Imputing missing values with SMOTE Technique

```
##### # Imports

# Pandas and numpy for data manipulation
import pandas as pd
import numpy as np

# No warnings about setting value on copy of slice
pd.options.mode.chained_assignment = None

# Display up to 60 columns of a dataframe
pd.set_option('display.max_columns', 60)

# Matplotlib visualization
import matplotlib.pyplot as plt
%matplotlib inline

# Set default font size
plt.rcParams['font.size'] = 24

# Internal ipython tool for setting figure size
from IPython.core.pylabtools import figsize

# Seaborn for visualization
import seaborn as sns
sns.set(font_scale = 2)

# Splitting data into training and testing
from sklearn.model_selection import train_test_split

from sklearn.linear_model import LogisticRegression,LinearRegression
```

```python
# %% [code]
# # # Data Cleaning and Formatting

# # Load in the Data and Examine

# Read in credit into a dataframe
credit = pd.read_csv('../input/my-dataset/credit_train.csv')

# Display top of dataframe
credit.head()

# %% [code]
credit.shape

# %% [markdown]
# **Handling NA vs Changing the type of Feilds**
#
# Always Make sure that ,You should handle Null values in Data on first priority and the typecase them into some categories or in some other form.
# Lets Find out the Row level Duplicate and Remove them ,Because they are of no Use to us.
#

# %% [code]
#Check for all Row level NULL(It means all column values for that row are null) from Dataframe because they are not carring any information.
credit=credit[credit.isna().all(axis=1)==False]

# %% [code]
#Check whether Row level NULL are hablded or Not
credit.shape
```

```
# %% [markdown]

# As we can see, it looks like some of the credit score are just scaled up by 10. For the ease of our calculation we can consider, scaling them back is accurate.


# %% [code]

credit.describe()


# %% [code]

# # Data Types and Missing Values


# See the column data types and non-missing values

credit.info()


# %% [code]

# Statistics for each column

credit.describe()


# %% [code]

credit.drop(labels=['Loan ID', 'Customer ID'], axis=1, inplace=True)


# These two features are only for identification.


# %% [markdown]

# **hANDLINF CATEGORICAL FEATURES**


# %% [code]

credit['Loan Status'] = credit['Loan Status'].map({'Fully Paid':int('0'),'Charged Off':int('1')})

credit['Term'] = credit['Term'].map({'Short Term':int('0'),'Long Term':int('1')})

credit['Years in current job'] = credit['Years in current job'].map({'< 1 year':int('0'),'1 year':int('1'),'2 years':int('2'),'3 years':int('3'),'4 years':int('4'),'5 years':int('5'),'6 years':int('6'),'7 years':int('7'),'8 years':int('8'),'9 years':int('9'),'10+ years':int('10')})
```

```
# %% [code]
del(credit['Months since last delinquent'])

# %% [code]
# # Encoding categorical data & Feature Scaling

# Select the categorical columns
categorical_subset = credit[['Home Ownership', 'Purpose']]

# One hot encode
categorical_subset = pd.get_dummies(categorical_subset)

# Join the dataframe in credit_train
# Make sure to use axis = 1 to perform a column bind
# First I will drop the 'old' categorical datas and after I will join the 'new' one.

credit.drop(labels=['Home Ownership', 'Purpose'], axis=1, inplace=True)
credit = pd.concat([credit, categorical_subset], axis = 1)

# %% [code]
credit.shape

# %% [code]
# #  Remove Collinear Features

def remove_collinear_features(x, threshold):
    '''
    Objective:
        Remove collinear features in a dataframe with a correlation coefficient
        greater than the threshold. Removing collinear features can help a model
```

```
    to generalize and improves the interpretability of the model.

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

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

# Dont want to remove correlations between Energy Star Score
y = x['Loan Status']
x = x.drop(columns = ['Loan Status'])

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

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

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

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

    # Add the score back in to the data
    x['Loan Status'] = y

    return x

# %% [code]
# Remove the collinear features above a specified correlation coefficient
credit = remove_collinear_features(credit, 0.6);

# %% [code]
credit.shape

# %% [code]
# # Missing Values

# Function to calculate missing values by column
def missing_values_table(df):
        # Total missing values
        mis_val = df.isnull().sum()

        # Percentage of missing values
        mis_val_percent = 100 * df.isnull().sum() / len(df)

        # Make a table with the results
        mis_val_table = pd.concat([mis_val, mis_val_percent], axis=1)
```

```
    # Rename the columns
    mis_val_table_ren_columns = mis_val_table.rename(
    columns = {0 : 'Missing Values', 1 : '% of Total Values'})

    # Sort the table by percentage of missing descending
    mis_val_table_ren_columns = mis_val_table_ren_columns[
      mis_val_table_ren_columns.iloc[:,1] != 0].sort_values(
    '% of Total Values', ascending=False).round(1)

    # Print some summary information
    print ("Your selected dataframe has " + str(df.shape[1]) + " columns.\n"
      "There are " + str(mis_val_table_ren_columns.shape[0]) +
        " columns that have missing values.")

