 non-null   float64
 26  V27     1941 non-null   float64
```

```
 27  V28     1941 non-null   int64
 28  V29     1941 non-null   int64
 29  V30     1941 non-null   int64
 30  V31     1941 non-null   int64
 31  V32     1941 non-null   int64
 32  V33     1941 non-null   int64
 33  Class   1941 non-null   int64
dtypes: float64(13), int64(21)
memory usage: 515.7 KB
```

The nunique() method returns the number of unique values for each column

[23]:
```python
data.nunique()
```

[23]:
```
V1        962
V2        994
V3       1939
V4       1940
V5        920
V6        399
V7        317
V8       1909
V9        161
V10       100
V11        84
V12         2
V13         2
V14        24
V15      1387
V16      1338
V17       770
V18       454
V19       818
V20       648
V21         3
V22       914
V23       183
V24       217
V25       918
V26      1522
V27       388
V28         2
V29         2
V30         2
V31         2
V32         2
V33         2
Class       2
```

```
dtype: int64
```

Checking Values of Target Variable

[24]:
```
data.Class.value_counts()
```

[24]:
```
1    1268
2     673
Name: Class, dtype: int64
```

[25]:
```
#HotCoding to change Class from 1,2 to 1,0
data["Class"] = data["Class"].replace(2, 0)
```

All the unique values for each column will store in data_objects. Dataframe is created by loading data_objects of unique values. We are selecting rows from V3 as we are using iloc function as data_objects.iloc[0:]. Providing columns names to the two columns as objects and unique_count.

[26]:
```
data_object = []
data_objects = data.nunique()
data_objects = pd.DataFrame(data_objects)
data_objects = data_objects.iloc[0:]
data_objects.reset_index(inplace=True)
data_objects.columns = ["Objects","Unique_count"]
print(data_objects)
```

```
   Objects  Unique_count
0       V1           962
1       V2           994
2       V3          1939
3       V4          1940
4       V5           920
5       V6           399
6       V7           317
7       V8          1909
8       V9           161
9      V10           100
10     V11            84
11     V12             2
12     V13             2
13     V14            24
14     V15          1387
15     V16          1338
16     V17           770
17     V18           454
18     V19           818
19     V20           648
20     V21             3
21     V22           914
22     V23           183
```

```
23      V24            217
24      V25            918
25      V26           1522
26      V27            388
27      V28              2
28      V29              2
29      V30              2
30      V31              2
31      V32              2
32      V33              2
33    Class              2
```

The loc() function helps us to retrieve data values from a data objects which is having unique values that are equal to 2. Making a list of all the data objects whose unique value equal to 2

[27]:
```
data_objects_select = data_objects.loc[data_objects['Unique_count'] == 2]
data_objects_select = list(data_objects_select.Objects)
data_objects_select
```

[27]:
```
['V12', 'V13', 'V28', 'V29', 'V30', 'V31', 'V32', 'V33', 'Class']
```

Converting the fields in "data object select" to object datatype and confirming that the object type changed on the selected fields.

[28]:
```
for i in data_objects_select :
  data[i] = data[i].astype("object")

data.info()
data_original = data.describe()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1941 entries, 0 to 1940
Data columns (total 34 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   V1      1941 non-null   int64
 1   V2      1941 non-null   int64
 2   V3      1941 non-null   int64
 3   V4      1941 non-null   int64
 4   V5      1941 non-null   int64
 5   V6      1941 non-null   int64
 6   V7      1941 non-null   int64
 7   V8      1941 non-null   int64
 8   V9      1941 non-null   int64
 9   V10     1941 non-null   int64
 10  V11     1941 non-null   int64
 11  V12     1941 non-null   object
 12  V13     1941 non-null   object
 13  V14     1941 non-null   int64
```

```
 14  V15     1941 non-null   float64
 15  V16     1941 non-null   float64
 16  V17     1941 non-null   float64
 17  V18     1941 non-null   float64
 18  V19     1941 non-null   float64
 19  V20     1941 non-null   float64
 20  V21     1941 non-null   float64
 21  V22     1941 non-null   float64
 22  V23     1941 non-null   float64
 23  V24     1941 non-null   float64
 24  V25     1941 non-null   float64
 25  V26     1941 non-null   float64
 26  V27     1941 non-null   float64
 27  V28     1941 non-null   object
 28  V29     1941 non-null   object
 29  V30     1941 non-null   object
 30  V31     1941 non-null   object
 31  V32     1941 non-null   object
 32  V33     1941 non-null   object
 33  Class   1941 non-null   object
dtypes: float64(13), int64(12), object(9)
memory usage: 515.7+ KB
```

The describe() method returns a description of the data in the DataFrame. Here, it will retrieve the information like count, unique, top and frequency as the data type is an object.

[29]:
```
data.describe(include = 'O')
```

[29]:

	V12	V13	V28	V29	V30	V31	V32	V33	Class
count	1941	1941	1941	1941	1941	1941	1941	1941	1941
unique	2	2	2	2	2	2	2	2	2
top	0	1	0	0	0	0	0	0	1
freq	1164	1164	1783	1751	1550	1869	1886	1539	1268

All the object data types are assigned to data_cat field and all the data types excluding object data type are assigned to data_num

[30]:
```
data_cat = data.select_dtypes(include='object').columns
data_num = data.select_dtypes(exclude='object').columns
```

Verifying the null values in the dataset through heatmap and we observe that there is no missing data

[31]:
```
sns.heatmap(data.isnull(),yticklabels=False,cbar=False)
plt.title('the missing values distribution in the data',fontsize=16)
plt.show()
```

Grubb's Test for Outlier Detection Grubbs' test is defined for the hypothesis: H0: There are no outliers in the data set Ha: There is exactly one outlier in the data set Test Statistic: The Grubbs' test statistic is defined as: G=max|Yi−Y¯|s with Y¯ and s denoting the sample mean and standard deviation, respectively. The Grubbs' test statistic is the largest absolute deviation from the sample mean in units of the sample standard deviation.

[32]:
```
#Grubbs Test Function

def grubbs_test(x):
  global out
  n=len(x)
  mean_x = np.mean(x)
  sd_x = np.std(x)
  num = max(abs(x-mean_x))
  g_calculated = num/sd_x
  print("Grubbs Calcuated Value is : ",round(g_calculated,2))
  t_value = sp.stats.t.ppf((1-0.05)/(2*n), n-2)
  g_critical = ((n - 1)*np.sqrt(np.square(t_value)))/(np.sqrt(n)*np.sqrt(n - 2 + np.square(t_value)))
  print("Grubbs Critical Value is :",round(g_critical,2))
  if g_critical > g_calculated :
    out = "No outliers exsist"
```

```
    print(out)
  else :
    out = "Outliers exsist"
    print(out)
```

Making a list of numerical varaibles and plotting outliers in each variable and verifying if the outliers exist are not.

[33]:
```
data_num_out = list()
for var in data_num:
  fig = px.violin(data, y=var, box=True, points='all')
  fig.show()
  print(grubbs_test(data[var]))
  if out == "Outliers exsist":
    data_num_out.append(str(var))
```

