A. Background

A data set containing 33 columns and a target column was provided. With no background information. This is truly an amazing dataset as personal biases cannot be introduced.

B. Objective

Objective is to explore the dataset, perform data preprocessing and build 10 models and decide on which model works best for predicting the target

The platform that was used: Python

C. Data Exploration

The Dataset

Pd.read_csv is a pandas function to read csv files and do operations on it. The head() function is used to get the first n rows. It is useful for quickly testing if your object has the right type of data in it

	V1	V2	V3	V4	V5	V6	V7	V8	V9	V10	...	V25	V26	V27	V28	V29	V30	V31	V32	V33	Class
0	42	50	270900	270944	267	17	44	24220	76	108	...	0.8182	-0.2913	0.5822	1	0	0	0	0	0	1
1	645	651	2538079	2538108	108	10	30	11397	84	123	...	0.7931	-0.1756	0.2984	1	0	0	0	0	0	1
2	829	835	1553913	1553931	71	8	19	7972	99	125	...	0.6667	-0.1228	0.2150	1	0	0	0	0	0	1
3	853	860	369370	369415	176	13	45	18996	99	126	...	0.8444	-0.1568	0.5212	1	0	0	0	0	0	1
4	1289	1306	498078	498335	2409	60	260	246930	37	126	...	0.9338	-0.1992	1.0000	1	0	0	0	0	0	1

5 rows × 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

```
<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

```
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 the values of target variables.

```
1    1268
2     673
Name: Class, dtype: int64
```

Replacing the target value 2 to 0 via hot encoding.

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

	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 data objects which have unique values that are equal to 2.

Making a list of all the data objects whose unique value equal to 2

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

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

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

```
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
```

Note: The above blocks of code automatically identities the columns which need to be changed to objects

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.

```
data.describe(include = 'O')
```

	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

```
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.

```
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:

$$G > \frac{(N-1)}{\sqrt{N}} \sqrt{\frac{(t_{\alpha/(2N),N-2})^2}{N-2+(t_{\alpha/(2N),N-2})^2}}$$

$G=\max|Yi-\bar{Y}|s$

with $\bar{Y}$ 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.

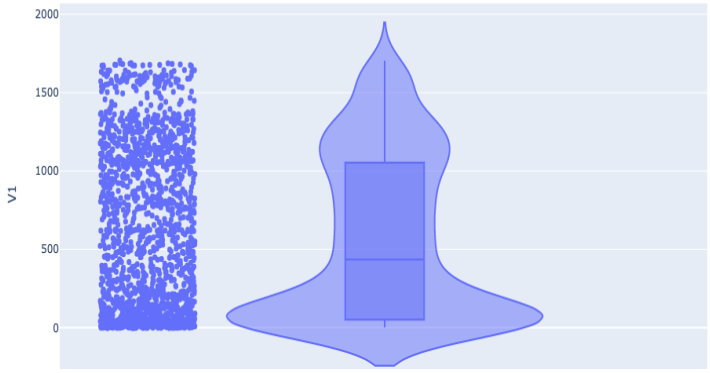
```
#Grubbs Test Function

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 variables and plotting outliers in each variable and verifying if the outliers exist are not.

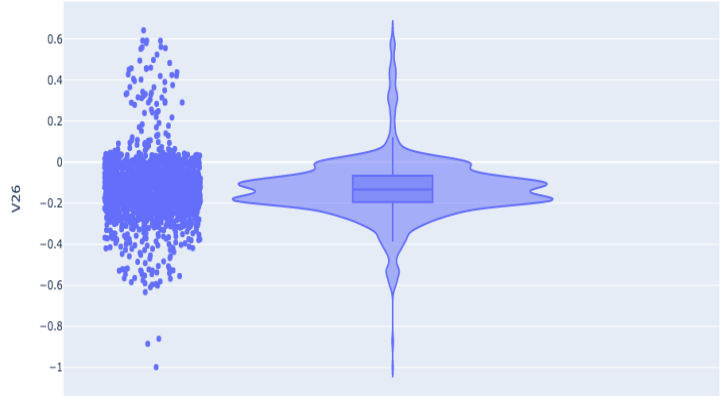
```
data_num_out = list()
for var in data_num:
  fig = px.violin(data, y=var, box=True, points='all')
  fig.show()
  print(grubbs_test(data[var]))
  if out == "Outliers exsist":
    data_num_out.append(str(var))
```

Outlier Detection

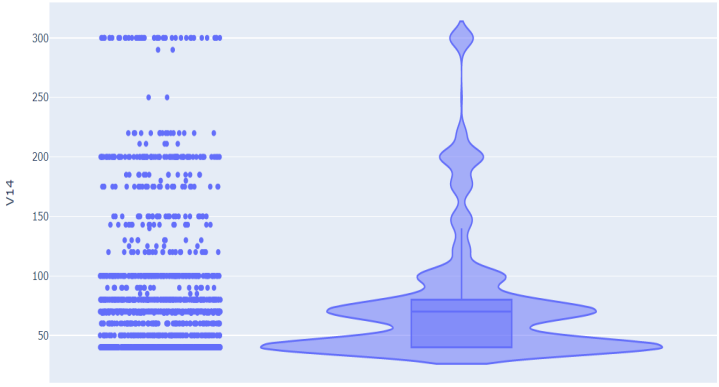
Making a list of numerical variables and plotting outliers in each variable and verifying if the outliers exist are not. Following are 2 of the 33 plots for outlier identification



V1
Grubbs Calculated Value is : 2.18
Grubbs Critical Value is : 3.48
No outliers exist
None



V26
Grubbs Calculated Value is : 5.83
Grubbs Critical Value is : 3.48
Outliers exist
None



V14
Grubbs Calculated Value is : 4.02
Grubbs Critical Value is : 3.48
Outliers exist
None

From the above box plot of V14, it doesn't look like it is having outliers as the data spreads evenly. So we will remove V14 from the list of outliers

