Project Title:

Marketing Campaign for Banking Products

Data Description:

The file Bank_loan_project.csv contains data on 5000 customers. The data include customer demographic information (age, income, etc.), the customer's relationship with the bank (mortgage, securities account, etc.), and the customer response to the last personal loan campaign (Personal Loan).

Among these 5000 customers, only 480 (= 9.6%) accepted the personal loan that was offered to them in the earlier campaign.

Context:

The bank has a growing customer base. The bank wants to increase borrowers (asset customers) base to bring in more loan business and earn more through the interest on loans. So , the bank wants to convert the liability based customers to personal loan customers. (while retaining them as depositors). A campaign that the bank ran last year for liability customers showed a healthy conversion rate of over 9% success. The department wants you to build a model that will help them identify the potential customers who have a higher probability of purchasing the loan. This will increase the success ratio while at the same time reduce the cost of the campaign.

Attribute Information:

• ID: Customer ID

• Age: Customer's age in completed years

• Experience: #years of professional experience

• Income: Annual income of the customer (\$000)

• Family: Family size of the customer
• CCAvg: Avg. spending on credit cards per month (\$000)
• Education: Education Level. 1: Undergrad; 2: Graduate; 3: Advanced/Professional
Mortgage: Value of house mortgage if any. (\$000)
 Personal Loan: Did this customer accept the personal loan offered in the last campaign?
• Securities Account: Does the customer have a securities account with the bank?
CD Account: Does the customer have a certificate of deposit (CD) account with the bank?
Online: Does the customer use internet banking facilities?
• Credit card: Does the customer use a credit card issued by the bank?
1. Import the datasets and libraries, check datatype, statistical summary, shape, null values etc.
Import necessary libraries
import numpy as np
import pandas as pd
from matplotlib import pyplot as plt
import seaborn as sns

• ZIP Code: Home Address ZIP code.

```
from sklearn.impute import SimpleImputer
import scipy.stats as stats
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn import metrics
from sklearn.ensemble import RandomForestClassifier
# Importing csv data and view data
data=pd.read_csv("/content/Bank_Personal_Loan_Modelling.csv")
data
# Checking the shape of data means no. of rows and columns.
data.shape
# Checking the first 5 rows
# by default the head() method gives the first 5 rows but if we need more rows then simply mention
how many first rows you want.
# For example if we want first 10 rows then simply write data.head(10)
data.head()
# Checking the last 5 rows
# by default the tail() method gives the last 5 rows but if we need more rows then simply mention
how many last rows you want.
# For example if we want last 10 rows then simply write data.tail(10)
data.tail()
# Checking the data types of each columns
data.dtypes
# Converting the data type of 'Personal Loan' from int to category
data['Personal Loan']=data['Personal Loan'].astype('category')
#checking that the data type of 'Personal Loan' is converted to category or not
data.dtypes
# Checking the count, mean, min, max, etc values of each columns of our data
data.describe()
# Checking the concise summary of our data
data.info()
# Checking the null values in our data
```

```
data.isnull().sum()
#**2. Check if you need to clean the data for any of the variables.**
**Cleansing of data**
# Replacing all the negative values of experience column into nan values and then applied .describe
method
data['Experience'].replace( to_replace= -1, value = np.nan, inplace = True )
data['Experience'].replace( to_replace= -2,value = np.nan,inplace = True )
data['Experience'].replace( to_replace= -3, value = np.nan, inplace = True )
data['Experience'].fillna(data['Experience'].median(),inplace=True)
data.describe()
# Removing the 'ID' and 'Experience' column as it won't required for determining accuracy of training
and testing dataset because
# 'ID' is randomly generated number given to the customer and 'experience' is also not related with
the 'Personal Loan'.
data=data.drop(['ID','Experience'],axis=1)
# Checking the first 5 rows
# by default the head() method gives the first 5 rows but if we need more rows then simply mention
how many first rows you want.
# For example if we want first 10 rows then simply write data.head(10)
data.head()
# Checking the shape of data means no. of rows and columns.
# Because we removed the 'ID' and 'Experience' therefore there are now total 12 columns remaining
out of 14 columns.
data.shape
# Checking the last 5 rows
# by default the tail() method gives the last 5 rows but if we need more rows then simply mention
how many last rows you want.
# For example if we want last 10 rows then simply write data.tail(10)
data.tail()
# **3. EDA: Study the data distribution in each attribute and target variable, share your findings.**
```

```
#.columns method is used to return the column labels of our data.
d_c=data.columns
# Applying 'for' loop to calculate no. of uniques in each columns of our data.
# This is helpful for us in determining which column of our data contains numerical data or
categorical data.
for i in d_c:
 data[i].unique()
 print(i +": "+str(len(data[i].unique())))
# After calculating the no. of uniques in each column by above method, I found another method
which does the same work as we did in above cell by applying for loop.
# This method is nunique() which gives us no. of unique in each column just by writing one line of
code.
data.nunique()
##Number of people with zero mortgage.
# Calculating the no. of people with zero mortgage by using list and by applying for loop.
d=list(data['Mortgage'])
count=0
for i in d:
if i==0:
  count=count+1
print(count)
##Number of people with zero credit card spending per month.
# Calculating the no. of people with zero credit card spending per month by using list and by applying
for loop.
c=list(data['CCAvg'])
count1=0
for i in c:
 if i==0:
  count1=count1+1
print(count1)
##Value counts of all categorical columns.
```