    # Return the dataframe with missing information
    return mis_val_table_ren_columns

# %% [code]
missing_values_table(credit)

# A curious thing about the table below is the last 10 features have the same number o missing values.
# I will go deeper and figure out what is happening.

# %% [markdown]
# # # Handling missing value with correlation methods

# %% [code]
corr = credit.corr()
corr
```

```
# %% [code]
sns.heatmap(corr)

# %% [code]
corr['Credit Score'][abs(corr['Credit Score']) > 0.1]

# %% [code]
corr['Annual Income'][abs(corr['Annual Income']) > 0.1]

# %% [code]
corr['Years in current job'][abs(corr['Years in current job']) > 0.1]

# %% [code]
corr['Tax Liens'][abs(corr['Tax Liens']) > 0.1]

# %% [code]
corr['Maximum Open Credit'][abs(corr['Maximum Open Credit']) > 0.1]

# %% [code]
credit_without_mv = credit.dropna()

# %% [code]
x1_col = ['Loan Status']
y1 = credit_without_mv['Credit Score']
x1 = credit_without_mv['Loan Status']

# %% [code]



# %% [code]
y1 = y1.values.reshape(77427,1)
```

```
x1 = x1.values.reshape(77427,1)


# %% [code]

linreg1 = LinearRegression()

linreg1.fit(x1,y1)


# %% [code]

credit['Credit Score'] = credit.apply(lambda x:linreg1.predict(x['Loan Status'].reshape(1,1))[0][0] if np.isnan(x['Credit Score']) else x['Credit Score'], axis =1)


# %% [code]

credit['Credit Score'].shape


# %% [code]

# for Annual income


x2_col = ['Monthly Debt', 'Years of Credit History', 'Number of Open Accounts', 'Current Credit Balance', 'Home Ownership_Home Mortgage', 'Home Ownership_Rent']

y2 = credit_without_mv['Annual Income']

x2 = credit_without_mv[x2_col]

y2 = y2.values.reshape(77427,1)

x2 = x2.values.reshape(77427,6)

linreg2 = LinearRegression()

linreg2.fit(x2,y2)

credit['Annual Income'] = credit.apply(lambda x:linreg2.predict(x[x2_col].values.reshape(1,6))[0][0] if np.isnan(x['Annual Income']) else x['Annual Income'], axis =1)

credit['Annual Income'].shape


# %% [code]

# for Years in current job
```

```
x3_col = ['Monthly Debt', 'Years of Credit History', 'Home Ownership_Home Mortgage', 'Home Ownership_Rent']

y3 = credit_without_mv['Years in current job']

x3 = credit_without_mv[x3_col]

y3 = y3.values.reshape(77271,1)

x3 = x3.values.reshape(77271,4)

linreg3 = LogisticRegression()

linreg3.fit(x3,y3)

credit['Years in current job'] = credit.apply(lambda x:linreg3.predict(x[x3_col].values.reshape(1,4))[0:4][0] if np.isnan(x['Years in current job']) else x['Years in current job'], axis =1)

credit['Years in current job'].shape


# %% [code]

# for Maximum Open Credit


y4 = credit_without_mv['Maximum Open Credit']

x4 = credit_without_mv['Current Credit Balance']

y4 = y4.values.reshape(77271,1)

x4 = x4.values.reshape(77271,1)

linreg4 = LinearRegression()

linreg4.fit(x4,y4)

credit['Maximum Open Credit'] = credit.apply(lambda x:linreg4.predict(x['Current Credit Balance'].reshape(1,1))[0][0] if np.isnan(x['Maximum Open Credit']) else x['Maximum Open Credit'], axis =1)

credit['Maximum Open Credit'].shape


# %% [code]

# for Bankruptcies


x5_col = ['Current Credit Balance', 'Number of Credit Problems']

y5 = credit_without_mv['Bankruptcies']

x5 = credit_without_mv[x5_col]
```

```
y5 = y5.values.reshape(77271,1)

x5 = x5.values.reshape(77271,2)

linreg5 = LinearRegression()

linreg5.fit(x5,y5)

credit['Bankruptcies'] = credit.apply(lambda x:linreg5.predict(x[x5_col].values.reshape(1,2))[0][0]
if np.isnan(x['Bankruptcies']) else x['Bankruptcies'], axis =1)

credit['Bankruptcies'].shape


# %% [code]

# for tax liens


y6 = credit_without_mv['Tax Liens']

x6 = credit_without_mv['Number of Credit Problems']

y6 = y6.values.reshape(77271,1)

x6 = x6.values.reshape(77271,1)

linreg6 = LinearRegression()

linreg6.fit(x6,y6)

credit['Tax Liens'] = credit.apply(lambda x:linreg6.predict(x['Number of Credit