```
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.

```
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))
```

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

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

The output of each field is as follows:-

```
V3
q1: 28533.0 q2: 10359325.199999997
Outliers :  [21349, 19815, 11430396, 11741476, 12577343, 12725281, 12917033, 12987661, 21512, 9007, 13302, 19000, 743
0, 15184, 10391495, 10507433, 7851, 11150448, 12438460, 12806495, 6712, 11066410, 11499942, 11569824, 12416454, 2328
8, 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, 745
8, 15196, 10391507, 10507445, 7865, 11150470, 12438491, 12806520, 6724, 11066424, 11499957, 11569844, 12416473, 2331
6, 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, 1889
6, 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, 2156
667, 2230510, 2160349, 2264960, 2225392, 2400588, 2155986, 2236201, 764, 255, 250, 1537, 1543, 1528, 1504, 1541, 153
9, 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, 25
3, 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: 0.190359999999999853
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, 0.4
177, 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, 0.60
21, 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: 2.44591999999999996
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: 0.426299999999999973
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, 0.59
09, 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, -0.88
5, -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
```

Looking at the data after outliers are removed

Now we run a KNN Imputer to replace the 5 nearest neighbor values for NAN and replace with blanks

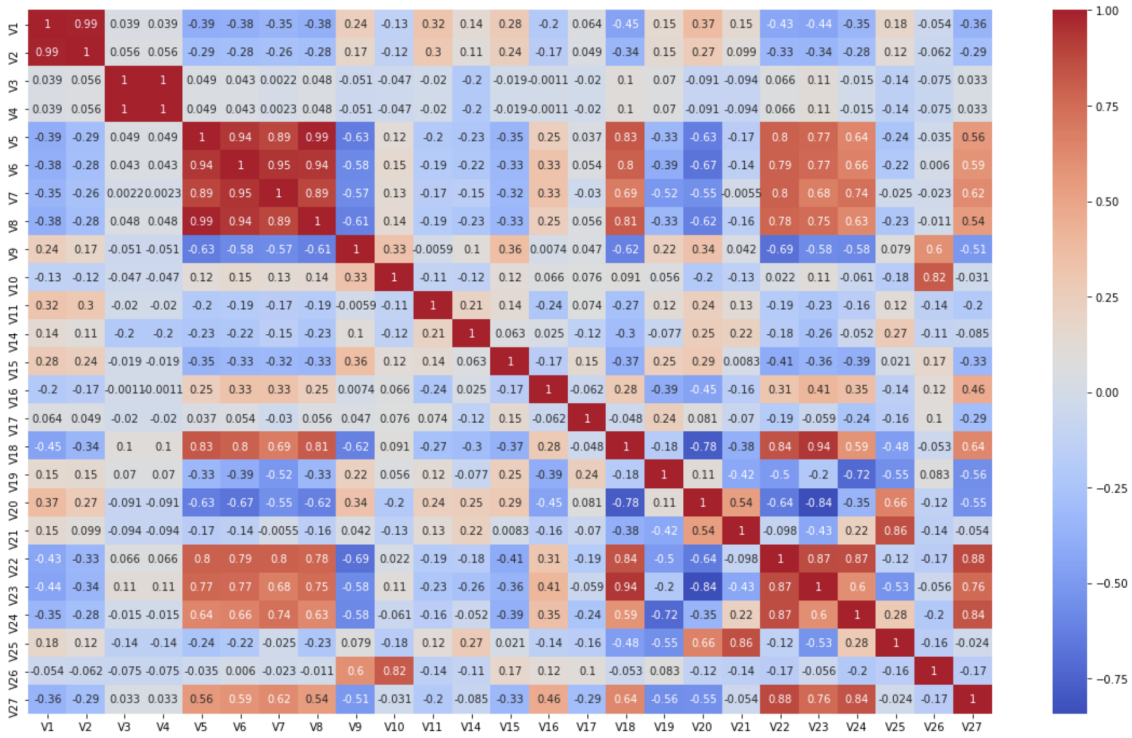
```
from sklearn.impute import KNNImputer
imputer = KNNImputer(n_neighbors=5)
data_imputer = imputer.fit_transform(data)
data_imputer = pd.DataFrame(data_imputer,columns =data_colname)
data = data_imputer
```