##Number of unique in each column.

```
# Calculating value counts of all categorical coilumns by applying .value_counts() method.
print("Value count of Family : " + "\n" + str(data['Family'].value_counts()))
print("\n")
print("Value count of Education : " + "\n" + str(data['Education'].value_counts()))
print("\n")
print("Value count of Personal Loan: " + "\n" + str(data['Personal Loan'].value_counts()))
print("\n")
print("Value count of Securities Account: " + "\n" + str(data['Securities Account'].value_counts()))
print("\n")
print("Value count of CD Account: " + "\n" + str(data['CD Account'].value_counts()))
print("\n")
print("Value count of Online : " + "\n" + str(data['Online'].value_counts()))
print("\n")
print("Value count of CreditCard : " + "\n" + str(data['CreditCard'].value_counts()))
##Univariate and Bivariate analysis
***Univariate Analysis***
sns.distplot(data['Age'])
sns.distplot(data['Income'])
sns.distplot(data['CCAvg'])
sns.distplot(data['Mortgage'])
sns.countplot(x='CreditCard',data=data)
sns.countplot(x='Family',data=data)
sns.countplot(x='Securities Account',data=data)
sns.countplot(x='CD Account',data=data)
sns.countplot(x='Education',data=data)
sns.countplot(x='Online',data=data)
sns.countplot(x='Personal Loan',data=data)
***Bivariate and Multivariate Analysis***
sns.set_style('whitegrid')
sns.pairplot(data)
plt.show()
```

```
sns.FacetGrid(data,hue="Personal Loan",size=5).map(plt.scatter,"Education","Income").add_legend()
plt.show()
sns.countplot(x='Securities Account',data=data,hue='Personal Loan')
sns.countplot(x='CreditCard',data=data,hue='Personal Loan')
sns.countplot(x='Family',data=data,hue='Personal Loan')
sns.countplot(x='Online',data=data,hue='Personal Loan')
sns.countplot(x='CD Account',data=data,hue='Personal Loan')
# **4. Apply necessary transformations for the feature variables.**
# here I applied Boxcox transformation for the feature variables
# function to plot a histogram and Q-Q plot side by side , for a certain variable
def transform_plot(data_tr, variable):
 plt.figure(figsize=(15,6))
 plt.subplot(1,2,1)
 data_tr[variable].hist()
 plt.subplot(1,2,2)
 stats.probplot(data_tr[variable], dist="norm", plot=plt)
 plt.show()
# Boxcox transformation for 'income'
data['Income_boxcox'], param = stats.boxcox(data.Income+1)
print('Parameters/optimal lambda: ', param)
transform_plot(data,'Income_boxcox')
sns.distplot(data['Income_boxcox'])
# here by seeing Q-Q plot, I come to know that I need to apply Logistic Regression in further steps
# Boxcox transformation for 'CCAvg'
data['CCAvg_boxcox'], param = stats.boxcox(data.CCAvg+1)
print('Parameters/optimal lambda: ', param)
```

```
transform_plot(data,'CCAvg_boxcox')
sns.distplot(data['CCAvg_boxcox'])
# here by seeing Q-Q plot, I come to know that I need to apply Logistic Regression in further steps
# I also applied the Boxcox transformation for mortgage but I didn't got expected result
# That's why I choose to go with another transformation
#data['Mortgage_boxcox'], param = stats.boxcox(data_x.Mortgage+1)
#print('Parameters/optimal lambda: ', param)
#transform_plot(data,'Mortgage_boxcox')
#sns.distplot(data['Mortgage_boxcox'])
# In this cell, I have applied Binning tranformation for Mortgage
data['Mortgage_Int']=pd.cut(data['Mortgage'],bins=[0,100,200,300,400,500,600,700],labels=[0,1,2,3,
4,5,6],include_lowest=True)
data.drop(['Mortgage'],axis=1,inplace=True)
sns.distplot(data['Mortgage_Int'])
# defining the data_x and data_y for training and testing purpose
data x = data.loc[:,data.columns != "Personal Loan"]
data y = data[['Personal Loan']]
# removing the 'Income', 'ZIP Code', 'CCAvg' columns from the data x as it is not required for further
steps
data_x=data_x.drop(['Income','ZIP Code','CCAvg'],axis=1)
data_x["Mortgage_Int"]=data_x["Mortgage_Int"].astype('int64')
# checking datatypes of each cloumns in data_x
data_x.dtypes
#data_x.head()
# **5. Normalise your data and split the data into training and test set in the ratio of 70:30
respectively.**
# splitting of data into training set and testing set in the ratio 70:30
# 70% training set is used to train the model and 30% testing set id used to test the model
```