Problems'].reshape(1,1))[0][0] if np.isnan(x['Tax Liens']) else x['Tax Liens'], axis =1)

credit['Tax Liens'].shape


# %% [markdown]

# # End hadling missing value using correlation


# %% [markdown]

# **start SMOTE**


# %% [code]

from sklearn.metrics import accuracy_score

from sklearn.metrics import precision_score, recall_score

from sklearn.metrics import f1_score, roc_auc_score, roc_curve

from sklearn.model_selection import train_test_split
```

```
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import confusion_matrix
import math

def generate_model_report(y_actual, y_predicted):

    conf_mat = confusion_matrix(y_actual, y_predicted)
    true_positive = conf_mat[1,1]
    true_negative = conf_mat[0,0]
    false_positive = conf_mat[0,1]
    false_negative = conf_mat[1,0]
    specificity = (true_negative)/(true_negative + false_positive)
    gm = math.sqrt(specificity * recall_score(y_actual, y_predicted))

    print("Accuracy = " , accuracy_score(y_actual, y_predicted))
    print("Precision = " ,precision_score(y_actual, y_predicted))
    print("Recall/Sensitivity = " ,recall_score(y_actual, y_predicted))
    print("Specificity = " ,specificity)
    print("F1 Score = " ,f1_score(y_actual, y_predicted))
    print("ROC-AUC Score = " ,roc_auc_score(y_actual, y_predicted))
    print("G-Measure = " ,gm)

    sns.heatmap(conf_mat,cmap="coolwarm_r", annot=True,linewidths=0.5,fmt='g')
    plt.title("Confusion Matrix")
    plt.xlabel("Predicted value")
    plt.ylabel("Actual label")
    plt.show()
    pass

def generate_auc_roc_curve(clf, X_test):
    y_pred_proba = clf.predict_proba(X_test)[:, 1]
```

```
    fpr, tpr, thresholds = roc_curve(Y_test,  y_pred_proba)
    auc = roc_auc_score(Y_test, y_pred_proba)
    plt.plot(fpr,tpr,label="AUC ="+str(auc))
    plt.legend(loc=4)
    plt.show()
    pass



# %% [code]
X = credit.loc[:, credit.columns!='Loan Status']


Y = credit.loc[:, credit.columns=='Loan Status']


from sklearn.model_selection import train_test_split
X_train, X_test, Y_train, Y_test = train_test_split(X, Y,
                                                    test_size=0.2,
                                                    random_state=42)


# %% [code]
# # Feature Scaling
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)


# Encoding the Dependent Variable
from sklearn.preprocessing import LabelEncoder, OneHotEncoder
labelencoder_y_train = LabelEncoder()
Y_train = labelencoder_y_train.fit_transform(Y_train)
labelencoder_y_test = LabelEncoder()
Y_test = labelencoder_y_test.fit_transform(Y_test)
```

```
# %% [code]
from imblearn.over_sampling import SMOTE
from imblearn.pipeline import make_pipeline
sm = SMOTE(random_state=12, ratio = 1.0)
x_train_res, y_train_res = sm.fit_sample(X_train, Y_train)

unique, count = np.unique(y_train_res, return_counts=True)
y_train_smote_value_count = { k:v for (k,v) in zip(unique, count)}
y_train_smote_value_count



# %% [code]
print("logistic regression")
clf = LogisticRegression().fit(x_train_res, y_train_res)
Y_Test_Pred = clf.predict(X_test)
generate_model_report(Y_test, Y_Test_Pred)
generate_auc_roc_curve(clf, X_test)

# %% [code]
from sklearn.neighbors import KNeighborsClassifier
print("KNN")

clf = KNeighborsClassifier(n_neighbors = 5, metric = 'minkowski', p = 2).fit(x_train_res, y_train_res)
Y_Test_Pred = clf.predict(X_test)
generate_model_report(Y_test, Y_Test_Pred)
generate_auc_roc_curve(clf, X_test)

# %% [code]
from sklearn import tree
print("Decision Tree")
```

```
clf = tree.DecisionTreeClassifier(random_state=1).fit(x_train_res, y_train_res)
Y_Test_Pred = clf.predict(X_test)
generate_model_report(Y_test, Y_Test_Pred)
generate_auc_roc_curve(clf, X_test)

# %% [code]
print("Naive bayes")
from sklearn.naive_bayes import GaussianNB

clf = GaussianNB().fit(x_train_res, y_train_res)
Y_Test_Pred = clf.predict(X_test)
