Assignment 2

Arina Sheredekina
Miguel Gonzalez Gallardo
Leidy Herrera Torres

The project used the heart disease prediction numeric dataset where linear SVM, different kernel SVM,Naive Bayes and KNN classifiers were implemented.

Preprocessing

- Install all the packages necessary for the model

```
[ ]  import pandas as pd
     import numpy as np
     import matplotlib.pyplot as plt
     import seaborn as sns
     from sklearn.model_selection import train_test_split
     from sklearn.metrics import accuracy_score
     from sklearn.metrics import confusion_matrix, classification_report
     from sklearn.multiclass import OneVsRestClassifier
     from sklearn.multiclass import OneVsOneClassifier
     from sklearn.svm import SVC
     from sklearn.neighbors import KNeighborsClassifier
     from sklearn.naive_bayes import GaussianNB
     from sklearn.linear_model import LogisticRegression
     from sklearn import metrics
     from imblearn.over_sampling import SMOTE
     from imblearn.under_sampling import NearMiss
```

- Import the CSV file using read_csv function

```
df = pd.read_csv('/content/framingham.csv')
```

- Use head to have a block of information and file content

```
df.head()
```

	male	age	education	currentSmoker	cigsPerDay	BPMeds	prevalentStroke	prevalentHyp	diabetes	totChol	sysBP	diaBP	BMI	heartRate	glucose	TenYearCHD
0	1	39	4.0	0	0.0	0.0	0	0	0	195.0	106.0	70.0	26.97	80.0	77.0	0
1	0	46	2.0	0	0.0	0.0	0	0	0	250.0	121.0	81.0	28.73	95.0	76.0	0
2	1	48	1.0	1	20.0	0.0	0	0	0	245.0	127.5	80.0	25.34	75.0	70.0	0
3	0	61	3.0	1	30.0	0.0	0	1	0	225.0	150.0	95.0	28.58	65.0	103.0	1
4	0	46	3.0	1	23.0	0.0	0	0	0	285.0	130.0	84.0	23.10	85.0	85.0	0

- Use the Describe method to obtain summary information from columns

```
df.describe()
```

	male	age	education	currentSmoker	cigsPerDay	BPMeds	prevalentStroke	prevalentHyp	diabetes	totChol	sysBP	diaBP	BMI	heartRate	glucose	TenYearCHD
count	4238.000000	4238.000000	4133.000000	4238.000000	4209.000000	4185.000000	4238.000000	4238.000000	4238.000000	4188.000000	4238.000000	4238.000000	4219.000000	4237.000000	3850.000000	4238.000000
mean	0.429212	49.584946	1.978950	0.494101	9.003089	0.029630	0.005899	0.310524	0.025720	236.721585	132.352407	82.893464	25.802008	75.878924	81.966753	0.151958
std	0.495022	8.572160	1.019791	0.500024	11.920094	0.169584	0.076587	0.462763	0.158316	44.590334	22.038097	11.910850	4.080111	12.026596	23.959998	0.359023
min	0.000000	32.000000	1.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	107.000000	83.500000	48.000000	15.540000	44.000000	40.000000	0.000000
25%	0.000000	42.000000	1.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	206.000000	117.000000	75.000000	23.070000	68.000000	71.000000	0.000000
50%	0.000000	49.000000	2.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	234.000000	128.000000	82.000000	25.400000	75.000000	78.000000	0.000000
75%	1.000000	56.000000	3.000000	1.000000	20.000000	0.000000	0.000000	1.000000	0.000000	263.000000	144.000000	89.875000	28.040000	83.000000	87.000000	0.000000
max	1.000000	70.000000	4.000000	1.000000	70.000000	1.000000	1.000000	1.000000	1.000000	696.000000	295.000000	142.500000	56.800000	143.000000	394.000000	1.000000

- Use Shape to know how many rows and columns are in the dataset (4238 rows, 16 columns)

```
df.shape
```

```
(4238, 16)
```

- Columns attribute to identify the index of column names(16)

```
df.columns
```

```
Index(['male', 'age', 'education', 'currentSmoker', 'cigsPerDay', 'BPMeds',
       'prevalentStroke', 'prevalentHyp', 'diabetes', 'totChol', 'sysBP',
       'diaBP', 'BMI', 'heartRate', 'glucose', 'TenYearCHD'],
      dtype='object')
```

- Info method to obtain an overview of the dataset, in this situation all the features are numeric and the memory usage is 529.9 KB