Correlation

Verifying again with a heatmap after dropping the variables so there must not be any spaces.

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

The below heatmap gives us an overview about the correlation between the variables. This process is one of the dimension reduction techniques in order to tune the model in a better way. The correlation between the independent variables would create noise to the data.



We proceeded with the correlation threshold of 90% and below is the result of the correlated feature

```
corr_feature = getcorel(X,0.90)
corr_feature
```

```
{'V2', 'V23', 'V4', 'V6', 'V7', 'V8'}
```

	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
6	V7	V6	0.948425
7	V6	V7	0.948425
8	V6	V5	0.939596
9	V5	V6	0.939596
10	V6	V8	0.937805
11	V8	V6	0.937805
12	V23	V18	0.936695
13	V18	V23	0.936695

The curse of dimensionality is such that it will both skew the data and sometimes there is information loss as well. Hence amongst the correlated features it was decided to retain certain variables which give the highest information gain, i.e, very important features for the target variable.

Hence we proceeded with the grouping of variables and in turn the information gain out of the group

The groups are

```
   features1 features2  corr_value
0         V4        V3         1.0
    features1 features2  corr_value
2          V8        V5    0.992703
11         V8        V6    0.937805
   features1 features2  corr_value
4         V1        V2    0.988314
   features1 features2  corr_value
6         V7        V6    0.948425
    features1 features2  corr_value
12        V23       V18    0.936695
```

We find the Feature Importance using RandomForest Classifier. The important features are

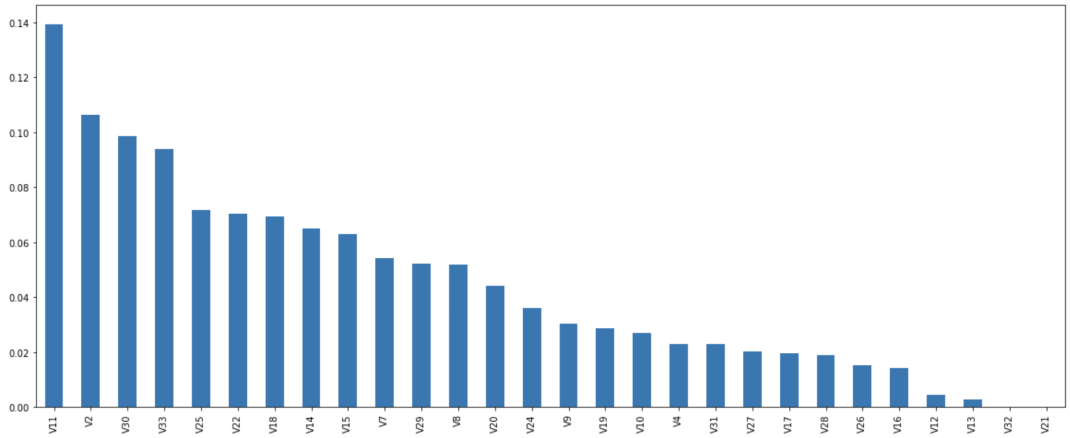
```
[features            V4
 importance    0.502034
 Name: 0, dtype: object,
 features            V8
 importance    0.392611
 Name: 0, dtype: object,
 features            V2
 importance    0.511688
 Name: 1, dtype: object,
 features            V7
 importance    0.503418
 Name: 0, dtype: object,
 features           V18
 importance    0.646545
 Name: 1, dtype: object]
```

By these important features we understand the feature V18 is very important in the group of V18 and V23.
In the group V5,V6 and V8 we understand V8 has the highest importance than other 2 variables. Similarly V7 is of higher importance between V6 and V7 and V4 is more important in the group V3 and V4.