```
Grubbs Calcuated Value is :  2.18
Grubbs Critical Value is : 3.48
No outliers exsist
None

Grubbs Calcuated Value is :  2.2
Grubbs Critical Value is : 3.48
No outliers exsist
None

Grubbs Calcuated Value is :  6.39
Grubbs Critical Value is : 3.48
Outliers exsist
None

Grubbs Calcuated Value is :  6.39
Grubbs Critical Value is : 3.48
Outliers exsist
None

Grubbs Calcuated Value is :  29.18
Grubbs Critical Value is : 3.48
Outliers exsist
None

Grubbs Calcuated Value is :  34.33
Grubbs Critical Value is : 3.48
Outliers exsist
None

Grubbs Calcuated Value is :  42.38
Grubbs Critical Value is : 3.48
Outliers exsist
None
```

```
Grubbs Calcuated Value is :  22.23
Grubbs Critical Value is : 3.48
Outliers exsist
None

Grubbs Calcuated Value is :  3.69
Grubbs Critical Value is : 3.48
Outliers exsist
None

Grubbs Calcuated Value is :  6.57
Grubbs Critical Value is : 3.48
Outliers exsist
None

Grubbs Calcuated Value is :  2.32
Grubbs Critical Value is : 3.48
No outliers exsist
None

Grubbs Calcuated Value is :  4.02
Grubbs Critical Value is : 3.48
Outliers exsist
None

Grubbs Calcuated Value is :  2.21
Grubbs Critical Value is : 3.48
No outliers exsist
None

Grubbs Calcuated Value is :  3.86
Grubbs Critical Value is : 3.48
Outliers exsist
None

Grubbs Calcuated Value is :  2.08
Grubbs Critical Value is : 3.48
No outliers exsist
None

Grubbs Calcuated Value is :  14.29
Grubbs Critical Value is : 3.48
Outliers exsist
None

Grubbs Calcuated Value is :  2.45
Grubbs Critical Value is : 3.48
No outliers exsist
None

Grubbs Calcuated Value is :  3.27
Grubbs Critical Value is : 3.48
```

```
No outliers exsist
None

Grubbs Calcuated Value is :  1.19
Grubbs Critical Value is : 3.48
No outliers exsist
None

Grubbs Calcuated Value is :  3.41
Grubbs Critical Value is : 3.48
No outliers exsist
None

Grubbs Calcuated Value is :  3.61
Grubbs Critical Value is : 3.48
Outliers exsist
None

Grubbs Calcuated Value is :  6.29
Grubbs Critical Value is : 3.48
Outliers exsist
None

Grubbs Calcuated Value is :  2.15
Grubbs Critical Value is : 3.48
No outliers exsist
None

Grubbs Calcuated Value is :  5.83
Grubbs Critical Value is : 3.48
Outliers exsist
None

Grubbs Calcuated Value is :  1.37
Grubbs Critical Value is : 3.48
No outliers exsist
None
```

Removing variable V14 as we need the variable to test

[34]:
```python
data_num_out.remove("V14")
```

Winsorization is the transformation of statistics by limiting extreme values in the statistical data to reduce the effect of possibly spurious outliers. With winsorizing, any value of a variable above or below a percentile k on each side of the variables' distribution is replaced with the value of the k-th percentile itself.

[35]:
```python
def winsor_outliers(df):
  global win_out
  win_out = []
  q1 = df.quantile(0.01)
  q3 = df.quantile(0.99)
```

```
for i in df :
  if i>q3 or i<q1:
    win_out.append(i)
print("q1:",q1 , "q2:",q3)
print("Outliers : ", win_out)
print("The number of outliers is : ",len(win_out))
```

[36]:

```
for var in data_num_out:
  print(var)
  print(winsor_outliers(data[var]))

  data[var] = data[var].replace(
    to_replace=win_out,
    value = "None" )
```

```
V3
q1: 28533.0 q2: 10359325.199999997
Outliers :  [21349, 19815, 11430396, 11741476, 12577343, 12725281, 12917033,
12987661, 21512, 9007, 13302, 19000, 7430, 15184, 10391495, 10507433, 7851,
11150448, 12438460, 12806495, 6712, 11066410, 11499942, 11569824, 12416454,
23288, 10369596, 10409376, 10440356, 10555505, 10624922, 28527, 14524, 9228,
12799, 18324, 23012, 7003, 15755, 21104]
The number of outliers is :  40
None
V4
q1: 28546.4 q2: 10359343.199999997
Outliers :  [21376, 19841, 11430416, 11741833, 12577396, 12725314, 12917094,
12987692, 21518, 9033, 13320, 19123, 7458, 15196, 10391507, 10507445, 7865,
11150470, 12438491, 12806520, 6724, 11066424, 11499957, 11569844, 12416473,
23316, 10369620, 10409388, 10440367, 10555515, 10624934, 28542, 14551, 9246,
12804, 18328, 23019, 7020, 15765, 21135]
The number of outliers is :  40
None
V5
q1: 12.800000000000004 q2: 18033.39999999999
Outliers :  [152655, 25323, 21110, 25473, 24365, 20894, 20726, 22554, 21987,
21036, 19629, 18517, 18071, 18908, 18896, 20313, 18203, 18954, 2, 2, 12, 12, 12,
12, 12, 12, 12, 12, 12, 11, 12, 9, 8, 8, 9, 6, 6, 10, 37334, 19818]
The number of outliers is :  40
None
V6
q1: 5.400000000000002 q2: 863.3999999999996
Outliers :  [10449, 1022, 1138, 992, 1084, 1169, 1193, 999, 1050, 1015, 1021,
926, 929, 867, 865, 976, 2, 2, 943, 4, 5, 4, 5, 5, 5, 5, 3, 4, 4, 3, 4, 5, 5, 5,
5, 5, 1275, 4, 908, 905]
The number of outliers is :  40
None
```