generate_model_report(Y_test, Y_Test_Pred)
generate_auc_roc_curve(clf, X_test)

# %% [code]
print("Random forest")
from sklearn.ensemble import RandomForestClassifier


clf = RandomForestClassifier(n_estimators = 10, criterion = 'entropy').fit(x_train_res, y_train_res)
Y_Test_Pred = clf.predict(X_test)
generate_model_report(Y_test, Y_Test_Pred)
generate_auc_roc_curve(clf, X_test)

# %% [code]
print("XGBoost")
from xgboost import XGBClassifier

clf = XGBClassifier().fit(x_train_res, y_train_res)
Y_Test_Pred = clf.predict(X_test)
```

```
generate_model_report(Y_test, Y_Test_Pred)
generate_auc_roc_curve(clf, X_test)


# %% [code]
# # # Models to Evaluate

# We will compare five different machine learning Classification models:

# 1 - Logistic Regression
# 2 - K-Nearest Neighbors Classification
# 3 - Suport Vector Machine
# 4 - Naive Bayes
# 5 - Random Forest Classification

# Function to calculate mean absolute error
def cross_val(X_train, y_train, model):
    # Applying k-Fold Cross Validation
    from sklearn.model_selection import cross_val_score

    accuracies = cross_val_score(estimator = model, X = X_train, y = y_train, cv = 10, verbose = 2)
    return accuracies.mean()



def confusion_metrix(X_train, y_train, model):

    from sklearn.model_selection import cross_val_predict
    from sklearn.metrics import accuracy_score
    from sklearn.metrics import precision_score, recall_score
    from sklearn.metrics import f1_score, roc_auc_score, roc_curve
    import math
```

```
    ## confusion metrix
    from sklearn.metrics import confusion_matrix
    y_pred = cross_val_predict(model, X_train, y_train, cv=3)
    conf_mat = confusion_matrix(y_train, y_pred)
    true_positive = conf_mat[1,1]
    true_negative = conf_mat[0,0]
    false_positive = conf_mat[0,1]
    false_negative = conf_mat[1,0]
    specificity = (true_negative)/(true_negative + false_positive)
    gm = math.sqrt(specificity * recall_score(y_train, y_pred))



    print("Accuracy = " , accuracy_score(y_train, y_pred))
    print("Precision = " ,precision_score(y_train, y_pred))
    print("Recall/ Sensitivity = " ,recall_score(y_train, y_pred))
    print("Specificity = " ,specificity)
    print("F1 Score = " ,f1_score(y_train, y_pred))
    print("ROC-AUC Score = " ,roc_auc_score(y_train, y_pred))
    print("G-Measure = " ,gm)
    return conf_mat


# Takes in a model, trains the model, and evaluates the model on the test set
def fit_and_evaluate(model):

    # Train the model
    #model.fit(X_train, y_train)

    # Make predictions and evalute
    #model_pred = model.predict(X_test)
```

```
    model_acc_cross = cross_val(x_train_res, y_train_res, model)
    print ("print accuracy is ",model_acc_cross)

    con_matrix = confusion_metrix(x_train_res, y_train_res, model)
    print ("print confusion metrix is ",con_matrix)
    sns.heatmap(con_matrix,cmap="coolwarm_r", annot=True,linewidths=0.5,fmt='g')
    plt.title("Confusion Matrix")
    plt.xlabel("Predicted value")
    plt.ylabel("Actual label")
    plt.show()


    # Return the performance metric
    return model_acc_cross

# %% [code]
# # Logistic Regression
from sklearn.linear_model import LogisticRegression
logr = LogisticRegression()
logr_cross = fit_and_evaluate(logr)

print('Logistic Regression Performance on the test set: Cross Validation Score = %0.4f' %
logr_cross)

# %% [code]
# # K-NN
from sklearn.neighbors import KNeighborsClassifier
knn = KNeighborsClassifier(n_neighbors = 5, metric = 'minkowski', p = 2)
knn_cross = fit_and_evaluate(knn)

print('KNN Performance on the test set: Cross Validation Score = %0.4f' % knn_cross)
```

```
# %% [code]

# # Naive Bayes

from sklearn.naive_bayes import GaussianNB

naive = GaussianNB()

naive_cross = fit_and_evaluate(naive)


print('Naive Bayes Performance on the test set: Cross Validation Score = %0.4f' % naive_cross)