Dropped Variables

Now that we have established the important features, we will go ahead and drop features V1, V3, V5, V6, V23

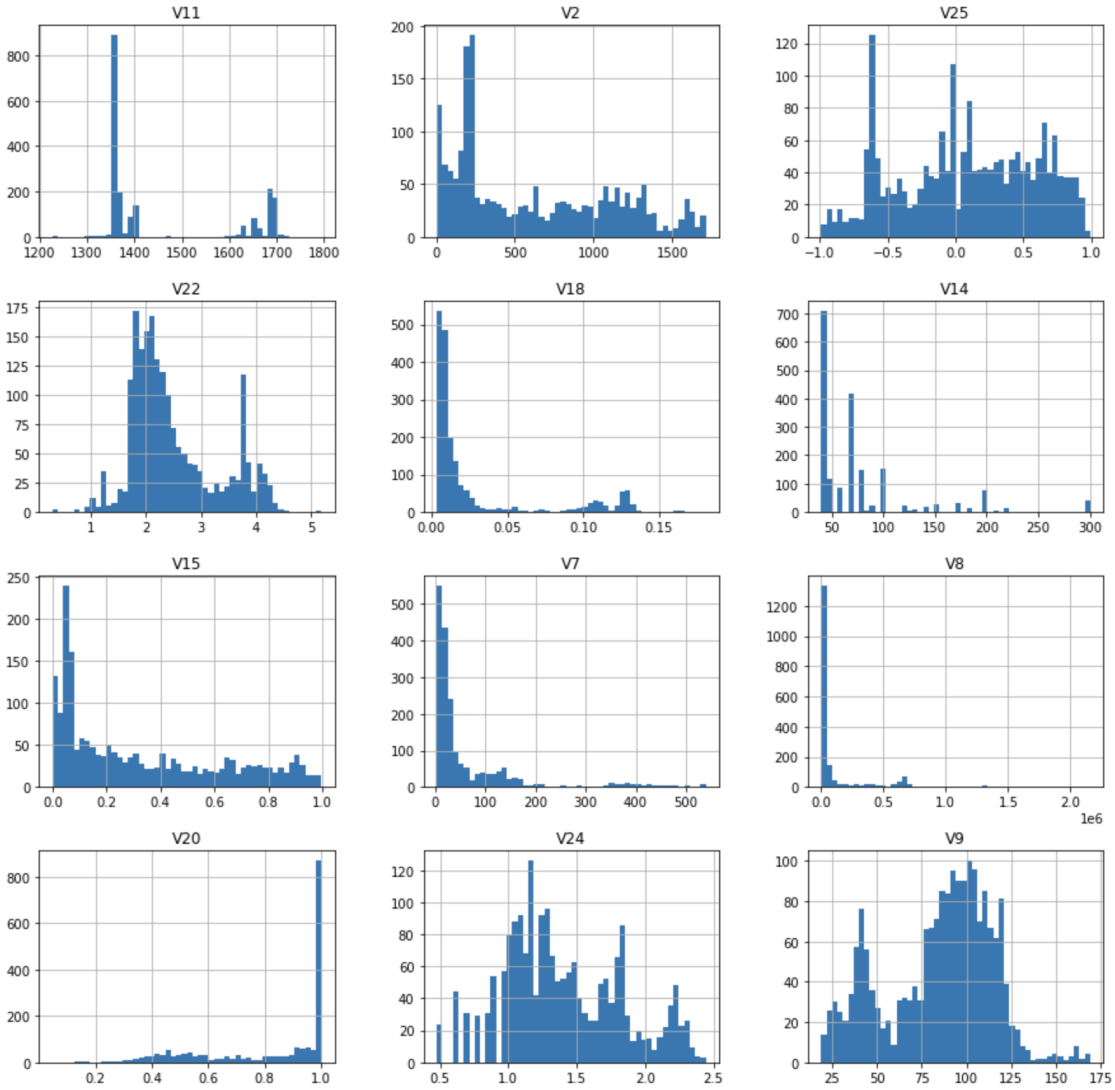
Then based on mutual information classifier, we get the following output



Based on the output and graph we will drop all variables which are less than 0.03, the variables we have dropped are **V31, V17, V16, V10, V4, V19, V26, V21, V13, V12**

```
midf = mi.to_frame('importance')
midf = midf[midf['importance'] >= 0.03]
keep = midf.index.values.tolist()
X = data[keep]
```

We have kept only important variables with an importance higher than 0.03.
Taking importance threshold of 0.01 leads to perfect results of 1.0 accuracy in some models, therefore to avoid overfitting we are taking importance threshold of 0.03
Following is a visual histogram of the of these important variables with a bin value of 50