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4238 entries, 0 to 4237
Data columns (total 16 columns):
 #   Column           Non-Null Count  Dtype
---  ------           --------------  -----
 0   male             4238 non-null   int64
 1   age              4238 non-null   int64
 2   education        4133 non-null   float64
 3   currentSmoker    4238 non-null   int64
 4   cigsPerDay       4209 non-null   float64
 5   BPMeds           4185 non-null   float64
 6   prevalentStroke  4238 non-null   int64
 7   prevalentHyp     4238 non-null   int64
 8   diabetes         4238 non-null   int64
 9   totChol          4188 non-null   float64
 10  sysBP            4238 non-null   float64
 11  diaBP            4238 non-null   float64
 12  BMI              4219 non-null   float64
 13  heartRate        4237 non-null   float64
 14  glucose          3850 non-null   float64
 15  TenYearCHD       4238 non-null   int64
dtypes: float64(9), int64(7)
memory usage: 529.9 KB
```

- The IsNull method to know if the dataset has missing values. The data has null (missing values) for education, cigsPerDay,BPMeds,totChol,BMI,heartRate, glucose

```
df.isnull().sum()
```

	0
male	0
age	0
education	105
currentSmoker	0
cigsPerDay	29
BPMeds	53
prevalentStroke	0
prevalentHyp	0
diabetes	0
totChol	50
sysBP	0
diaBP	0
BMI	19
heartRate	1
glucose	388
TenYearCHD	0

dtype: int64

- Duplicated method to identify duplicate rows.In this case dataset is clean from them

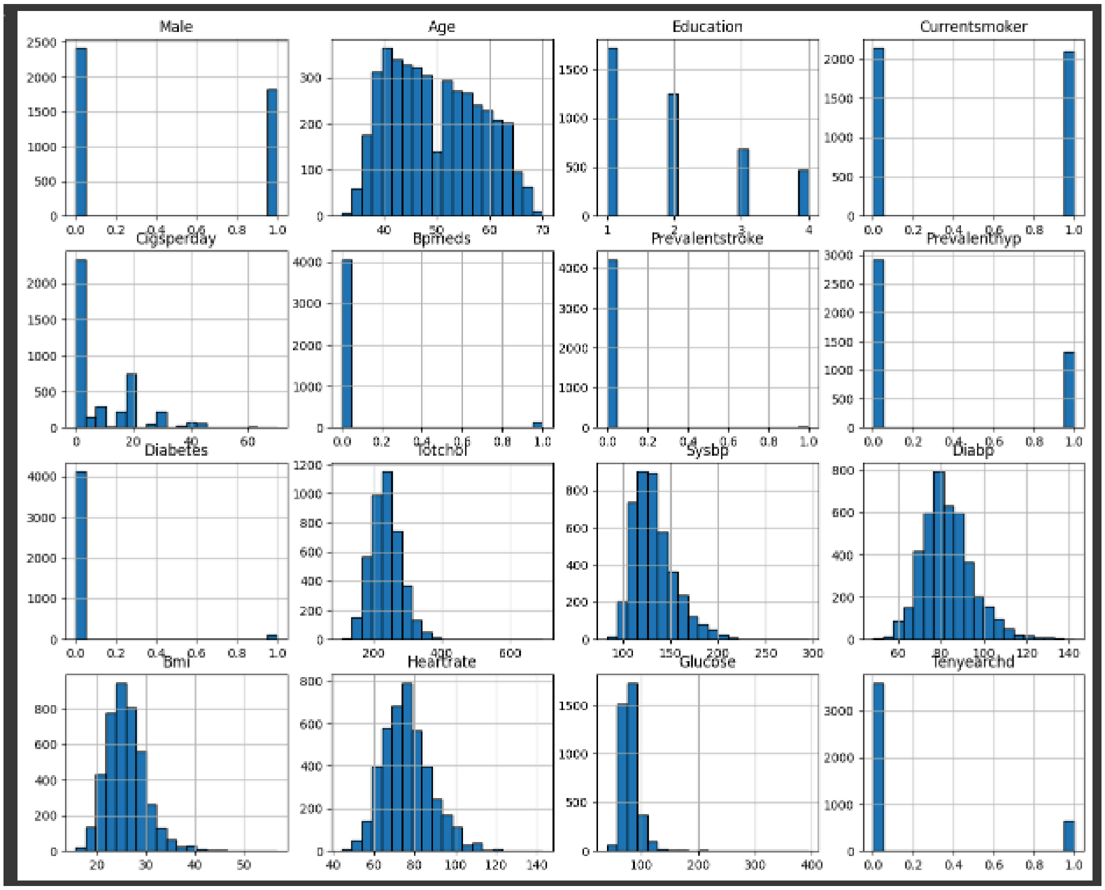
```
df.duplicated().sum()
```