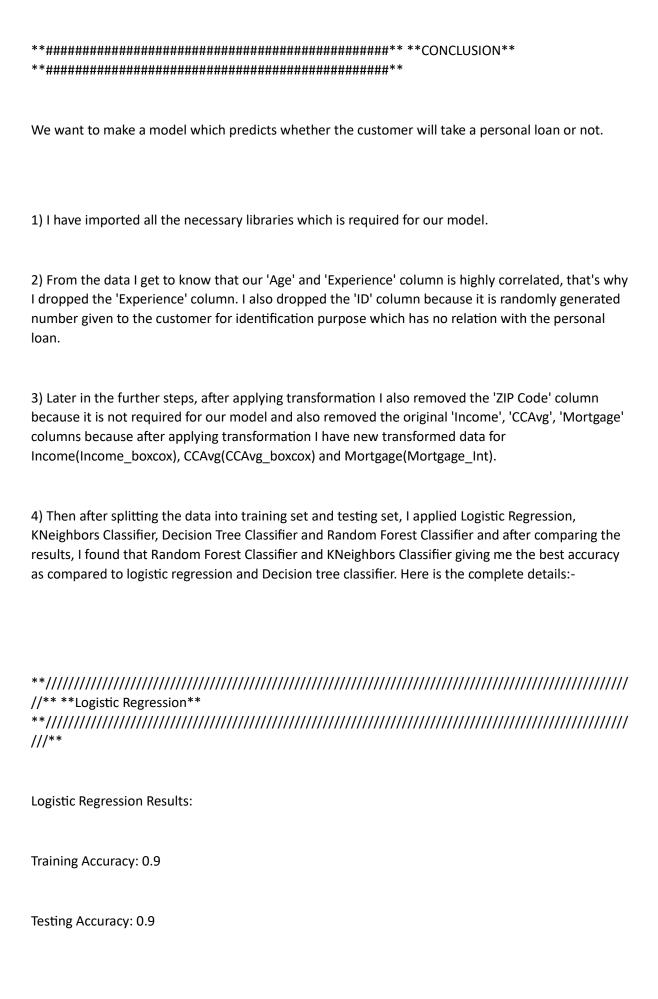
```
train_X, test_X, train_Y, test_Y = train_test_split(data_x, data_y, test_size = 0.3, stratify = data_y,
random state=0)
# printing the shape of train X, test X, train Y, test Y
print("Shape of train X: ",train X.shape)
print("Shape of test X: ",test X.shape,"\n")
print("Shape of train Y: ",train Y.shape)
print("Shape of test_Y: ",test_Y.shape)
# here applying 'I2' normalisation for train_X, test_X, train_Y, test_Y
from sklearn import preprocessing
train_X = preprocessing.normalize(train_X,norm='l2')
test_X = preprocessing.normalize(test_X,norm='l2')
train_Y = preprocessing.normalize(train_Y,norm='l2')
test_Y = preprocessing.normalize(test_Y,norm='l2')
"""print(train X)
print(test_X)
print(train_Y)
print(test Y)"""
# here I comment the standard scaling technique because I just want to compare above '12'
normalisation technique with standard scaling technique
# I got good result with 'I2' normalisation technique thats why I choose to go with 'I2' normalisation
technique instead of going with StandardScaler technique
"""from sklearn.preprocessing import StandardScaler
scaling= StandardScaler()
scaling.fit_transform(train_X)
scaling.fit_transform(test_X)
scaling.fit_transform(train_Y)
scaling.fit transform(test Y)"""
# **6. Using the Logistic Regression model to predict the likelihood of a customer buying personal
loans.**
#**7. Printing all the metrics related for evaluating the model performance.**
#**8. Building various other classification algorithms and comparing their performance.**
```

```
# importing necessary libraries
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import precision_score, recall_score
from sklearn.metrics import plot_confusion_matrix, plot_roc_curve, plot_precision_recall_curve
from sklearn.metrics import classification_report
# Fitting logistic regression to Training Set
class_names = ['wont take loan', 'take loan']
log_reg = LogisticRegression(C=1.0, max_iter=200)
log_reg.fit(train_X,train_Y)
# printing the results of Logistic regression
# printing the training accuracy, testing accuracy, precision value, recall value
# plotting the confusion matrix, ROC curve and Precision-recall curve for the logistic regression
print('Logistic Regression Results: ')
train_score = log_reg.score(train_X,train_Y)
print('Training Accuracy:', train_score.round(2))
test_score = log_reg.score(test_X,test_Y)
print('Testing Accuracy:', test_score.round(2))
y_pred_log = log_reg.predict(test_X)
precision_logi = precision_score(test_Y, y_pred_log, labels=class_names).round(2)
print('Precision:', precision_logi)
recall_logi = recall_score(test_Y, y_pred_log).round(2)
print('Recall:', recall_logi)
plot_confusion_matrix(log_reg,test_X,test_Y, display_labels=class_names)
plt.title('Confusion Matrix for Logistic Regression')
plot_roc_curve(log_reg,test_X,test_Y)
plt.title('ROC Curve for Logistic Regression')
```

```
plot_precision_recall_curve(log_reg,test_X,test_Y)
plt.title('Precision-Recall Curve for Logistic Regression')
print(classification_report(test_Y,y_pred_log))
# Fitting KNeighbors classifier to Training Set
knn = KNeighborsClassifier(n_neighbors = 3)
knn.fit(train_X, train_Y)
# printing the results of KNeighbors classifier
# printing the training accuracy, testing accuracy, precision value, recall value
# plotting the confusion matrix, ROC curve and Precision-recall curve for the KNeighbors classifier
print('K Neighbors Classifier Results: ')