```
V7
q1: 4.0 q2: 539.0
Outliers :  [604, 18152, 593, 597, 578, 680, 712, 709, 605, 684, 586, 696, 562,
591, 568, 541, 557, 1, 1, 3, 3, 3, 3, 3, 3, 2, 3, 903, 583, 3, 3, 2]
The number of outliers is :  32
None
V8
q1: 1551.0 q2: 2155800.8
Outliers :  [11591414, 3037459, 2554885, 3061597, 2935414, 2529140, 2499819,
2712104, 2638402, 2519511, 2332320, 2156667, 2230510, 2160349, 2264960, 2225392,
2400588, 2155986, 2236201, 764, 255, 250, 1537, 1543, 1528, 1504, 1541, 1539,
1509, 1398, 1522, 1335, 1063, 958, 950, 1059, 718, 775, 1233, 3918209]
The number of outliers is :  40
None
V9
q1: 19.0 q2: 170.79999999999973
Outliers :  [11, 15, 16, 178, 195, 172, 196, 192, 195, 178, 178, 177, 6, 0, 18,
9, 12, 6, 4, 203, 7, 11, 0, 16, 0, 0, 178, 191, 190, 179, 172, 175, 179, 172,
173, 192, 14]
The number of outliers is :  37
None
V10
q1: 84.0 q2: 199.0
Outliers :  [78, 78, 79, 71, 77, 77, 70, 207, 206, 221, 213, 212, 236, 210, 207,
212, 207, 205, 82, 252, 252, 220, 253, 79, 247, 37, 39, 71, 78, 71, 207, 207,
212]
The number of outliers is :  33
None
V16
q1: 0.13340000000000002 q2: 0.75552
Outliers :  [0.0595, 0.7566, 0.8648, 0.1169, 0.0278, 0.0972, 0.0818, 0.0992,
0.9439, 0.7612, 0.8767, 0.8429, 0.7906, 0.8817, 0.894, 0.8268, 0.8473, 0.0, 0.0,
0.8487, 0.8856, 0.1136, 0.0781, 0.1286, 0.1, 0.1272, 0.9275, 0.0368, 0.0926,
0.7878, 0.0682, 0.1, 0.8011, 0.1198, 0.0714, 0.7718, 0.83, 0.8888, 0.7558,
0.1111]
The number of outliers is :  40
None
V18
q1: 0.0035 q2: 0.19035999999999853
Outliers :  [0.3105, 0.5906, 0.2739, 0.4957, 0.296, 0.4964, 0.6209, 0.2868,
0.2511, 0.0015, 0.0015, 0.5692, 0.6226, 0.0029, 0.003, 0.003, 0.003, 0.0022,
0.0029, 0.0029, 0.0022, 0.003, 0.003, 0.3878, 0.003, 0.0029, 0.2537, 0.4698,
0.4177, 0.3466, 0.8759, 0.2175, 0.2979, 0.0022, 0.0024, 0.1968]
The number of outliers is :  36
None
V23
q1: 0.699 q2: 2.4394599999999933
Outliers :  [2.6395, 2.918, 2.5843, 2.8414, 2.6181, 2.842, 2.9385, 2.6031,
```

```
2.5465, 0.301, 0.301, 2.8882, 2.9335, 0.6021, 0.6021, 0.6021, 0.6021, 0.4771,
0.6021, 0.6021, 0.4771, 0.6021, 0.6021, 2.7235, 0.6021, 2.5378, 2.8048, 2.7543,
2.6721, 3.0741, 2.4683, 2.6064, 0.4771, 0.6021, 2.5224]
The number of outliers is :  35
None
V24
q1: 0.4771 q2: 2.4459199999999996
Outliers :  [2.6181, 2.4487, 2.776, 2.6294, 2.5515, 2.5527, 2.4829, 2.5922,
2.4683, 2.4487, 4.2587, 0.0, 0.0, 0.301, 2.4472, 2.5752, 0.301, 2.5052, 2.4594,
0.301, 2.4928, 2.6149, 2.5011, 2.4533, 2.5752, 0.301, 0.301, 0.301]
The number of outliers is :  28
None
V26
q1: -0.54368 q2: 0.42629999999999973
Outliers :  [-0.5528, -0.5816, -0.5678, -0.6096, -0.5644, -0.5902, -0.555,
0.5237, 0.46, 0.5916, 0.4946, 0.5917, 0.5909, 0.5613, 0.5799, 0.4976, 0.4831,
0.4569, 0.4504, -0.9989, -0.5971, -0.566, 0.6421, -0.6332, -0.585, -0.8603,
-0.885, -0.5462, 0.4379, -0.6017, -0.5754, -0.594, 0.5518, 0.5552, 0.4573,
0.4545, 0.4275, 0.5591, -0.5454, -0.5455]
The number of outliers is :  40
None
```

[37]:
```python
data=data.replace("None", np.nan) #replacing none values to NAN
sns.heatmap(data.isnull(),yticklabels=False,cbar=False)
plt.title('Data after Outliers are removed',fontsize=16)
plt.show()
```

[38]:
```
data_colname = data.columns
data_colname
```

[38]:
```
Index(['V1', 'V2', 'V3', 'V4', 'V5', 'V6', 'V7', 'V8', 'V9', 'V10', 'V11',
       'V12', 'V13', 'V14', 'V15', 'V16', 'V17', 'V18', 'V19', 'V20', 'V21',
       'V22', 'V23', 'V24', 'V25', 'V26', 'V27', 'V28', 'V29', 'V30', 'V31',
       'V32', 'V33', 'Class'],
      dtype='object')
```

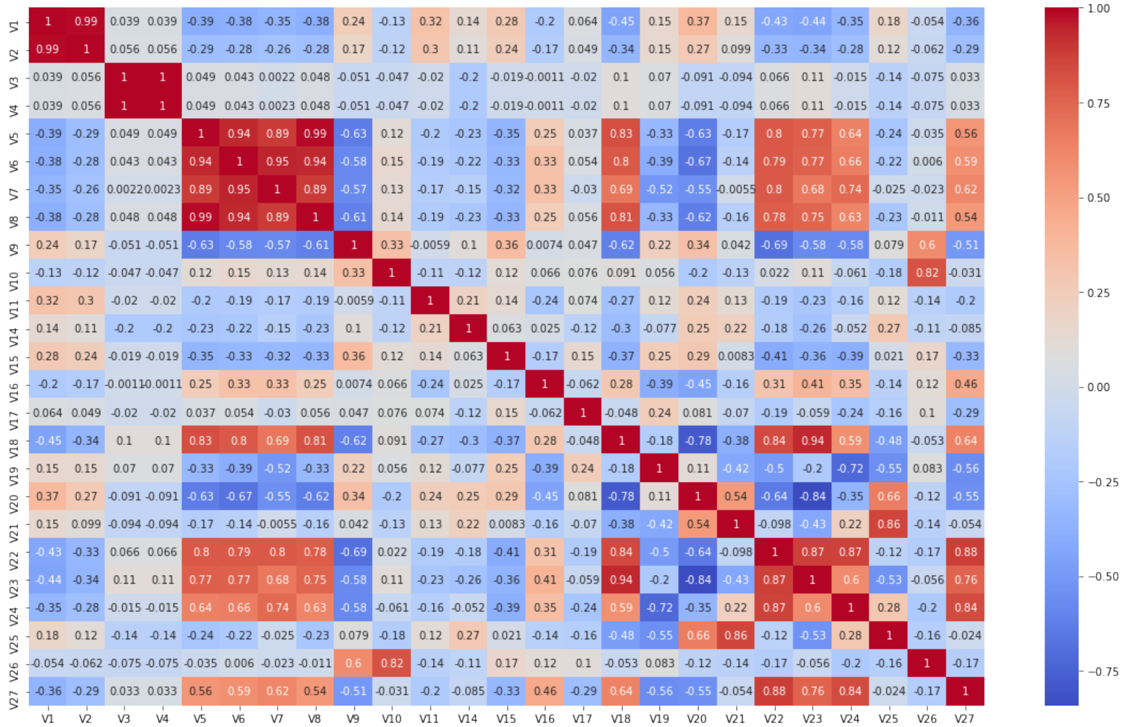
Replace NaN values with Knn Imputer

[39]:
```
from sklearn.impute import KNNImputer
imputer = KNNImputer(n_neighbors=5)
data_imputer = imputer.fit_transform(data)
data_imputer = pd.DataFrame(data_imputer,columns =data_colname)
data = data_imputer
```

[40]:
```
data_object = []
data_objects = data.nunique()
data_objects = pd.DataFrame(data_objects)
data_objects = data_objects.iloc[0:]
data_objects.reset_index(inplace=True)
data_objects.columns = ["Objects","Unique_count"]
data_objects_select = data_objects.loc[data_objects['Unique_count'] == 2]
data_objects_select = list(data_objects_select.Objects)
for i in data_objects_select :
  data[i] = data[i].astype("object")
```

[41]:
```
plt.subplots(figsize =(20, 12))
sns.heatmap(data.corr(), annot=True,cmap='coolwarm')
```

[41]: <AxesSubplot:>



[42]:
```
corrmat = data.corr()
```

[43]:
```
def getcorel(data,threshold):
  corr_col=set()
  corrmat=data.corr()
  for i in range (len(corrmat.columns)):
    for j in range(i):
      if abs(corrmat.iloc[i,j])>threshold :
        colname = corrmat.columns [i]
        corr_col.add(colname)
  return corr_col
```

[44]:
```
X = data.drop("Class", axis = 1)
y = data["Class"]
```

[45]:
```
corr_feature = getcorel(X,0.95)
corr_feature
```

[45]:
```
{'V2', 'V4', 'V8'}
```

[46]:
```
corrdata = corrmat.abs().stack()
corrdata = corrdata.sort_values(ascending=False)
corrdata = corrdata[corrdata>0.95]
corrdata = corrdata[corrdata<1]
```