We normalized the data by using a robust scaler.This Scaler removes the median and scales the data according to the quantile range (defaults to IQR: Interquartile Range). The IQR is the range between the 1st quartile (25th quantile) and the 3rd quartile (75th quantile).Centering and scaling happen independently on each feature by computing the relevant statistics on the samples in the training set. Median and interquartile range are then stored to be used on later data using the transform method.

```
# #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
```

Modeling

We start by splitting the data into test and train sections. We do a random 30% split for test and the seed is set at 100 so that we get consistent splits

The data which is wrangled then was split into X_train, X_test, y_train, y_test with 30% of the data into test and 70% for training. The code random state 10 was used inorder to keep the 100^{th} combination of splitting throughout the modeling. Sklearn was used to make all the models

D. Finding The Optimal Method

1. Naive Bayes

It is a classification technique based on Bayes' theorem with an assumption of independence between predictors. In simple terms, a Naive Bayes classifier assumes that the presence of a particular feature in a class is unrelated to the presence of any other feature.

- Since the random state is mentioned as 10 the same test and train data is used to do naïve bayes method and the score of the model is 0.94078
- The 10-fold cross validation error was
  [0.94117647 0.91911765 0.94117647 0.95588235 0.94852941 0.91911765
  0.94852941 0.94117647 0.97777778 0.88148148]
- The mean is 0.93739, test error is 0.06261 and standard deviation is 0.02461
- The confusion matrix accuracy is 0.94078

```
              precision    recall  f1-score   support

           0       0.82      1.00      0.90       203
           1       1.00      0.88      0.94       380

    accuracy                           0.92       583
   macro avg       0.91      0.94      0.92       583
weighted avg       0.94      0.92      0.92       583

confusion matrix
[[203   0]
 [ 45 335]]

balanced accuracy score is  0.9407894736842105
cohen kappa score is  0.8383001016980492

cross validation
Scores: [0.94117647 0.91911765 0.94117647 0.95588235 0.94852941 0.91911765
 0.94852941 0.94117647 0.97777778 0.88148148]

Mean: 0.9373965141612199
Standard Deviation: 0.024612688925041586
```

2. K-Nearest Neighbor

It can be used for both classification and regression problems. However, it is more widely used in classification problems in the industry. K nearest neighbors is a simple algorithm that stores all available cases and classifies new cases by a majority vote of its k neighbors. The case being assigned to the class is most common amongst its K nearest neighbors measured by a distance function.

- KNN has a mean accuracy score of 0.92719 and the distance used is by default the minkowski distance, and the number of n's that is taken in the model is 7
- The 10-fold cross validation error was
  [0.94852941 0.96323529 0.94852941 0.92647059 0.90441176 0.88970588
  0.94117647 0.94117647 0.94074074 0.97037037]
  The mean is 0.93743, test error is 0.06257 and standard deviation is 0.02350
- The confusion matrix accuracy is 0.92719

```
              precision    recall  f1-score   support

           0       0.86      0.94      0.90       203
           1       0.96      0.92      0.94       380

    accuracy                           0.92       583
   macro avg       0.91      0.93      0.92       583
weighted avg       0.93      0.92      0.93       583

confusion matrix
[[190  13]
 [ 31 349]]

balanced accuracy is  0.927190821882292
cohen kappa score is  0.8370949919347669

cross validation
Scores: [0.94852941 0.96323529 0.94852941 0.92647059 0.90441176 0.88970588
 0.94117647 0.94117647 0.94074074 0.97037037]

Mean: 0.9374346405228758
Standard Deviation: 0.023503854861876866
```

3. Decision Tree

It is a type of supervised learning algorithm that is mostly used for classification problems. Surprisingly, it works for both categorical and continuous dependent variables. In this algorithm, we split the population into two or more homogeneous sets. This is done based on most significant attributes/ independent variables to make as distinct groups as possible.