# %% [code]

 # Random Forest Classification

from sklearn.ensemble import RandomForestClassifier

random = RandomForestClassifier(n_estimators = 10, criterion = 'entropy')

random_cross = fit_and_evaluate(random)


print('Random Forest Performance on the test set: Cross Validation Score = %0.4f' %
random_cross)


# %% [code]

# # Gradiente Boosting Classification

from xgboost import XGBClassifier

gb = XGBClassifier()

gb_cross = fit_and_evaluate(gb)


print('Gradiente Boosting Classification Performance on the test set: Cross Validation Score =
%0.4f' % gb_cross)


# %% [code]

# # Decision tree

from sklearn import tree

dt = tree.DecisionTreeClassifier(random_state=1)

dt_cross = fit_and_evaluate(dt)
```

```
print('Decision tree Performance on the test set: Cross Validation Score = %0.4f' % dt_cross)
```

Imputing missing values with UndersamplingTechnique

```
# %% [code]

# This Python 3 environment comes with many helpful analytics libraries installed

# It is defined by the kaggle/python docker image: https://github.com/kaggle/docker-python

# For example, here's several helpful packages to load in


import numpy as np # linear algebra

import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)


# Input data files are available in the "../input/" directory.

# For example, running this (by clicking run or pressing Shift+Enter) will list all files under the input directory


import os

for dirname, _, filenames in os.walk('/kaggle/input'):

    for filename in filenames:

        print(os.path.join(dirname, filename))


# Any results you write to the current directory are saved as output.


# %% [code]

##### # Imports


# Pandas and numpy for data manipulation

import pandas as pd

import numpy as np


# No warnings about setting value on copy of slice

pd.options.mode.chained_assignment = None


# Display up to 60 columns of a dataframe
```

```
pd.set_option('display.max_columns', 60)

# Matplotlib visualization
import matplotlib.pyplot as plt
%matplotlib inline

# Set default font size
plt.rcParams['font.size'] = 24

# Internal ipython tool for setting figure size
from IPython.core.pylabtools import figsize

# Seaborn for visualization
import seaborn as sns
sns.set(font_scale = 2)

# Splitting data into training and testing
from sklearn.model_selection import train_test_split

# %% [code]
 # # Data Cleaning and Formatting

# # Load in the Data and Examine

# Read in credit into a dataframe
credit = pd.read_csv('../input/my-dataset/credit_train.csv')

# Display top of dataframe
credit.head()

# %% [code]
```

```
credit.shape

# %% [code]
credit.drop(labels=['Loan ID', 'Customer ID'], axis=1, inplace=True)

# These two features are only for identification.

# %% [code]
# # Data Types and Missing Values

# See the column data types and non-missing values
credit.info()

# %% [code]
# # Missing Values

# Function to calculate missing values by column
def missing_values_table(df):
    # Total missing values
    mis_val = df.isnull().sum()

    # Percentage of missing values
    mis_val_percent = 100 * df.isnull().sum() / len(df)

    # Make a table with the results
    mis_val_table = pd.concat([mis_val, mis_val_percent], axis=1)

    # Rename the columns
    mis_val_table_ren_columns = mis_val_table.rename(
    columns = {0 : 'Missing Values', 1 : '% of Total Values'})
```

```
    # Sort the table by percentage of missing descending
    mis_val_table_ren_columns = mis_val_table_ren_columns[
        mis_val_table_ren_columns.iloc[:,1] != 0].sort_values(
    '% of Total Values', ascending=False).round(1)

    # Print some summary information
    print ("Your selected dataframe has " + str(df.shape[1]) + " columns.\n"
        "There are " + str(mis_val_table_ren_columns.shape[0]) +
          " columns that have missing values.")