```
corrdata = pd.DataFrame(corrdata).reset_index()
corrdata.columns = ["features1","features2","corr_value"]
corrdata
```

[46]:

	features1	features2	corr_value
0	V4	V3	1.000000
1	V3	V4	1.000000
2	V8	V5	0.992703
3	V5	V8	0.992703
4	V1	V2	0.988314
5	V2	V1	0.988314

[47]:

```
grouped_feature_list = []
correlated_groups_list = []

for feature in corrdata.features1.unique():
  if feature not in grouped_feature_list:
    correlated_block = corrdata[corrdata.features1 == feature]
    grouped_feature_list = grouped_feature_list + list(correlated_block.
↪features2.unique()) + [feature]
    correlated_groups_list.append(correlated_block)
```

[48]:

```
correlated_groups_list

for group in correlated_groups_list:
  print(group)
```

```
  features1 features2  corr_value
0        V4        V3         1.0
  features1 features2  corr_value
2        V8        V5    0.992703
  features1 features2  corr_value
4        V1        V2    0.988314
```

[49]:

```
# Feature importance using RF classifier

from sklearn.ensemble import RandomForestClassifier
important_features = []
for group in correlated_groups_list:
    features = list(group.features1.unique()) + list(group.features2.unique())
    #features = col_names[features]
    rf = RandomForestClassifier(n_estimators=100, random_state=0)
    y= y.astype('int')
    rf.fit(data[features], y)

    importance = pd.concat([pd.Series(features), pd.Series(rf.
↪feature_importances_)], axis = 1)
    importance.columns = ['features', 'importance']
```

```
    importance.sort_values(by = 'importance', ascending = False, inplace = True)
    feat = importance.iloc[0]
    important_features.append(feat)
```

[50]:
```
important_features
```

[50]:
```
[features            V4
 importance    0.502034
 Name: 0, dtype: object,
 features            V8
 importance    0.575011
 Name: 0, dtype: object,
 features            V2
 importance    0.511688
 Name: 1, dtype: object]
```

[51]:
```
X.drop(["V1","V3","V5"],axis = 1, inplace = True)
data.drop(["V1","V3","V5"],axis = 1, inplace = True)
```

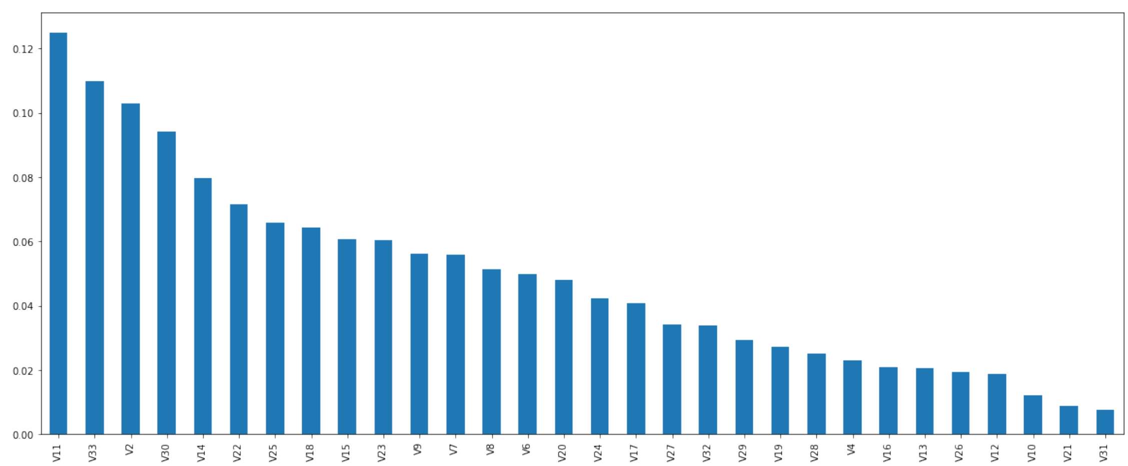
[52]:
```
#feature selection based on mutual information

from sklearn.feature_selection import VarianceThreshold, mutual_info_classif,␣
 ↪mutual_info_regression