```
0
```

- Visualize the distribution of the data in each column.

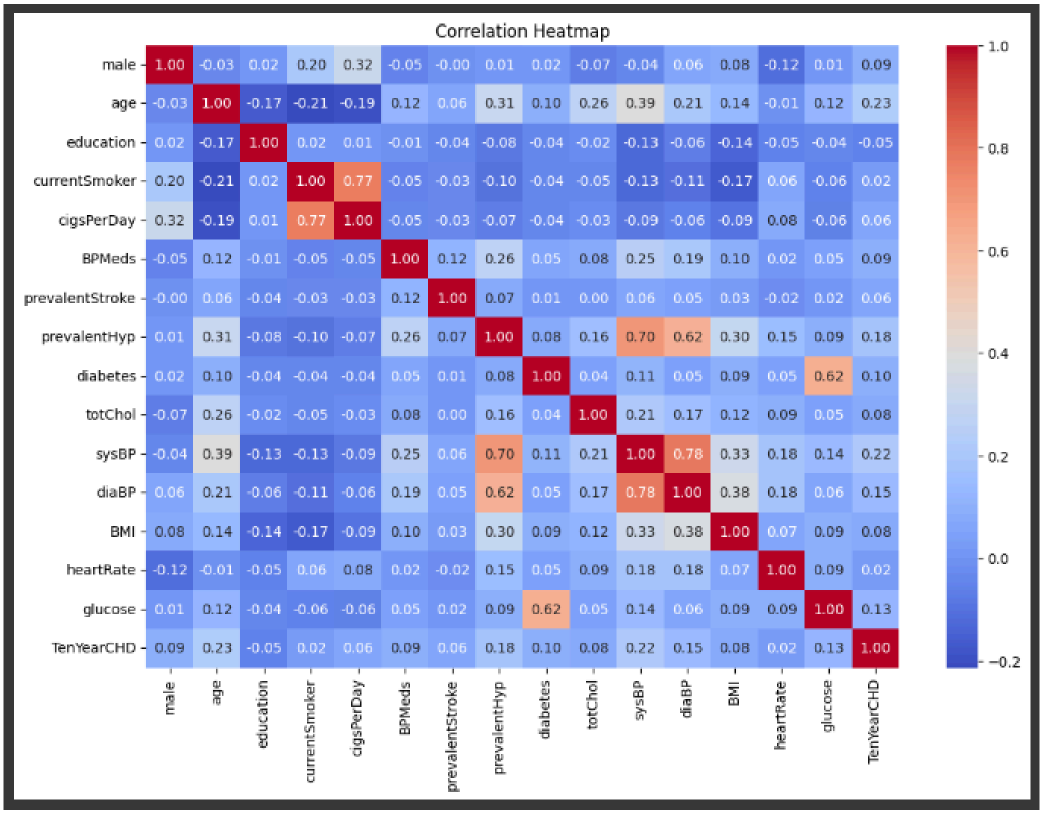
```
plt.figure(figsize=(15, 12))

for i, column in enumerate(df.columns[:]):
    plt.subplot(4, 4, i + 1)  # Create subplots
    df[column].hist(bins=20, edgecolor='black')  # Plot histogram
    plt.title(column.capitalize())  # Set title
```



- The correlation heatmap was generated for all numeric columns in the DateFrame.

```
numeric_df = df.select_dtypes(include=[np.number])
plt.figure(figsize=(12, 8))
sns.heatmap(numeric_df.corr(), annot=True, fmt='.2f',
cmap='coolwarm')
plt.title('Correlation Heatmap')
plt.show()
```



Data Cleaning

- Data cleaning was used. It was verified that none of the columns with null values exceed the 10%. The null values were replaced with the media ,except for the features-education and BPMeds which are categorical, so they were replaced with the mode.

```
missing_perc = df.isnull().mean()*100
missing_perc
df_copy = df.copy()
df_copy['cigsPerDay'] =
df_copy['cigsPerDay'].fillna(df_copy['cigsPerDay'].median())
df_copy['totChol'] =
df_copy['totChol'].fillna(df_copy['totChol'].median())
df_copy['BMI'] = df_copy['BMI'].fillna(df_copy['BMI'].median())
df_copy['heartRate'] =
df_copy['heartRate'].fillna(df_copy['heartRate'].median())
df_copy['glucose'] =
df_copy['glucose'].fillna(df_copy['glucose'].median())
```

- Dropped th columns that are categorical to increase the accuracy of the models

```
df_copy = df_copy.drop(['male', 'currentSmoker', 'BPMeds',
'prevalentStroke', 'prevalentHyp', 'diabetes', 'education'], axis=1)
```

- The IsNull method was used to make sure the dataset doesn't have any missing values left. The result is ok, the data doesn't have null.