knntrain_score = knn.score(train_X,train_Y)
print('Training Accuracy:', knntrain_score.round(2))
knntest_score = knn.score(test_X,test_Y)
print('Testing Accuracy:', knntest_score.round(2))
y_pred_knn = knn.predict(test_X)
precision_knn = precision_score(test_Y, y_pred_knn, labels=class_names).round(2)
print('Precision:', precision_knn)
recall_knn = recall_score(test_Y, y_pred_knn).round(2)
print('Recall:', recall_knn)
plot_confusion_matrix(knn,test_X,test_Y, display_labels=class_names)
plt.title('Confusion Matrix for K Neighbors Classifier')
plot_roc_curve(knn,test_X,test_Y)
plt.title('ROC Curve for K Neighbors Classifier')
plot_precision_recall_curve(knn,test_X,test_Y)
```

```
plt.title('Precision-Recall for K Neighbors Classifier')
print(classification_report(test_Y,y_pred_knn))
#Fitting Decision Tree classifier to Training Set
from sklearn.tree import DecisionTreeClassifier
dt_clf= DecisionTreeClassifier()
dt_clf.fit(train_X,train_Y)
# printing the results of Decision Tree classifier
# printing the training accuracy, testing accuracy, precision value, recall value
# plotting the confusion matrix, ROC curve and Precision-recall curve for the Decision Tree classifier
print('Decision Tree Classifier Results: ')
trainscore = dt_clf.score(train_X,train_Y)
print('Training Accuracy:', trainscore.round(2))
testscore = dt_clf.score(test_X,test_Y)
print('Testing Accuracy:', testscore.round(2))
y_pred_dt = dt_clf.predict(test_X)
precision_dt = precision_score(test_Y, y_pred_dt, labels=class_names).round(2)
print('Precision:', precision_dt)
recall_dt = recall_score(test_Y, y_pred_dt).round(2)
print('Recall:', recall_dt)
plot_confusion_matrix(dt_clf,test_X,test_Y, display_labels=class_names)
plt.title('Confusion Matrix for Decision Tree Classifier')
plot_roc_curve(dt_clf,test_X,test_Y)
plt.title('ROC Curve for Decision Tree Classifier')
plot_precision_recall_curve(dt_clf,test_X,test_Y)
plt.title('Precision-Recall for Decision Tree Classifier')
```

```
print(classification_report(test_Y,y_pred_dt))
# Fitting Random Forest classifier to Training Set
rf_clf = RandomForestClassifier(n_estimators=300, max_depth=7,n_jobs=-1)
rf_clf.fit(train_X,train_Y)
# printing the results of Random Forest classifier
# printing the training accuracy, testing accuracy, precision value, recall value
# plotting the confusion matrix, ROC curve and Precision-recall curve for the Random Forest classifier
print('Random Forest Classifier Results: ')
train_score = rf_clf.score(train_X,train_Y)
print('Training Accuracy:', train_score.round(2))
test_score = rf_clf.score(test_X,test_Y)
print('Testing Accuracy:', test_score.round(2))
y_pred_rf = rf_clf.predict(test_X)
precision_rf = precision_score(test_Y, y_pred_rf, labels=class_names).round(2)
print('Precision:', precision_rf)
recall_rf = recall_score(test_Y, y_pred_rf).round(2)
print('Recall:', recall_rf)
plot_confusion_matrix(rf_clf,test_X,test_Y, display_labels=class_names)
plt.title('Confusion Matrix for Random Forest Classifier')
plot_roc_curve(rf_clf,test_X,test_Y)
plt.title('ROC Curve for Random Forest Classifier')
plot_precision_recall_curve(rf_clf,test_X,test_Y)
plt.title('Precision-Recall for Random Forest Classifier')
print(classification_report(test_Y,y_pred_rf))
#**CONCLUSION**
```