mi = mutual_info_classif(X,y.astype(int))
mi = pd.Series(mi)
mi.index = X.columns
mi.sort_values (ascending = False, inplace = True)
```

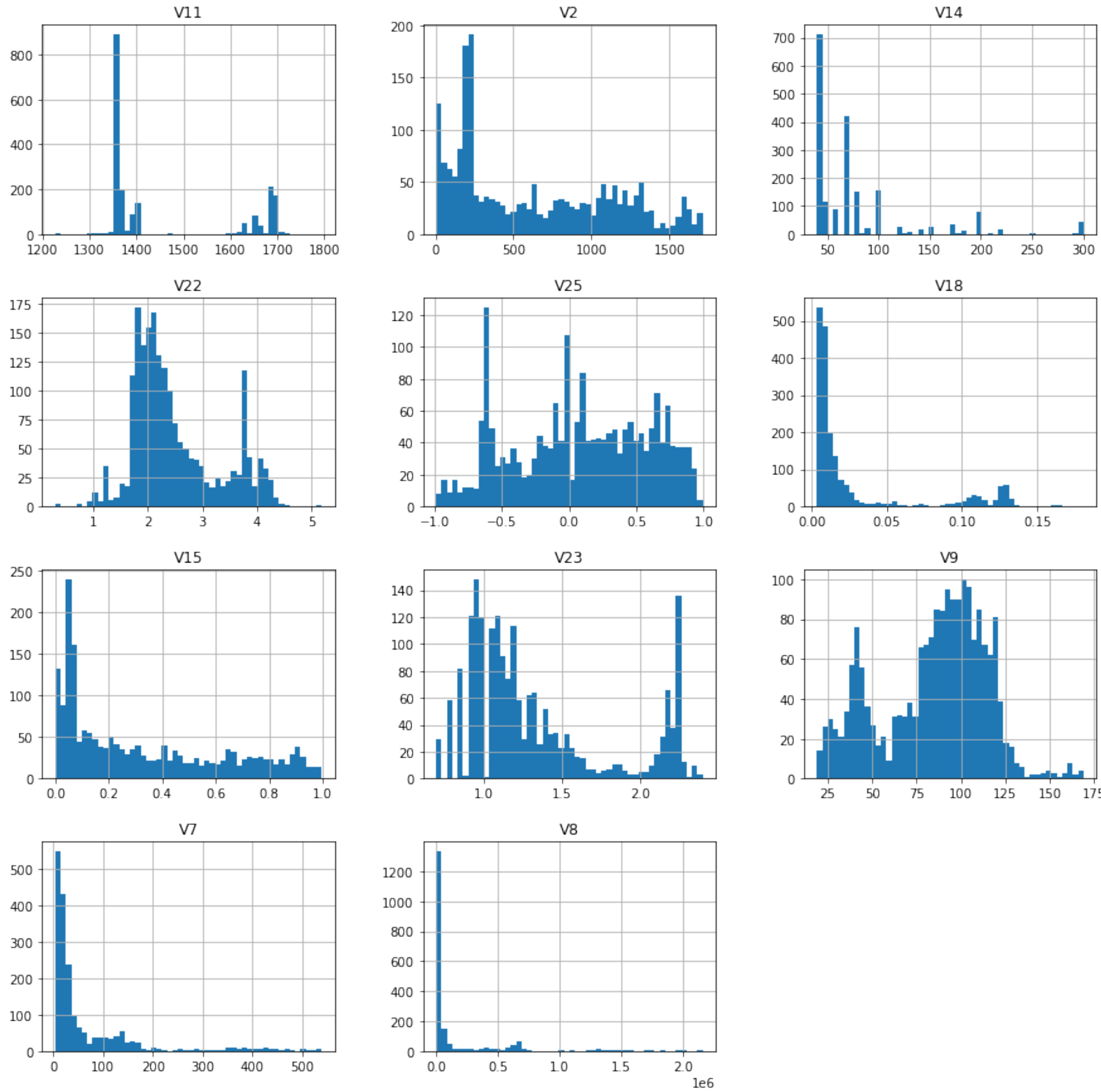
[53]:
```
mi.plot.bar(figsize = (20,8))
```

[53]:
```
<AxesSubplot:>
```



[54]:
```
midf = mi.to_frame('importance')
midf = midf[midf['importance'] >= 0.05]
keep = midf.index.values.tolist()
X = data[keep]
```

[55]:
```
X.hist(bins=50, figsize=(15,15))
plt.show()
```



[56]:
```
# #Normalizing Data
X_colname = X.columns
from sklearn import preprocessing
```

```
scaler = preprocessing.RobustScaler()
robust_df = preprocessing.RobustScaler(unit_variance=True).fit_transform(X)
robust_df = pd.DataFrame(robust_df,columns =X_colname)
X=robust_df
```

[57]:
```
# X_colname = X.columns
# from sklearn import preprocessing
# scale_df = preprocessing.MinMaxScaler(feature_range = (-1,1)).fit_transform(X)
# scale_df = pd.DataFrame(scale_df,columns =X_colname)
# X=scale_df
```

[58]:
```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X,y,test_size=0.
 ↪3,random_state=10)
```

1 Models

[59]:
```
from sklearn.metrics import classification_report
from sklearn.metrics import roc_curve, roc_auc_score
from sklearn.metrics import confusion_matrix
```

Naive Bayes Prediction and Score

[60]:
```
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import accuracy_score, balanced_accuracy_score,␣
 ↪cohen_kappa_score
from sklearn.model_selection import cross_val_score
nb = GaussianNB()
nb.fit(X_train, y_train)
y_predNB = nb.predict(X_test)

# Summary of the predictions made by the classifier
print(classification_report(y_test, y_predNB))
print('confusion matrix')
print(confusion_matrix(y_test, y_predNB))
print()
# Accuracy score
print('balanced accuracy score is ',balanced_accuracy_score(y_test,y_predNB))
print('cohen kappa score is ',cohen_kappa_score(y_test,y_predNB))
print()

NBB = balanced_accuracy_score(y_test,y_predNB)

#Cross Validation


NBcv = GaussianNB()
scoresNB = cross_val_score(NBcv, X_train, y_train, cv=10, scoring = "accuracy")
```

```
print('cross validation')
print("Scores:", scoresNB)
print()
print("Mean:", scoresNB.mean())
print("Standard Deviation:", scoresNB.std())
```

```
              precision    recall  f1-score   support

           0       0.57      0.99      0.72       203
           1       0.99      0.61      0.75       380

    accuracy                           0.74       583
   macro avg       0.78      0.80      0.74       583
weighted avg       0.84      0.74      0.74       583

confusion matrix
[[200   3]
 [150 230]]

balanced accuracy score is  0.7952424163857921
cohen kappa score is  0.5052717985124708

cross validation
Scores: [0.74264706 0.78676471 0.76470588 0.75735294 0.79411765 0.78676471
 0.69117647 0.73529412 0.78518519 0.73333333]

Mean: 0.7577342047930282
Standard Deviation: 0.030932369498036186
```

KNN Prediction and Score

[61]:
```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import cross_validate
from sklearn.model_selection import cross_val_score
from sklearn.metrics import accuracy_score, balanced_accuracy_score,␣
 ↪cohen_kappa_score


knn = KNeighborsClassifier(n_neighbors=7)
knn.fit(X_train,y_train)

y_predKN = knn.predict (X_test)
# Summary of the predictions made by the classifier
print(classification_report(y_test, y_predKN))
print('confusion matrix')
print(confusion_matrix(y_test, y_predKN))
print()
# Accuracy score
```

```
print('balanced accuracy is ',balanced_accuracy_score(y_test,y_predKN))
print('cohen kappa score is ',cohen_kappa_score(y_test,y_predKN))
print()

KNN = balanced_accuracy_score(y_test,y_predKN)

#Cross Validation

KNcv = KNeighborsClassifier(n_neighbors=3)
scoresKN = cross_val_score(KNcv, X_train, y_train, cv=10, scoring = "accuracy")
print('cross validation')
print("Scores:", scoresKN)
print()
print("Mean:", scoresKN.mean())
print("Standard Deviation:", scoresKN.std())
```