```
df_copy.isnull().sum()
```

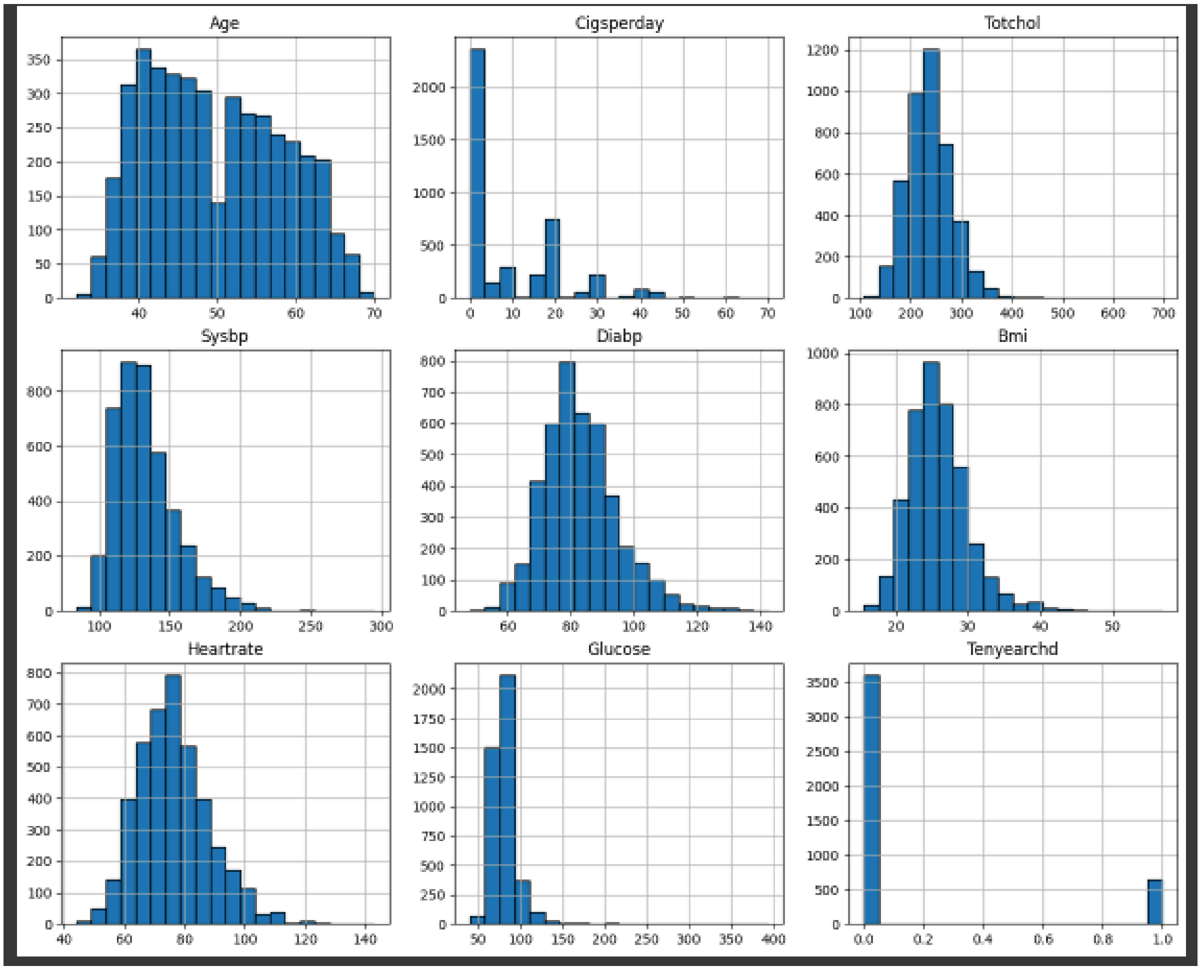
	0
age	0
cigsPerDay	0
totChol	0
sysBP	0
diaBP	0
BMI	0
heartRate	0
glucose	0
TenYearCHD	0