Precision: 0.0

Recall: 0.0

precision recall f1-score support

0.0 0.90 1.00 0.95 1356

1.0 0.00 0.00 0.00 144

accuracy 0.90 1500

macro avg 0.45 0.50 0.47 1500

weighted avg 0.82 0.90 0.86 1500

/ KNeighbors Classifier

K Neighbors Classifier Results:

Training Accuracy: 0.96

Testing Accuracy: 0.93

Precision: 0.69

Recall: 0.41

precision recall f1-score support

0.0 0.94 0.98 0.96 1356 accuracy 0.93 1500
macro avg 0.82 0.70 0.74 1500
weighted avg 0.92 0.93 0.92 1500

Decision Tree Classifier Results:

Training Accuracy: 1.0

Testing Accuracy: 0.95

Precision: 0.77

Recall: 0.75

precision recall f1-score support

0.0 0.97 0.98 0.97 13561.0 0.77 0.75 0.76 144

accuracy 0.95 1500
macro avg 0.87 0.86 0.87 1500
weighted avg 0.95 0.95 0.95 1500

**/////////////////////////////////////	///////////////////////////////////////
Random Forest Classifier	
///////////////////////////////////////	///////////////////////////////////////

Random Forest Classifier Results:

Training Accuracy: 0.95

Testing Accuracy: 0.94

Precision: 0.95

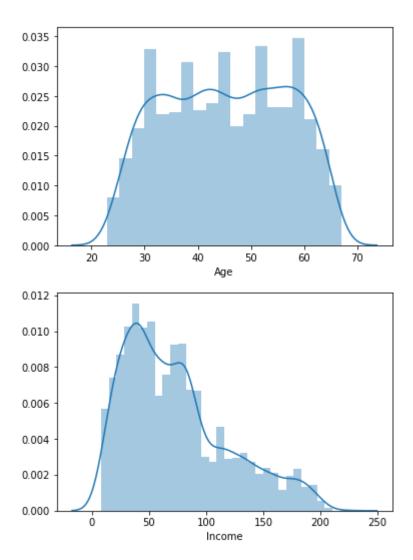
Recall: 0.4

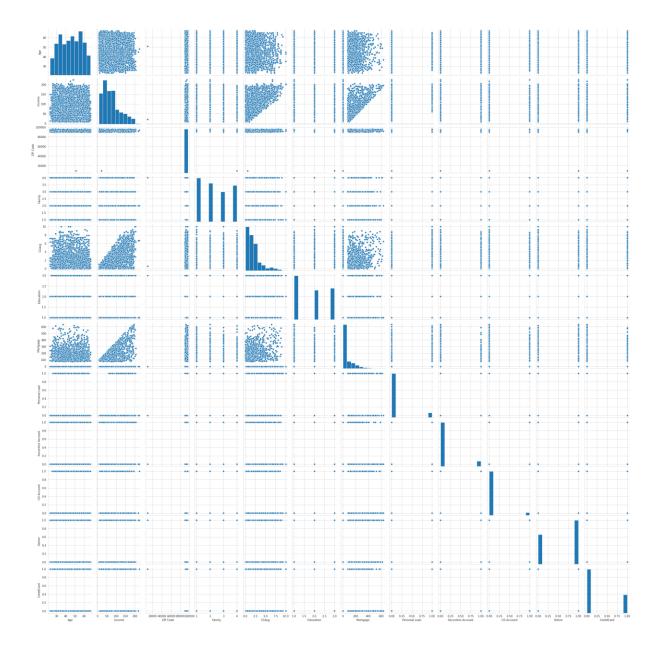
precision recall f1-score support

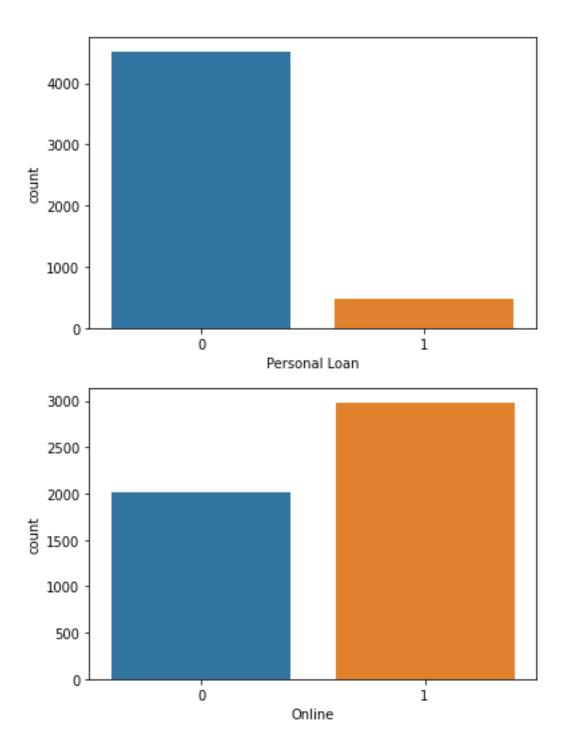
0.0 0.94 1.00 0.97 1356 1.0 0.95 0.40 0.56 144