```
              precision    recall  f1-score   support

           0       0.82      0.82      0.82       203
           1       0.90      0.90      0.90       380

    accuracy                           0.87       583
   macro avg       0.86      0.86      0.86       583
weighted avg       0.87      0.87      0.87       583

confusion matrix
[[166  37]
 [ 37 343]]

balanced accuracy is  0.8601827845475758
cohen kappa score is  0.7203655690951516

cross validation
Scores: [0.92647059 0.86764706 0.85294118 0.82352941 0.875      0.83088235
 0.84558824 0.86029412 0.88148148 0.8962963 ]

Mean: 0.8660130718954248
Standard Deviation: 0.02924837615575747
```

Decision Tree Prediction and Score

[62]:
```
# Decision Tree
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import cross_val_score
from sklearn.metrics import accuracy_score, balanced_accuracy_score,␣
↪cohen_kappa_score
```

```python
DT = DecisionTreeClassifier()
DT.fit(X_train,y_train)
y_predDT = DT.predict(X_test)

# Summary of the predictions made by the classifier
print(classification_report(y_test, y_predDT))
print('confusion matrix')
print(confusion_matrix(y_test, y_predDT))

print()
# Accuracy score
print('balanced accuracy is ',balanced_accuracy_score(y_test,y_predDT))
print('cohen kappa score is ',cohen_kappa_score(y_test,y_predDT))
print()

DT = balanced_accuracy_score(y_test,y_predDT)

#Cross Validation

DTcv = DecisionTreeClassifier()
scoresDT = cross_val_score(DTcv, X_train, y_train, cv=10, scoring = "accuracy")
print('cross validation')
print("Scores:", scoresDT)
print()
print("Mean:", scoresDT.mean())
print("Standard Deviation:", scoresDT.std())
```

```
              precision    recall  f1-score   support

           0       0.81      0.85      0.83       203
           1       0.92      0.89      0.90       380

    accuracy                           0.88       583
   macro avg       0.86      0.87      0.87       583
weighted avg       0.88      0.88      0.88       583

confusion matrix
[[172  31]
 [ 41 339]]

balanced accuracy is  0.8696979517759917
cohen kappa score is  0.7310092918936238

cross validation
Scores: [0.88235294 0.90441176 0.84558824 0.84558824 0.88235294 0.85294118
 0.88235294 0.93382353 0.88888889 0.81481481]

Mean: 0.8733115468409587
```

```
Standard Deviation: 0.03238950641608347
```

LDA

[63]:
```python
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
from sklearn.model_selection import cross_val_score
from sklearn.metrics import accuracy_score, balanced_accuracy_score,␣
↪cohen_kappa_score

lda = LinearDiscriminantAnalysis()
lda.fit(X_train, y_train)
y_predLD = lda.predict(X_test)

# Summary of the predictions made by the classifier
print(classification_report(y_test, y_predLD))
print('confusion matrix')
print(confusion_matrix(y_test, y_predLD))

print()

# Accuracy score
print('Balanced accuracy is',balanced_accuracy_score(y_test,y_predLD))
print('cohen kappa score is',cohen_kappa_score(y_test,y_predLD))
print()

LDA = balanced_accuracy_score(y_test,y_predLD)

#Cross Validation

ldacv = LinearDiscriminantAnalysis()
scoresLDA = cross_val_score(ldacv, X_train, y_train, cv=10, scoring =␣
↪"accuracy")
print('cross validation')
print("Scores:", scoresLDA)
print()
print("Mean:", scoresLDA.mean())
print("Standard Deviation:", scoresLDA.std())
```

```
              precision    recall  f1-score   support

           0       0.65      0.80      0.72       203
           1       0.88      0.77      0.82       380

    accuracy                           0.78       583
   macro avg       0.77      0.79      0.77       583
weighted avg       0.80      0.78      0.79       583

confusion matrix
```

```
[[163  40]
 [ 86 294]]

Balanced accuracy is 0.7883199377754732
cohen kappa score is 0.5477336813978402

cross validation
Scores: [0.83823529 0.80882353 0.83088235 0.85294118 0.85294118 0.80147059
 0.80882353 0.82352941 0.83703704 0.77777778]

Mean: 0.8232461873638345
Standard Deviation: 0.022791261407323286
```

Nu-SVC

[64]:
```python
from sklearn.svm import NuSVC
from sklearn.model_selection import cross_val_score
from sklearn.metrics import accuracy_score, balanced_accuracy_score,␣
↪cohen_kappa_score

NUS = NuSVC()
NUS.fit(X_train, y_train)
y_predNU = NUS.predict(X_test)

# Summary of the predictions made by the classifier
print(classification_report(y_test, y_predNU))
print('confusion matrix')
print(confusion_matrix(y_test, y_predNU))

print()

# Accuracy score
print('Balanced accuracy is',balanced_accuracy_score(y_test,y_predNU))
print('cohen kappa score is',cohen_kappa_score(y_test,y_predNU))
print()

NUS = balanced_accuracy_score(y_test,y_predNU)

#Cross Validation
NuCv = NuSVC()
scoresNUS = cross_val_score(NuCv, X_train, y_train, cv=10, scoring = "accuracy")
print("Scores:", scoresNUS)
print()
print("Mean:", scoresNUS.mean())
print("Standard Deviation:", scoresNUS.std())
```

```
              precision    recall  f1-score   support
```

```
           0       0.76      0.73      0.74       203
           1       0.86      0.87      0.87       380

    accuracy                           0.82       583
   macro avg       0.81      0.80      0.80       583
weighted avg       0.82      0.82      0.82       583

confusion matrix
[[148  55]
 [ 48 332]]

Balanced accuracy is 0.8013741249675914
cohen kappa score is 0.6076280212492078

Scores:  [0.85294118 0.82352941 0.875      0.81617647 0.85294118 0.82352941
 0.875      0.83823529 0.86666667 0.81481481]

Mean: 0.8438834422657951
Standard Deviation: 0.022567181517879016
```

Extra Trees Classfier

[65]:
```
from sklearn.ensemble import ExtraTreesClassifier
from sklearn.model_selection import cross_val_score
from sklearn.metrics import accuracy_score, balanced_accuracy_score,␣
↪cohen_kappa_score

excla = ExtraTreesClassifier(n_estimators=10,␣
↪max_depth=None,min_samples_split=2)
excla.fit(X_train, y_train)
y_predET = excla.predict(X_test)

# Summary of the predictions made by the classifier
print(classification_report(y_test, y_predET))
print('confusion matrix')
print(confusion_matrix(y_test, y_predET))

print()

# Accuracy score
print('Balanced accuracy is',balanced_accuracy_score(y_test,y_predET))
print('cohen kappa score is',cohen_kappa_score(y_test,y_predET))
print()

ETC = balanced_accuracy_score(y_test,y_predET)

#Cross Validation
```

```
exclacv = ExtraTreesClassifier(n_estimators=10,␣
↪max_depth=None,min_samples_split=2, random_state=0)
scoreET = cross_val_score(exclacv, X_train, y_train, cv=10, scoring =␣
↪"accuracy")
print("Scores:", scoreET)
print()
print("Mean:", scoreET.mean())
print("Standard Deviation:", scoreET.std())
```