dtype: int64

- Visualize the distribution of the data in each column after cleaning data.

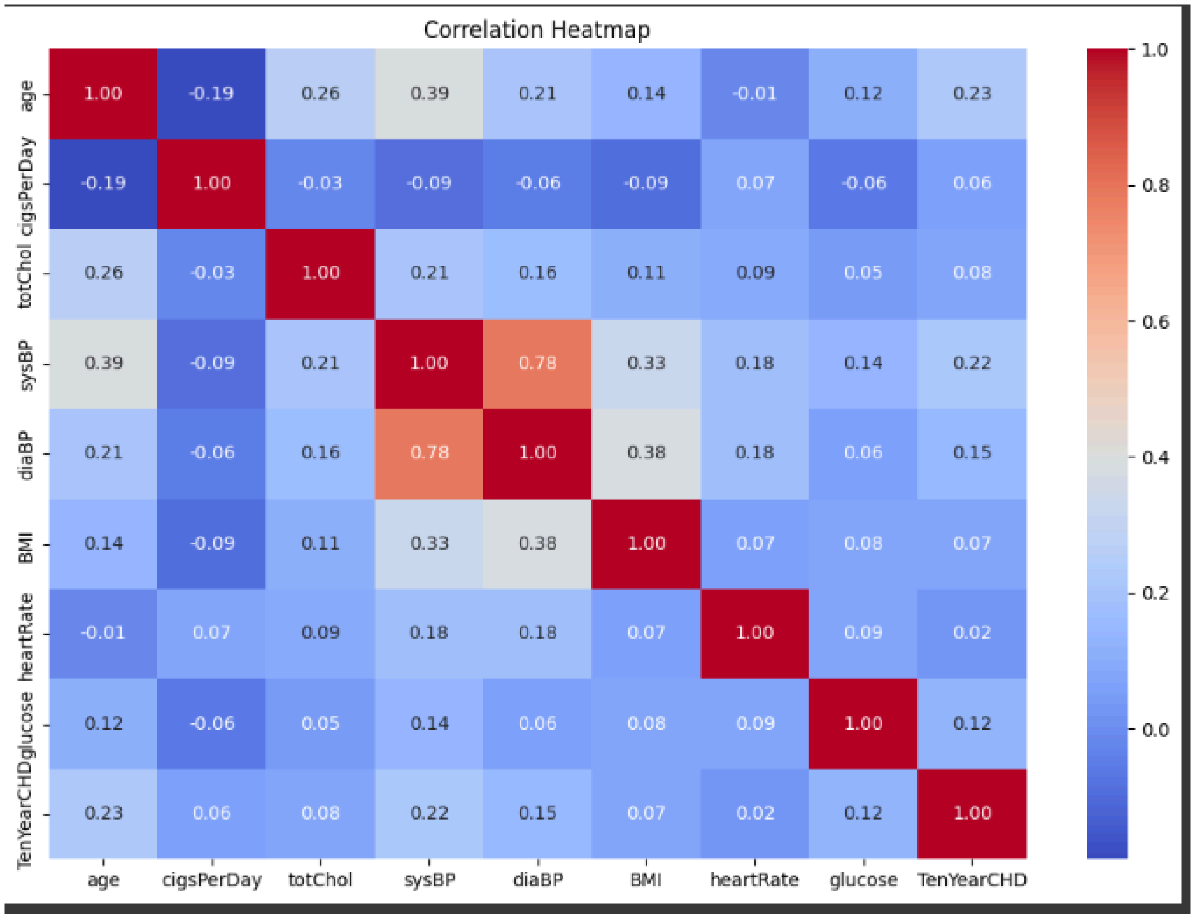
```
plt.figure(figsize=(15, 12))