- Decision tree confusion matrix accuracy score is 0.97617
- The 10 fold cross validation error was
  [0.95588235 0.96323529 0.97794118 0.98529412 0.94852941 0.97058824
  0.95588235 0.99264706 0.99259259 0.95555556]
  The mean is 0.96981, test error is 0.03019 and standard deviation is 0.01560

```
              precision    recall  f1-score   support

           0       0.98      0.97      0.97       203
           1       0.98      0.99      0.98       380

    accuracy                           0.98       583
   macro avg       0.98      0.98      0.98       583
weighted avg       0.98      0.98      0.98       583

confusion matrix
[[196   7]
 [  5 375]]

balanced accuracy is  0.9761796733212341
cohen kappa score is  0.9545495887634318

cross validation
Scores: [0.95588235 0.96323529 0.97794118 0.98529412 0.94852941 0.97058824
 0.95588235 0.99264706 0.99259259 0.95555556]

Mean: 0.9698148148148148
Standard Deviation: 0.015602228091178023
```

4. Linear Discriminant analysis

A classifier with a linear decision boundary, generated by fitting class conditional densities to the data and using Bayes' rule. The model fits a Gaussian density to each class, assuming that all classes share the same covariance matrix.
The fitted model can also be used to reduce the dimensionality of the input by projecting it to the most discriminative directions

- The confusion matrix accuracy is 0.94078
- The 10 fold cross validation error was
  [0.94117647 0.91911765 0.94117647 0.95588235 0.95588235 0.92647059
  0.94852941 0.94852941 0.98518519 0.88148148]
  The mean is 0.94034, test error is 0.05966 and standard deviation is 0.02595

```
              precision    recall  f1-score   support

           0       0.82      1.00      0.90       203
           1       1.00      0.88      0.94       380

    accuracy                           0.92       583
   macro avg       0.91      0.94      0.92       583
weighted avg       0.94      0.92      0.92       583

confusion matrix
[[203   0]
 [ 45 335]]

Balanced accuracy is 0.9407894736842105
cohen kappa score is 0.8383001016980492

cross validation
Scores: [0.94117647 0.91911765 0.94117647 0.95588235 0.95588235 0.92647059
 0.94852941 0.94852941 0.98518519 0.88148148]

Mean: 0.940343137254902
Standard Deviation: 0.025955397032627798
```

5. Nu-SVC

In this algorithm, we plot each data item as a point in n-dimensional space (where n is number of features you have) with the value of each feature being the value of a particular coordinate. For example, if we only had two features like Height and Hair length of an individual, we'd first plot these two variables in two dimensional space where each point has two coordinates (these coordinates are known as Support Vectors)

- The confusion matrix accuracy is 0.93208
- The 10 fold cross validation error was
  [0.94117647 0.91911765 0.94117647 0.94852941 0.94117647 0.91176471
  0.94852941 0.94117647 0.97037037 0.87407407]
  The mean is 0.93371, test error is 0.06629 and standard deviation is 0.02501

```
              precision    recall  f1-score   support

           0       0.81      0.99      0.89       203
           1       0.99      0.88      0.93       380

    accuracy                           0.92       583
   macro avg       0.90      0.93      0.91       583
weighted avg       0.93      0.92      0.92       583

confusion matrix
[[200   3]
 [ 46 334]]

Balanced accuracy is 0.93208452164895
cohen kappa score is 0.8235417657559716

Scores: [0.94117647 0.91911765 0.94117647 0.94852941 0.94117647 0.91176471
 0.94852941 0.94117647 0.97037037 0.87407407]

Mean: 0.9337091503267972
Standard Deviation: 0.025014359416282144
```

6. Extra Trees Classifier

ExtraTreesClassifier is an ensemble learning method fundamentally based on decision trees. Similar to RandomForest, ExtraTreesClassifier randomizes certain decisions and subsets of data to minimize over-learning from the data and overfitting. Let's look at some ensemble methods ordered from high to low variance, ending in ExtraTreesClassifier.

- The confusion matrix accuracy is 0.98454
- The 10 fold cross validation error was
  [0.96323529 0.98529412 0.99264706 0.98529412 0.98529412 0.98529412
  1. 0.97794118 0.99259259 0.96296296]
  The mean is 0.98305, test error is 0.01695 and standard deviation is 0.01145

```
              precision    recall  f1-score   support

           0       0.96      0.99      0.98       203
           1       0.99      0.98      0.99       380

    accuracy                           0.98       583
   macro avg       0.98      0.98      0.98       583
weighted avg       0.98      0.98      0.98       583

confusion matrix
[[201   2]
 [  8 372]]

Balanced accuracy is 0.9845475758361422
cohen kappa score is 0.9624699051125902

Scores: [0.96323529 0.98529412 0.99264706 0.98529412 0.98529412 0.98529412
 1.         0.97794118 0.99259259 0.96296296]

Mean: 0.9830555555555556
Standard Deviation: 0.011457847803361672
```

7. Gradient Boosting

GBM is a boosting algorithm used when we deal with plenty of data to make a prediction with high prediction power. Boosting is actually an ensemble of learning algorithms which combines the prediction of several base estimators in order to improve robustness over a single estimator. It combines multiple weak or average predictors to build a strong predictor.