```
              precision    recall  f1-score   support

           0       0.80      0.88      0.84       203
           1       0.93      0.88      0.91       380

    accuracy                           0.88       583
   macro avg       0.87      0.88      0.87       583
weighted avg       0.89      0.88      0.88       583

confusion matrix
[[179  24]
 [ 44 336]]

Balanced accuracy is 0.8829919626652839
cohen kappa score is 0.7488024331516918

Scores: [0.875      0.89705882 0.91176471 0.83823529 0.91176471 0.86029412
 0.89705882 0.91176471 0.9037037  0.85925926]

Mean: 0.8865904139433551
Standard Deviation: 0.02514819542798188
```

Gradient Boost

[66]:
```
from sklearn.ensemble import GradientBoostingClassifier
ModelG=GradientBoostingClassifier()
ModelG.fit(X_train, y_train)
y_predGR=ModelG.predict(X_test)

# Summary of the predictions made by the classifier
print(classification_report(y_test, y_predGR))
print('confusion matrix')
print(confusion_matrix(y_test, y_predGR))

# Accuracy score
print('Balanced accuracy is',balanced_accuracy_score(y_test,y_predGR))
print('cohen kappa score is',cohen_kappa_score(y_test,y_predGR))
print()
```

```
GR = balanced_accuracy_score(y_test,y_predGR)

#Cross Validation
GRcv = GradientBoostingClassifier()
scoreGR = cross_val_score(GRcv, X_train, y_train, cv=10, scoring = "accuracy")
print("Scores:", scoreGR)
print()
print("Mean:", scoreGR.mean())
print("Standard Deviation:", scoreGR.std())
```

```
              precision    recall  f1-score   support

           0       0.84      0.89      0.86       203
           1       0.94      0.91      0.92       380

    accuracy                           0.90       583
   macro avg       0.89      0.90      0.89       583
weighted avg       0.90      0.90      0.90       583

confusion matrix
[[180  23]
 [ 34 346]]
Balanced accuracy is 0.8986129115893182
cohen kappa score is 0.78729028913056

Scores: [0.94117647 0.90441176 0.90441176 0.89705882 0.91176471 0.86029412
 0.875      0.94117647 0.92592593 0.9037037 ]

Mean: 0.9064923747276689
Standard Deviation: 0.02458232794712384
```

Logistic Regression

[67]:
```
# LogisticRegression
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import cross_val_score
from sklearn.metrics import accuracy_score, balanced_accuracy_score,␣
 ↪cohen_kappa_score

LR = LogisticRegression()
LR.fit(X_train, y_train)
y_predLR = LR.predict(X_test)

# Summary of the predictions made by the classifier
print(classification_report(y_test, y_predLR))
print('confusion matrix')
print(confusion_matrix(y_test, y_predLR))
```

```
# Accuracy score
print('Balanced accuracy is',balanced_accuracy_score(y_test,y_predLR))
print('cohen kappa score is',cohen_kappa_score(y_test,y_predLR))
print()

LR = balanced_accuracy_score(y_test,y_predLR)
#Cross Validation
LRcv = LogisticRegression()
scoreLR = cross_val_score(LRcv, X_train, y_train, cv=10, scoring = "accuracy")
print("Scores:", scoreLR)
print()
print("Mean:", scoreLR.mean())
print("Standard Deviation:", scoreLR.std())
```

```
              precision    recall  f1-score   support

           0       0.71      0.78      0.74       203
           1       0.88      0.83      0.85       380

    accuracy                           0.81       583
   macro avg       0.79      0.80      0.80       583
weighted avg       0.82      0.81      0.81       583

confusion matrix
[[158  45]
 [ 65 315]]
Balanced accuracy is 0.8036362457868811
cohen kappa score is 0.5936509948042074

Scores: [0.83088235 0.82352941 0.83088235 0.86029412 0.84558824 0.80147059
 0.82352941 0.80147059 0.84444444 0.8       ]

Mean: 0.8262091503267973
Standard Deviation: 0.019592886452451025
```

Random forest

[68]:
```
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import cross_val_score
from sklearn.metrics import accuracy_score, balanced_accuracy_score,␣
 ↪cohen_kappa_score

rf=RandomForestClassifier()
rf.fit(X_train, y_train)
y_predR=rf.predict(X_test)

# Summary of the predictions made by the classifier
print(classification_report(y_test,y_predR))
```

```
print('confusion matrix')
print(confusion_matrix(y_predR,y_test))

#Accuracy Score
print('Balanced accuracy is',balanced_accuracy_score(y_test,y_predR))
print('cohen kappa score is',cohen_kappa_score(y_test,y_predR))
print()

RF = balanced_accuracy_score(y_test,y_predR)

#Cross Validation
rfcv = RandomForestClassifier(max_depth=2)
scoreRF = cross_val_score(rfcv, X_train, y_train, cv=10, scoring = "accuracy")
print("Scores:", scoreRF)
print()
print("Mean:", scoreRF.mean())
print("Standard Deviation:", scoreRF.std())
```

```
              precision    recall  f1-score   support

           0       0.85      0.90      0.87       203
           1       0.94      0.91      0.93       380

    accuracy                           0.91       583
   macro avg       0.89      0.90      0.90       583
weighted avg       0.91      0.91      0.91       583

confusion matrix
[[182  33]
 [ 21 347]]
Balanced accuracy is 0.9048548094373865
cohen kappa score is 0.7987135878877778

Scores: [0.76470588 0.71323529 0.76470588 0.75       0.72058824 0.70588235
 0.75       0.75735294 0.71851852 0.74074074]

Mean: 0.7385729847494552
Standard Deviation: 0.02102547208073098
```

ADABOOST

[69]:

```
from sklearn.ensemble import AdaBoostClassifier
from sklearn.model_selection import cross_val_score
from sklearn.metrics import accuracy_score, balanced_accuracy_score,␣
 ↪cohen_kappa_score

ada=AdaBoostClassifier()
ada.fit(X_train, y_train)
```

```
y_predAD=ada.predict(X_test)

# Summary of the predictions made by the classifier
print(classification_report(y_test,y_predAD))
print('confusion matrix')
print(confusion_matrix(y_predAD,y_test))

#Accuracy Score
print('Balanced accuracy is',balanced_accuracy_score(y_test,y_predAD))
print('cohen kappa score is',cohen_kappa_score(y_test,y_predAD))
print()

ADA = balanced_accuracy_score(y_test,y_predAD)

#Cross Validation
adacv=AdaBoostClassifier()
scoreADA = cross_val_score(adacv, X_train, y_train, cv=10, scoring = "accuracy")
print("Scores:", scoreADA)
print()
print("Mean:", scoreADA.mean())
print("Standard Deviation:", scoreADA.std())
```

```
              precision    recall  f1-score   support

           0       0.83      0.85      0.84       203
           1       0.92      0.91      0.91       380

    accuracy                           0.89       583
   macro avg       0.87      0.88      0.87       583
weighted avg       0.89      0.89      0.89       583

confusion matrix
[[172  36]
 [ 31 344]]
Balanced accuracy is 0.8762768991444128
cohen kappa score is 0.7482615280507847