for i, column in enumerate(df_copy.columns[:]):
    plt.subplot(3, 3, i + 1)  # Create subplots
    df_copy[column].hist(bins=20, edgecolor='black')  # Plot
histogram
    plt.title(column.capitalize())  # Set title
```



- The correlation heatmap was generated for all numeric columns in the DateFrame after cleaning

```
numeric_df = df_copy.select_dtypes(include=[np.number])
plt.figure(figsize=(12, 8))
sns.heatmap(numeric_df.corr(), annot=True, fmt='.2f',
cmap='coolwarm')
plt.title('Correlation Heatmap')
plt.show()
```



Undersampling and OverSampling

- After the result of target values we can infer that values are imbalance,therefore we have to use oversampling/undersampling techniques

```
df['TenYearCHD'].value_counts()
```

	count
TenYearCHD	
0	3594
1	644

dtype: int64

- Drop the target variable and assign remained columns to x; extracts the target variable and assigns it to y `then` print the shape (number of rows and columns) of x and y

```
x = df_copy.drop(['TenYearCHD'], axis=1).values
y = df_copy['TenYearCHD']
x.shape
y.shape
```

```
x.shape
```

```
(4238, 8)
```

```
y.shape
```

```
(4238,)
```

- The data was split into training and testing set and printed for the training features(x_train), the testing features(x_test),the training labels(y_train), the testing labels(y_test)

```
x_train, x_test, y_train, y_test = train_test_split(x, y,
test_size=0.2, random_state=42, stratify=y)
print("The shape of the x train dataset is: ",  x_train.shape)
print("The shape of the x test dataset is: ",  x_test.shape)
print("The shape of the y train dataset is: ",  y_train.shape)
print("The shape of the y test dataset is: ",  y_test.shape)
```

```
The shape of the x train dataset is:  (3390, 8)
The shape of the x test dataset is:  (848, 8)
The shape of the y train dataset is:  (3390,)
The shape of the y test dataset is:  (848,)
```

- The Logistic Regression classifier was used to model the data.

```
lr = LogisticRegression()
```

- Trains (fits) the logistic regression model using the training data

```
lr.fit(x_train, y_train.ravel())
```

- Logistic regression model to make predictions on the test data

```
LogisticRegression
LogisticRegression()
```

- Classification report showing metrics was printed

```
predictions = lr.predict(x_test)
print(classification_report(y_test, predictions))
```

```
              precision    recall  f1-score   support

           0       0.85      0.99      0.92       719
           1       0.50      0.06      0.11       129

    accuracy                           0.85       848
   macro avg       0.68      0.53      0.51       848
weighted avg       0.80      0.85      0.79       848
```

- Calculates and returns the accuracy of the predictions were made

```
metrics.accuracy_score(y_test,predictions)
```

```
0.847877358490566
```

- Confusion matrix was printed

```
print(confusion_matrix(y_test,predictions))
```

```
[[711   8]
 [121   8]]
```

- Oversampling technique was used and prints the count of positive cases in the training dataset before oversampling and prints the count of negative cases in the training dataset before oversampling.

```
print("Before Over Sampling counts of label '1' is: 
{}".format(sum(y_train==1)))
print("Before Over Sampling counts of label '0' is: {} 
\n".format(sum(y_train==0)))
```

```
Before Over Sampling counts of label '1' is: 515
Before Over Sampling counts of label '0' is: 2875
```

- SMOTE was created to handle class imbalance

```
sm = SMOTE(random_state = 2)
```

- Resamples the training data to balance the classes

```
x_train_res, y_train_res = sm.fit_resample(x_train, y_train.ravel())
```

- Print the results after oversampling

```
print("After Over Sampling the shape of x train is : 
{}".format(x_train_res.shape))
print("After Over Sampling the shape of y train is : {} 
\n".format(y_train_res.shape))
```

```
print("After Over Sampling counts of label '1' is : {}".format(sum(y_train_res==1)))
print("After Over Sampling counts of label '0' is : {} \n".format(sum(y_train_res==0)))
```

```
After Over Sampling the shape of x train is : (5750, 8)
After Over Sampling the shape of y train is : (5750,)

After Over Sampling counts of label '1' is : 2875
After Over Sampling counts of label '0' is : 2875
```

- A new instance of the Logistic Regression model was created

```
lr1 = LogisticRegression()
```

- Trains the logistic regression model using the oversampled training data

```
lr1.fit(x_train_res, y_train_res.ravel())
```

```
  n_iter_i = _check_optimize_res
LogisticRegression
LogisticRegression()
```

- The trained model to make predictions on the test data and prints the classification report after training with SMOTE-oversampled data.

```
predictions1 = lr1.predict(x_test)
print(classification_report(y_test, predictions1))
```

	precision	recall	f1-score	support
0	0.90	0.66	0.77	719
1	0.24	0.60	0.35	129
accuracy			0.66	848
macro avg	0.57	0.63	0.56	848
weighted avg	0.80	0.66	0.70	848

- Calculates and returns the accuracy of the predictions after oversampling.