    # Return the dataframe with missing information
    return mis_val_table_ren_columns

# %% [code]
missing_values_table(credit)

# %% [code]
credit.drop(credit.tail(514).index, inplace=True) # drop last 514 rows
missing_values_table(credit)

# %% [code]
data_without_ms = credit.dropna()

# %% [code]
data_without_ms.info()

# %% [code]
## caregorical data to numerical

data_without_ms['Loan Status'] = data_without_ms['Loan Status'].map({'Fully
Paid':int('0'),'Charged Off':int('1')})
```

```
data_without_ms['Term'] = data_without_ms['Term'].map({'Short Term':int('0'),'Long Term':int('1')})

data_without_ms['Years in current job'] = data_without_ms['Years in current job'].map({'< 1 year':int('0'),'1 year':int('1'),'2 years':int('2'),'3 years':int('3'),'4 years':int('4'),'5 years':int('5'),'6 years':int('6'),'7 years':int('7'),'8 years':int('8'),'9 years':int('9'),'10+ years':int('10')})



# %% [code]
# # Encoding categorical data & Feature Scaling

# Select the categorical columns
categorical_subset = data_without_ms[[ 'Home Ownership', 'Purpose']]

# One hot encode
categorical_subset = pd.get_dummies(categorical_subset)

# Join the dataframe in credit_train
# Make sure to use axis = 1 to perform a column bind
# First I will drop the 'old' categorical datas and after I will join the 'new' one.

data_without_ms.drop(labels=['Home Ownership', 'Purpose'], axis=1, inplace=True)
data_without_ms = pd.concat([data_without_ms, categorical_subset], axis = 1)

# %% [code]
data_without_ms.head()

# %% [code]
data_without_ms.shape

# %% [markdown]
# **Handling under sampling**
```

```
# %% [code]
from sklearn.metrics import accuracy_score
from sklearn.metrics import precision_score, recall_score
from sklearn.metrics import f1_score, roc_auc_score, roc_curve
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import confusion_matrix
import math

def generate_model_report(y_actual, y_predicted):

   conf_mat = confusion_matrix(y_actual, y_predicted)
   true_positive = conf_mat[1,1]
   true_negative = conf_mat[0,0]
   false_positive = conf_mat[0,1]
   false_negative = conf_mat[1,0]
   specificity = (true_negative)/(true_negative + false_positive)
   gm = math.sqrt(specificity * recall_score(y_actual, y_predicted))

   print("Accuracy = " , accuracy_score(y_actual, y_predicted))
   print("Precision = " ,precision_score(y_actual, y_predicted))
   print("Recall/Sensitivity = " ,recall_score(y_actual, y_predicted))
   print("Specificity = " ,specificity)
   print("F1 Score = " ,f1_score(y_actual, y_predicted))
   print("ROC-AUC Score = " ,roc_auc_score(y_actual, y_predicted))
   print("G-Measure = " ,gm)

   sns.heatmap(conf_mat,cmap="coolwarm_r", annot=True,linewidths=0.5,fmt='g')
   plt.title("Confusion Matrix")
   plt.xlabel("Predicted value")
   plt.ylabel("Actual label")
```

```
    plt.show()
    pass

minority_class_len = len(data_without_ms[data_without_ms['Loan Status'] == 1])
print(minority_class_len)

majority_class_indices = data_without_ms[data_without_ms['Loan Status'] == 0].index
print(majority_class_indices)

random_majority_indices = np.random.choice(majority_class_indices,
                                           minority_class_len,
                                           replace=False)
print(len(random_majority_indices))

minority_class_indices = data_without_ms[data_without_ms['Loan Status'] == 1].index
print(minority_class_indices)

under_sample_indices = np.concatenate([minority_class_indices,random_majority_indices])
#under_sample = data_without_ms.loc[under_sample_indices]
data_without_ms = data_without_ms.loc[under_sample_indices]




# %% [code]

def generate_auc_roc_curve(clf, X_test):
    y_pred_proba = clf.predict_proba(X_test)[:, 1]
    fpr, tpr, thresholds = roc_curve(Y_test,  y_pred_proba)
    auc = roc_auc_score(Y_test, y_pred_proba)
    plt.plot(fpr,tpr,label="AUC  ="+str(auc))