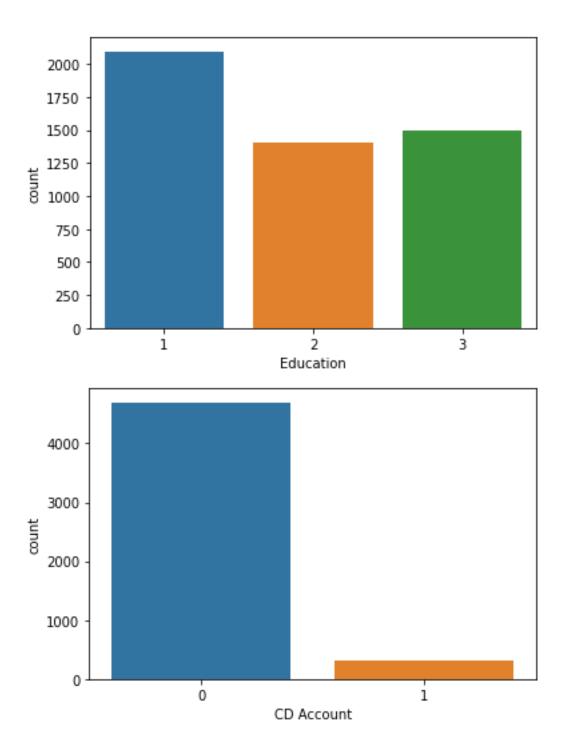
accuracy 0.94 1500
macro avg 0.94 0.70 0.76 1500
weighted avg 0.94 0.94 0.93 1500