Scores: [0.88235294 0.86764706 0.86029412 0.875      0.94117647 0.88970588
 0.90441176 0.88970588 0.91851852 0.88148148]

Mean: 0.8910294117647058
Standard Deviation: 0.023168401366670033
```

[70]:

```
models = pd.DataFrame({
    'Model': ['Decision Tree','LogisticRegression','K-Nearest Neighbours',␣
 ↪'Naive Bayes','Linear Discriminant Analysis',
```

```python
              'Nu-Support Vector Classification','Extra␣
↪Classifier','Gradiant Boost','Random Forest','AdaBoost'],
   'Score': [DT, LR, KNN, NBB, LDA, NUS, ETC, GR, RF, ADA],
   'CV Mean' : [scoresDT.mean(),scoreLR.mean(),scoresKN.mean(),scoresNB.
↪mean(),scoresLDA.mean(),
                 scoresNUS.mean(),scoreET.mean(),scoreGR.mean(),scoreRF.
↪mean(),scoreADA.mean()],
   'CV Std' : [scoresDT.std(),scoreLR.std(),scoresKN.std(),scoresNB.
↪std(),scoresLDA.std(),
                scoresNUS.std(),scoreET.std(),scoreGR.std(),scoreRF.
↪std(),scoreADA.std()]
   })
models = models.sort_values(by='Score', ascending=False)
models.reset_index(drop=True, inplace=True)
models
```

[70]:

	Model	Score	CV Mean	CV Std
0	Random Forest	0.904855	0.738573	0.021025
1	Gradiant Boost	0.898613	0.906492	0.024582
2	Extra Tree Classifier	0.882992	0.886590	0.025148
3	AdaBoost	0.876277	0.891029	0.023168
4	Decision Tree	0.869698	0.873312	0.032390
5	K-Nearest Neighbours	0.860183	0.866013	0.029248
6	LogisticRegression	0.803636	0.826209	0.019593
7	Nu-Support Vector Classification	0.801374	0.843883	0.022567
8	Naive Bayes	0.795242	0.757734	0.030932
9	Linear Discriminant Analysis	0.788320	0.823246	0.022791

[71]:
```python
from sklearn.metrics import roc_curve, auc
#plt.style.use('seaborn-pastel')
FPR_1, TPR_1, _ = roc_curve(y_test, y_predDT)
FPR_2, TPR_2, _ = roc_curve(y_test, y_predLR)
FPR_3, TPR_3, _ = roc_curve(y_test, y_predNB)
FPR_4, TPR_4, _ = roc_curve(y_test, y_predKN)
FPR_5, TPR_5, _ = roc_curve(y_test, y_predLD)
FPR_6, TPR_6, _ = roc_curve(y_test, y_predNU)
FPR_7, TPR_7, _ = roc_curve(y_test, y_predET)
FPR_8, TPR_8, _ = roc_curve(y_test, y_predGR)
FPR_9, TPR_9, _ = roc_curve(y_test, y_predR)
FPR_10, TPR_10, _ = roc_curve(y_test, y_predAD)

ROC_AUC_1 = auc(FPR_1, TPR_1)
ROC_AUC_2 = auc(FPR_2, TPR_2)
ROC_AUC_3 = auc(FPR_3, TPR_3)
ROC_AUC_4 = auc(FPR_4, TPR_4)
ROC_AUC_5 = auc(FPR_5, TPR_5)
ROC_AUC_6 = auc(FPR_6, TPR_6)
```

```
ROC_AUC_7 = auc(FPR_7, TPR_7)
ROC_AUC_8 = auc(FPR_8, TPR_8)
ROC_AUC_9 = auc(FPR_9, TPR_9)
ROC_AUC_10 = auc(FPR_10, TPR_10)



print ("Desicion Tree ROC is ", ROC_AUC_1)
print ("Logistic Regression ROC is ", ROC_AUC_2)
print ("Naive Bayes ROC is ", ROC_AUC_3)
print ("K-Nearest Neighbours is ", ROC_AUC_4)
print ("Linear Discriminant Analysis is ", ROC_AUC_5)
print ("Nu-Support Classification is ", ROC_AUC_6)
print ("Extra Tree Classifier is ", ROC_AUC_7)
print ("Gradient Boost is ", ROC_AUC_8)
print ("Random Forest is ", ROC_AUC_9)
print ("AdaBoost is ", ROC_AUC_10)


plt.figure(figsize =[11,9])
plt.plot(FPR_1, TPR_1, label= 'DT ROC curve(area = %0.2f)'%ROC_AUC_1,␣
 ↪linewidth= 4)
plt.plot(FPR_2, TPR_2, label= 'LR ROC curve(area = %0.2f)'%ROC_AUC_2,␣
 ↪linewidth= 4)
plt.plot(FPR_3, TPR_3, label= 'NB ROC curve(area = %0.2f)'%ROC_AUC_3,␣
 ↪linewidth= 4)
plt.plot(FPR_4, TPR_4, label= 'KNN ROC curve(area = %0.2f)'%ROC_AUC_4,␣
 ↪linewidth= 4)
plt.plot(FPR_5, TPR_5, label= 'LDA ROC curve(area = %0.2f)'%ROC_AUC_5,␣
 ↪linewidth= 4)
plt.plot(FPR_6, TPR_6, label= 'NU ROC curve(area = %0.2f)'%ROC_AUC_6,␣
 ↪linewidth= 4)
plt.plot(FPR_7, TPR_7, label= 'ETC ROC curve(area = %0.2f)'%ROC_AUC_7,␣
 ↪linewidth= 4)
plt.plot(FPR_8, TPR_8, label= 'GR ROC curve(area = %0.2f)'%ROC_AUC_8,␣
 ↪linewidth= 4)
plt.plot(FPR_9, TPR_9, label= 'RF ROC curve(area = %0.2f)'%ROC_AUC_9,␣
 ↪linewidth= 4)
plt.plot(FPR_10, TPR_10, label= 'ADA ROC curve(area = %0.2f)'%ROC_AUC_10,␣
 ↪linewidth= 4)
plt.plot([0,1],[0,1], 'k--', linewidth = 4)
plt.xlim([0.0,1.0])
plt.ylim([0.0,1.05])
plt.xlabel('False Positive Rate', fontsize = 18)
plt.ylabel('True Positive Rate', fontsize = 18)
plt.title('ROC for Unknown Data Set', fontsize= 18)
```

```
plt.legend()
plt.show()
```

```
Desicion Tree ROC is  0.8696979517759916
Logistic Regression ROC is  0.8036362457868809
Naive Bayes ROC is  0.7952424163857921
K-Nearest Neighbours is  0.8601827845475758
Linear Discriminant Analysis is  0.7883199377754732
Nu-Support Classification is  0.8013741249675914
Extra Tree Classifier is  0.882991962665284
Gradient Boost is  0.8986129115893181
Random Forest is  0.9048548094373865
AdaBoost is  0.8762768991444129
```