    plt.legend(loc=4)
```

```
    plt.show()
    pass

# %% [code]
print("logistic regression")
X = data_without_ms.loc[:, data_without_ms.columns!='Loan Status']
Y = data_without_ms.loc[:, data_without_ms.columns=='Loan Status']
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.2, random_state=42)
clf = LogisticRegression().fit(X_train, Y_train)
Y_Test_Pred = clf.predict(X_test)

generate_model_report(Y_test, Y_Test_Pred)
generate_auc_roc_curve(clf, X_test)



# %% [code]
from sklearn.neighbors import KNeighborsClassifier
print("KNN")

clf = KNeighborsClassifier(n_neighbors = 5, metric = 'minkowski', p = 2).fit(X_train, Y_train)

Y_Test_Pred = clf.predict(X_test)

generate_model_report(Y_test, Y_Test_Pred)
generate_auc_roc_curve(clf, X_test)

# %% [code]
from sklearn import tree
print("Decision Tree")

clf = tree.DecisionTreeClassifier(random_state=1).fit(X_train, Y_train)
```

```
Y_Test_Pred = clf.predict(X_test)

generate_model_report(Y_test, Y_Test_Pred)
generate_auc_roc_curve(clf, X_test)

# %% [code]
print("Naive bayes")
from sklearn.naive_bayes import GaussianNB

clf = GaussianNB().fit(X_train, Y_train)

Y_Test_Pred = clf.predict(X_test)

generate_model_report(Y_test, Y_Test_Pred)
generate_auc_roc_curve(clf, X_test)

# %% [code]
print("Random forest")
from sklearn.ensemble import RandomForestClassifier


clf = RandomForestClassifier(n_estimators = 10, criterion = 'entropy').fit(X_train, Y_train)

Y_Test_Pred = clf.predict(X_test)

generate_model_report(Y_test, Y_Test_Pred)
generate_auc_roc_curve(clf, X_test)

# %% [code]
print("XGBoost")
```

```
from xgboost import XGBClassifier

clf = XGBClassifier().fit(X_train, Y_train)

Y_Test_Pred = clf.predict(X_test)

generate_model_report(Y_test, Y_Test_Pred)
generate_auc_roc_curve(clf, X_test)

# %% [markdown]
# # k-flod

# %% [code]
 # # Split Into Training and Testing Sets

# Separate out the features and targets
features = data_without_ms.drop(columns='Loan Status')
targets = pd.DataFrame(data_without_ms['Loan Status'])

# Split into 80% training and 20% testing set
X_train, X_test, y_train, y_test = train_test_split(features, targets, test_size = 0.2, random_state = 16)

print(X_train.shape)
print(X_test.shape)
print(y_train.shape)
print(y_test.shape)

# %% [code]
# # Feature Scaling
from sklearn.preprocessing import StandardScaler
```

```
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)

# Encoding the Dependent Variable
from sklearn.preprocessing import LabelEncoder, OneHotEncoder
labelencoder_y_train = LabelEncoder()
y_train = labelencoder_y_train.fit_transform(y_train)
labelencoder_y_test = LabelEncoder()
y_test = labelencoder_y_test.fit_transform(y_test)

# %% [code]
# # # Models to Evaluate

# We will compare five different machine learning Classification models:

# 1 - Logistic Regression
# 2 - K-Nearest Neighbors Classification
# 3 - Suport Vector Machine
# 4 - Naive Bayes
# 5 - Random Forest Classification

# Function to calculate mean absolute error
def cross_val(X_train, y_train, model):
  # Applying k-Fold Cross Validation
  from sklearn.model_selection import cross_val_score

  accuracies = cross_val_score(estimator = model, X = X_train, y = y_train, cv = 10, verbose = 2)
  return accuracies.mean()
```

```
def confusion_metrix(X_train, y_train, model):

    from sklearn.model_selection import cross_val_predict
    from sklearn.metrics import accuracy_score
    from sklearn.metrics import precision_score, recall_score
    from sklearn.metrics import f1_score, roc_auc_score, roc_curve
    import math

    ## confusion metrix
    from sklearn.metrics import confusion_matrix
    y_pred = cross_val_predict(model, X_train, y_train, cv=3)
    conf_mat = confusion_matrix(y_train, y_pred)
    true_positive = conf_mat[1,1]
    true_negative = conf_mat[0,0]
    false_positive = conf_mat[0,1]