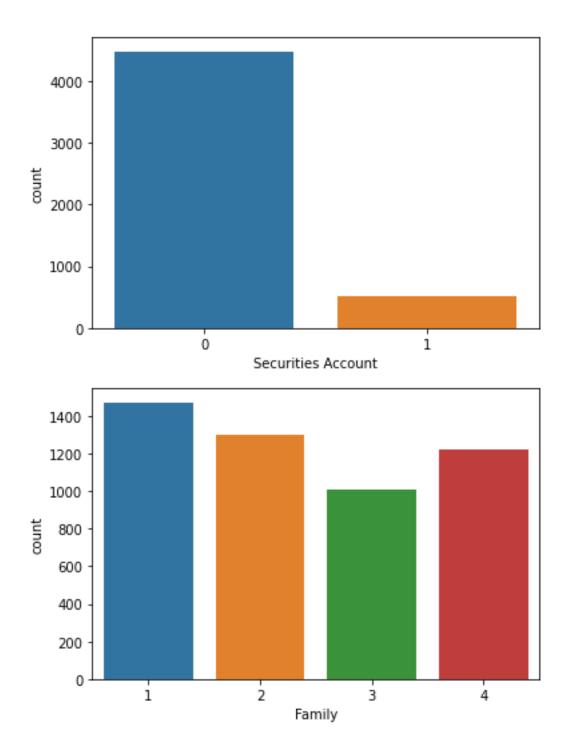
^{**##################**}

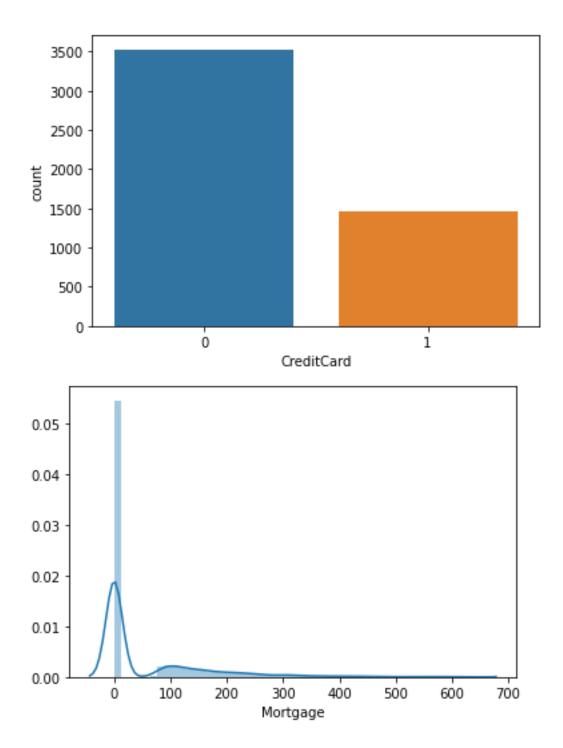


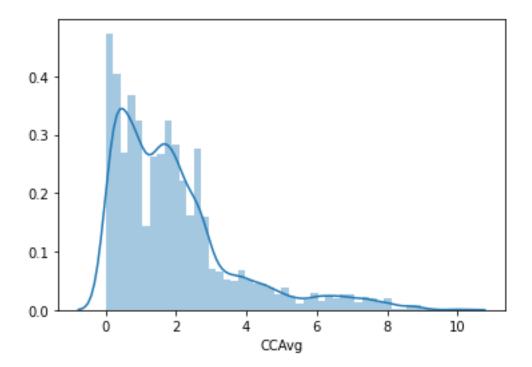


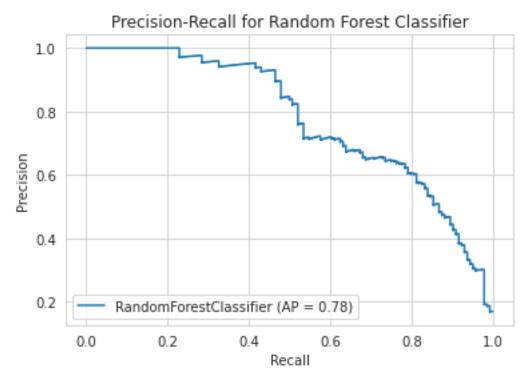