```
metrics.accuracy_score(y_test,predictions1)
```

```
0.6556603773584906
```

- Prints the confusion matrix after using the SMOTE-oversampled model

```
print(confusion_matrix(y_test,predictions1))
```

```
[[478 241]
 [ 51  78]]
```

- Undersampling technique was used; print labels before undersampling

```
print("Before Under Sampling counts of label '1' is : {}".format(sum(y_train==1)))
```

```
print("Before Under Sampling counts of label '0' is : {}
\n".format(sum(y_train==0)))
```

```
Before Under Sampling counts of label '1' is : 515
Before Under Sampling counts of label '0' is : 2875
```

- NearMiss an undersampling technique was used; resamples the training data by undersampling;the shape was printed after

```
nr = NearMiss()
x_train_miss, y_train_miss = nr.fit_resample(x_train,
y_train.ravel())
print("After Under Sampling the shape of x train is:
{}".format(x_train_miss.shape))
print("After Under Sampling the shape of y train is: {}
\n".format(y_train_miss.shape))

print("After Under Sampling counts of label '1' is :
{}".format(sum(y_train_miss==1)))
print("After Under Sampling counts of label '0' is : {}
\n".format(sum(y_train_miss==0)))
```

```
After Under Sampling the shape of x train is: (1030, 8)
After Under Sampling the shape of y train is: (1030,)

After Under Sampling counts of label '1' is : 515
After Under Sampling counts of label '0' is : 515
```

- New Logistic Regression model was created

```
lr2 = LogisticRegression()
lr2.fit(x_train_miss, y_train_miss.ravel())
```

```
LogisticRegression
LogisticRegression()
```

- Trained model to make predictions on the test data and prints the classification report after training with undersampled data

```
predictions2 = lr2.predict(x_test)
print(classification_report(y_test,predictions2))
```

```
              precision    recall  f1-score   support

           0       0.90      0.58      0.71       719
           1       0.21      0.63      0.32       129

    accuracy                           0.59       848
   macro avg       0.55      0.61      0.51       848
weighted avg       0.79      0.59      0.65       848
```

- Calculates and returns the accuracy of the predictions after undersampling.

```
metrics.accuracy_score(y_test,predictions2)
```

```
0.589622641509434
```

- Confusion matrix was printed after undersampling

```
print(confusion_matrix(y_test,predictions2))
```

```
[[419 300]
 [ 48  81]]
```

- After the comparison of two techniques, it was estimated that the most balanced result was achieved by using **oversampling**

```
print("Before oversampling the accuracy is:",
      round(metrics.accuracy_score(y_test, predictions), 2),"and
confusion matrix is: ")
print(confusion_matrix(y_test,predictions))

print("After oversampling the accuracy is:",
      round(metrics.accuracy_score(y_test, predictions1), 2),"and
confusion matrix is: ")
print(confusion_matrix(y_test,predictions1))

print("After undersampling the accuracy is:",
      round(metrics.accuracy_score(y_test, predictions2), 2),"and
confusion matrix is: ")
print(confusion_matrix(y_test,predictions2))
```

```
Before oversampling the accuracy is: 0.85 and confusion matrix is:
[[711   8]
 [121   8]]
After oversampling the accuracy is: 0.66 and confusion matrix is:
[[478 241]
 [ 51  78]]
After undersampling the accuracy is: 0.59 and confusion matrix is:
[[419 300]
 [ 48  81]]
```

Modeling

KNN model

- Check and verify the dimensions (shape) of the training and test datasets to build KNN model.

```
print("The shape of x train is: {}".format(x_train_res.shape))
print("The shape of y train is: {} \n".format(y_train_res.shape))

print("The shape of x test is: {}".format((x_test.shape)))
print("The shape of y test is: {}".format((y_test.shape)))
```

```
The shape of x train is: (5750, 8)
The shape of y train is: (5750,)

The shape of x test is: (848, 8)
The shape of y test is: (848,)
```

- Initializes KNN classifier with the following parameters

```
knn1 = KNeighborsClassifier(n_neighbors = 4, metric = 'minkowski',
p=2)
knn1.fit(x_train_res,y_train_res.ravel())
```

```
KNeighborsClassifier
KNeighborsClassifier(n_neighbors=4)
```

- KNN model to make predictions on the test data

```
pred1 = knn1.predict(x_test)
```

- Prints the accuracy of the KNN model on the test data

```
print(knn1.score(x_test,y_test))
```

```
0.6981132075471698
```

- Prints a detailed classification report for the KNN model

```
print(classification_report(y_test,pred1))
```

```
              precision    recall  f1-score   support

           0       0.87      0.75      0.81       719
           1       0.22      0.40      0.28       129

    accuracy                           0.70       848
   macro avg       0.55      0.57      0.55       848
weighted avg       0.77      0.70      0.73       848
```

- Prints the confusion matrix, which shows how well the model's predictions match the actual labels

```
print(confusion_matrix(y_test, pred1))
```

```
[[541 178]
 [ 78  51]]
```

LInear SVM model

- The first model-linear SVM; SVC is being configured to classify data using