- The confusion matrix accuracy is 0.98191
- The 10 fold cross validation error was
  [0.97794118 0.96323529 0.99264706 0.99264706 0.97058824 0.97794118
  0.98529412 0.98529412 1. 0.97777778]
  The mean is 0.98233, test error is 0.01767 and standard deviation is 0.01051

```
              precision    recall  f1-score   support

           0       0.95      0.99      0.97       203
           1       0.99      0.97      0.98       380

    accuracy                           0.98       583
   macro avg       0.97      0.98      0.98       583
weighted avg       0.98      0.98      0.98       583

confusion matrix
[[201   2]
 [ 10 370]]
Balanced accuracy is 0.9819159968887736
cohen kappa score is 0.9550662830130511

Scores: [0.97794118 0.96323529 0.99264706 0.99264706 0.97058824 0.97794118
 0.98529412 0.98529412 1.         0.97777778]

Mean: 0.9823366013071896
Standard Deviation: 0.010509076833204609
```

8. Logistic Regression

It is used to estimate discrete values based on a given set of independent variables. It predicts the probability of occurrence of an event by fitting data to a logit function. Hence, it is also known as logit regression. Since it predicts the probability, its output value lies between 0 and 1 (as expected).

- The confusion matrix accuracy is 0.95576
- The 10 fold cross validation error was
  [0.94852941 0.94852941 0.95588235 0.97794118 0.96323529 0.94117647
  0.95588235 0.94852941 0.97777778 0.95555556]
  The mean is 0.95730, test error is 0.0427 and standard deviation is 0.01174

```
              precision    recall  f1-score   support

           0       0.88      0.99      0.93       203
           1       0.99      0.93      0.96       380

    accuracy                           0.95       583
   macro avg       0.93      0.96      0.94       583
weighted avg       0.95      0.95      0.95       583

confusion matrix
[[200   3]
 [ 28 352]]
Balanced accuracy is 0.9557687321752657
cohen kappa score is 0.8861220503449797

Scores: [0.94852941 0.94852941 0.95588235 0.97794118 0.96323529 0.94117647
 0.95588235 0.94852941 0.97777778 0.95555556]

Mean: 0.9573039215686274
Standard Deviation: 0.011740636689463653
```

9. Random Forest Classifier

In Random Forest, we've collection of decision trees (so known as "Forest"). To classify a new object based on attributes, each tree gives a classification and we say the tree "votes" for that class. The forest chooses the classification having the most votes (over all the trees in the forest).

- The confusion matrix accuracy is 0.98060
- The 10 fold cross validation error was
  [0.75735294 0.71323529 0.77941176 0.74264706 0.74264706 0.70588235
  0.76470588 0.79411765 0.73333333 0.74814815]
  The mean is 0.74815, test error is 0.25185 and standard deviation is 0.02595

```
              precision    recall  f1-score   support

           0       0.95      0.99      0.97       203
           1       0.99      0.97      0.98       380

    accuracy                           0.98       583
   macro avg       0.97      0.98      0.98       583
weighted avg       0.98      0.98      0.98       583

confusion matrix
[[201  11]
 [  2 369]]
Balanced accuracy is 0.9806002074150895
cohen kappa score is 0.9513770826249575

Scores: [0.75735294 0.71323529 0.77941176 0.74264706 0.74264706 0.70588235
 0.76470588 0.79411765 0.73333333 0.74814815]

Mean: 0.7481481481481482
Standard Deviation: 0.025951498711256165
```

10. ADABoost Classifier

An AdaBoost classifier is a meta-estimator that begins by fitting a classifier on the original dataset and then fits additional copies of the classifier on the same dataset but where the weights of incorrectly classified instances are adjusted such that subsequent classifiers focus more on difficult cases.