    false_negative = conf_mat[1,0]
    specificity = (true_negative)/(true_negative + false_positive)
    gm = math.sqrt(specificity * recall_score(y_train, y_pred))



    print("Accuracy = " , accuracy_score(y_train, y_pred))
    print("Precision = " ,precision_score(y_train, y_pred))
    print("Recall/ Sensitivity = " ,recall_score(y_train, y_pred))
    print("Specificity = " ,specificity)
    print("F1 Score = " ,f1_score(y_train, y_pred))
    print("ROC-AUC Score = " ,roc_auc_score(y_train, y_pred))
    print("G-Measure = " ,gm)
    return conf_mat
```

```
# Takes in a model, trains the model, and evaluates the model on the test set
def fit_and_evaluate(model):

    # Train the model
    #model.fit(X_train, y_train)

    # Make predictions and evalute
    #model_pred = model.predict(X_test)
    model_acc_cross = cross_val(X_train, y_train, model)
    print ("print accuracy is ",model_acc_cross)

    con_matrix = confusion_metrix(X_train, y_train, model)
    print ("print confusion metrix is ",con_matrix)
    sns.heatmap(con_matrix,cmap="coolwarm_r", annot=True,linewidths=0.5,fmt='g')
    plt.title("Confusion Matrix")
    plt.xlabel("Predicted value")
    plt.ylabel("Actual label")
    plt.show()



    # Return the performance metric
    return model_acc_cross

# %% [code]
# Matplotlib visualization
import matplotlib.pyplot as plt
%matplotlib inline

# Set default font size
plt.rcParams['font.size'] = 24
```

```
# Internal ipython tool for setting figure size
from IPython.core.pylabtools import figsize

# Seaborn for visualization
import seaborn as sns
sns.set(font_scale = 2)

# Splitting data into training and testing
from sklearn.model_selection import train_test_split

# %% [code]
# # Decision tree
from sklearn import tree
dt = tree.DecisionTreeClassifier(random_state=1)
dt_cross = fit_and_evaluate(dt)

print('Decision tree Performance on the test set: Cross Validation Score = %0.4f' % dt_cross)

# %% [code]
# # K-NN
from sklearn.neighbors import KNeighborsClassifier
knn = KNeighborsClassifier(n_neighbors = 5, metric = 'minkowski', p = 2)
knn_cross = fit_and_evaluate(knn)

print('KNN Performance on the test set: Cross Validation Score = %0.4f' % knn_cross)

# %% [code]
# # K-NN
from sklearn.neighbors import KNeighborsClassifier
knn = KNeighborsClassifier(n_neighbors = 1, metric = 'minkowski', p = 2)
knn_cross = fit_and_evaluate(knn)
```

```
print('KNN Performance on the test set: Cross Validation Score = %0.4f' % knn_cross)


# %% [code]

# # K-NN

# used eclbian distance

from sklearn.neighbors import KNeighborsClassifier

knn = KNeighborsClassifier(n_neighbors = 4, metric = 'minkowski', p = 2)

knn_cross = fit_and_evaluate(knn)


print('KNN Performance on the test set: Cross Validation Score = %0.4f' % knn_cross)


# %% [code]

# # Logistic Regression

from sklearn.linear_model import LogisticRegression

logr = LogisticRegression()

logr_cross = fit_and_evaluate(logr)


print('Logistic Regression Performance on the test set: Cross Validation Score = %0.4f' %
logr_cross)


# %% [code]

# # Random Forest Classification

from sklearn.ensemble import RandomForestClassifier

random = RandomForestClassifier(n_estimators = 10, criterion = 'entropy')

random_cross = fit_and_evaluate(random)


print('Random Forest Performance on the test set: Cross Validation Score = %0.4f' %
random_cross)


# %% [code]

# # Gradiente Boosting Classification
```

```
from xgboost import XGBClassifier

gb = XGBClassifier()

gb_cross = fit_and_evaluate(gb)


print('Gradiente Boosting Classification Performance on the test set: Cross Validation Score = %0.4f' % gb_cross)


# %% [code]

# # Naive Bayes

from sklearn.naive_bayes import GaussianNB

naive = GaussianNB()

naive_cross = fit_and_evaluate(naive)


print('Naive Bayes Performance on the test set: Cross Validation Score = %0.4f' % naive_cross)
```