```
myModel = SVC(kernel = 'rbf', random_state = 0)
myModel.fit(x_train_res, y_train_res)
C = 1.0
svc1 = SVC(kernel = 'linear', C=C).fit(x_train_res, y_train_res)
```

```
SVC
SVC(random_state=0)
```

- Prints the accuracy of the linear SVC model

```
y_pred_svc1 = svc1.predict(x_test)
print('Accuracy of linear SVC: {:.2f}'.format(accuracy_score(y_test,
y_pred_svc1)))
```

```
Accuracy of linear SVC: 0.64
```

- Prints a detailed classification report for the linear SVC model

```
print(classification_report(y_test, y_pred_svc1))
```

	precision	recall	f1-score	support
0	0.91	0.64	0.75	719
1	0.24	0.64	0.35	129
accuracy			0.64	848
macro avg	0.57	0.64	0.55	848
weighted avg	0.81	0.64	0.69	848

- Prints the confusion matrix, which shows how well the model's predictions match the actual labels

```
print(confusion_matrix(y_test, y_pred_svc1))
```

```
[[457 262]
 [ 46  83]]
```

Different kernel SVM

- Two different SVM (kernel) models were built: SVM RBF and SVM POLY, but after the classification report it RBF model was giving a bad accuracy because for the accuracy of value 1 the model had 0 for precision.

RBF model

- Calculate the accuracy

```
C = 1.0
rbf_svc = SVC(kernel = 'rbf', gamma =0.7, C=C).fit(x_train_res,
y_train_res)
y_pred_rbf = rbf_svc.predict(x_test)
print('Accuracy of rbf SVC: {:.2f}'.format(accuracy_score(y_test,
y_pred_rbf)))
```

```
Accuracy of rbf SVC: 0.85
```

- Print the classification report

```
print(classification_report(y_test, y_pred_rbf))
```

```
              precision    recall  f1-score   support

           0       0.85      1.00      0.92       719
           1       0.00      0.00      0.00       129

    accuracy                           0.85       848
   macro avg       0.42      0.50      0.46       848
weighted avg       0.72      0.85      0.78       848
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1531: UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 in labels with no predicted samples.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1531: UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 in labels with no predicted samples.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1531: UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 in labels with no predicted samples.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
```

- Prints the confusion matrix for the model. It can be seen that all 0 values were identified correctly,while all 1 values were identified incorrectly.

```
print(confusion_matrix(y_test, y_pred_rbf))
```

```
[[719   0]
 [129   0]]
```

SVM POLY model

```
C = 1.0
poly_svc = SVC(kernel = 'poly', degree = 3, C=C).fit(x_train_res,
y_train_res)
y_pred_poly = poly_svc.predict(x_test)
print('Accuracy of poly: {:.2f}'.format(accuracy_score(y_test,
y_pred_poly)))
```

```
Accuracy of poly: 0.66
```

- Prints a detailed classification report for the SVC(POLY) model

```
print(classification_report(y_test, y_pred_poly))
```

	precision	recall	f1-score	support
0	0.90	0.68	0.77	719
1	0.24	0.56	0.33	129
accuracy			0.66	848
macro avg	0.57	0.62	0.55	848
weighted avg	0.80	0.66	0.70	848

- Prints the confusion matrix, which shows how well the model's predictions match the actual labels

```
[[487 232]
 [ 57  72]]
```

Naive Bayes model

- Initializes a Gaussian Naive Bayes (GaussianNB) classifier to build Naive Bayes model

```
gnb1= GaussianNB()
predNB = gnb1.fit(x_train_res, y_train_res).predict(x_test)
```

- Prints the accuracy score of the model on the test data

```
print(gnb1.score(x_test,y_test))
```

```
0.7346698113207547
```

- Prints a detailed classification report for the KNN model

```
print(classification_report(y_test,predNB))
```

```
              precision    recall  f1-score   support

           0       0.88      0.80      0.84       719
           1       0.26      0.40      0.31       129

    accuracy                           0.73       848
   macro avg       0.57      0.60      0.57       848
weighted avg       0.79      0.73      0.76       848
```

- Prints the confusion matrix, which shows how well the model's predictions match the actual labels

```
print(confusion_matrix(y_test,predNB))
```

```
[[572 147]
 [ 78  51]]
```

After implementing linear SVM, different kernel SVM, Naïve Bayes and KNN classifiers to build models it is concluded that the best model for classification Naïve Bayes with an accuracy of 0.734.