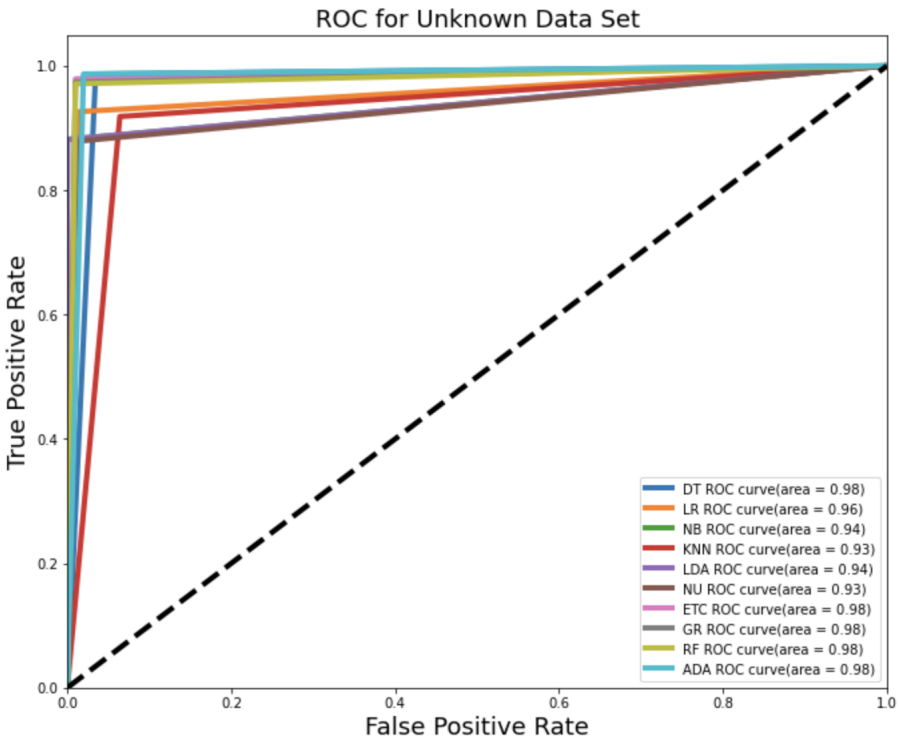
- The confusion matrix accuracy is 0.98356
- The 10 fold cross validation error was
  [0.97058824 0.97794118 0.98529412 0.99264706 0.98529412 0.99264706
  0.98529412 0.97058824 1. 0.94814815]
  The mean is 0.98084, test error is 0.01916 and standard deviation is 0.01412

```
              precision    recall  f1-score   support

           0       0.98      0.98      0.98       203
           1       0.99      0.99      0.99       380

    accuracy                           0.98       583
   macro avg       0.98      0.98      0.98       583
weighted avg       0.98      0.98      0.98       583

confusion matrix
[[199   5]
 [  4 375]]
Balanced accuracy is 0.9835688358828105
cohen kappa score is 0.9660293803453388

Scores: [0.97058824 0.97794118 0.98529412 0.99264706 0.98529412 0.99264706
 0.98529412 0.97058824 1.         0.94814815]

Mean: 0.9808442265795208
Standard Deviation: 0.014116635236677319
```

Optimal Method

	Model	Score	CV Mean	CV Std
0	Extra Tree Classifier	0.984548	0.983056	0.011458
1	AdaBoost	0.983569	0.980844	0.014117
2	Gradiant Boost	0.981916	0.982337	0.010509
3	Random Forest	0.980600	0.748148	0.025951
4	Decision Tree	0.976180	0.969815	0.015602
5	LogisticRegression	0.955769	0.957304	0.011741
6	Naive Bayes	0.940789	0.937397	0.024613
7	Linear Discriminant Analysis	0.940789	0.940343	0.025955
8	Nu-Support Vector Classification	0.932085	0.933709	0.025014
9	K-Nearest Neighbours	0.927191	0.937435	0.023504

```
Desicion Tree ROC is  0.9761796733212342
Logistic Regression ROC is  0.9557687321752658
Naive Bayes ROC is  0.9407894736842105
K-Nearest Neighbours is  0.927190821882292
Linear Discriminant Analysis is  0.9407894736842105
Nu-Support Classification is  0.93208452164895
Extra Tree Classifier is  0.984547575836142
Gradient Boost is  0.9819159968887737
Random Forest is  0.9806002074150895
AdaBoost is  0.9835688358828105
```



Conclusion

Based on the balanced accuracy score

Balanced Accuracy = (Sensitivity+specificity) / 2.

Extra trees classifier has the highest score. We can choose that as our best model to classify target variable

Taking importance threshold of 0.01 leads to perfect results of 1.0 accuracy in some models as shown in the table below, therefore to avoid overfitting we are taking importance threshold of 0.03 and recommending

	Model	Score	CV Mean	CV Std
0	Decision Tree	1.000000	1.000000	0.000000
1	LogisticRegression	1.000000	1.000000	0.000000
2	Naive Bayes	1.000000	0.998529	0.002941
3	Extra Tree Classifier	1.000000	0.997794	0.003370
4	Gradiant Boost	1.000000	1.000000	0.000000
5	AdaBoost	1.000000	1.000000	0.000000
6	Random Forest	0.995074	0.753312	0.022140
7	Nu-Support Vector Classification	0.988663	0.986019	0.008967
8	K-Nearest Neighbours	0.880224	0.899123	0.017988
9	Linear Discriminant Analysis	0.700389	0.772435	0.034528