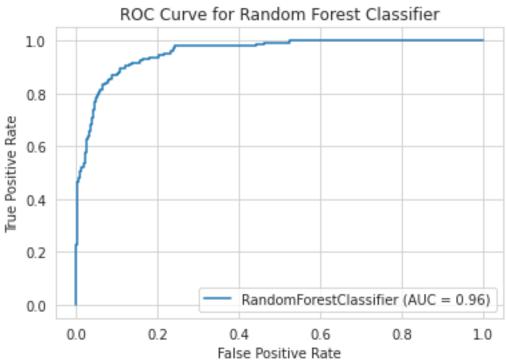


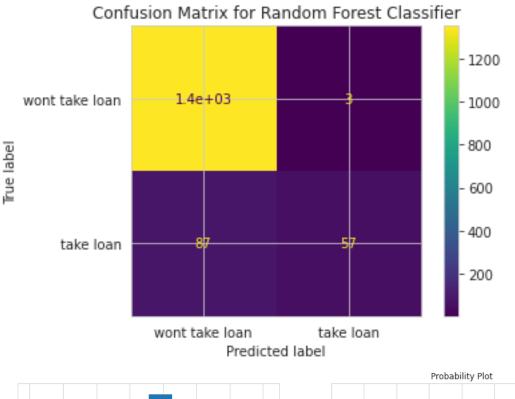


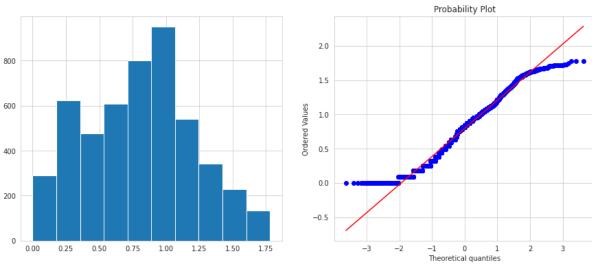


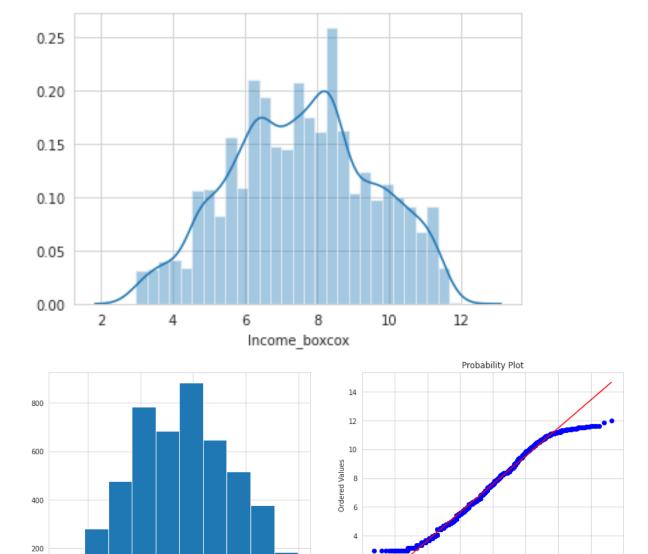












-1 0 1 Theoretical quantiles

-3

-2

