Analysis Report

Domain: Banking

Prepared By:

Tohfa Siddika Barbhuiya,

EXL Analytics

17 Jan’2021

EXECUTIVE SUMMARY:

BACKGROUND:

Context:

This case is about a bank (Thera Bank) whose management wants to explore ways of converting its liability 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. This has encouraged the retail marketing department to devise campaigns with better target marketing to increase the success ratio with minimal budget.

Approach:

The file Bank.xls 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. We will implement Classification algorithms to differentiate people who will buy loans vs the who will not.

OBJECTIVE:

To dive deep into this data to find some valuable insights.

Attribute Information:

- **ID :** Customer ID
- **Age :** Customer's age in completed years
- **Experience :** #years of professional experience
- **Income :** Annual income of the customer ($000)
- **ZIP Code :** Home Address ZIP code.
- **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 UniversalBank?

KEY FINDINGS:

```
import warnings
warnings.filterwarnings('ignore')

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
sns.set()
%matplotlib inline

from sklearn.linear_model import LogisticRegression
from sklearn import metrics
from sklearn.preprocessing import scale
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score

from scipy.stats import zscore
from sklearn import tree
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import BaggingClassifier
from sklearn.feature_extraction.text import CountVectorizer
```

```
from sklearn.naive_bayes import GaussianNB
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import BaggingClassifier, RandomForestClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn import model_selection

import os
```

Q1)1. Read the column description and ensure you understand each attribute well

```
df = pd.read_csv("C:/Users/user/Downloads/Bank_Personal_Loan_Modelling.csv")
df_copy=df.copy()

df.head()
```

OUTPUT:

	ID	Age	Experience	Income	ZIP Code	Family	CCAvg	Education	Mortgage	Personal Loan	Securities Account	CD Account	Online	CreditCard
0	1	25	1	49	91107	4	1.6	1	0	0	1	0	0	0
1	2	45	19	34	90089	3	1.5	1	0	0	1	0	0	0
2	3	39	15	11	94720	1	1.0	1	0	0	0	0	0	0
3	4	35	9	100	94112	1	2.7	2	0	0	0	0	0	0
4	5	35	8	45	91330	4	1.0	2	0	0	0	0	0	1

```
df.info()
```

OUTPUT:

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5000 entries, 0 to 4999
Data columns (total 14 columns):
 #   Column              Non-Null Count  Dtype  
---  ------              --------------  -----  
 0   ID                  5000 non-null   int64  
 1   Age                 5000 non-null   int64  
 2   Experience          5000 non-null   int64  
 3   Income              5000 non-null   int64  
 4   ZIP Code            5000 non-null   int64  
 5   Family              5000 non-null   int64  
 6   CCAvg               5000 non-null   float64
 7   Education           5000 non-null   int64  
 8   Mortgage            5000 non-null   int64  
 9   Personal Loan       5000 non-null   int64  
 10  Securities Account  5000 non-null   int64  
 11  CD Account          5000 non-null   int64  
 12  Online              5000 non-null   int64  
 13  CreditCard          5000 non-null   int64  
dtypes: float64(1), int64(13)
memory usage: 547.0 KB
```

df.isna().apply(pd.value_counts) #null value check

OUTPUT:

	ID	Age	Experience	Income	ZIP Code	Family	CCAvg	Education	Mortgage	Personal Loan	Securities Account	CD Account	Online	CreditCard
False	5000	5000	5000	5000	5000	5000	5000	5000	5000	5000	5000	5000	5000	5000

df.describe().T

OUTPUT:

	count	mean	std	min	25%	50%	75%	max
ID	5000.0	2500.500000	1443.520003	1.0	1250.75	2500.5	3750.25	5000.0
Age	5000.0	45.338400	11.463166	23.0	35.00	45.0	55.00	67.0
Experience	5000.0	20.104600	11.467954	-3.0	10.00	20.0	30.00	43.0
Income	5000.0	73.774200	46.033729	8.0	39.00	64.0	98.00	224.0
ZIP Code	5000.0	93152.503000	2121.852197	9307.0	91911.00	93437.0	94608.00	96651.0
Family	5000.0	2.396400	1.147663	1.0	1.00	2.0	3.00	4.0
CCAvg	5000.0	1.937938	1.747659	0.0	0.70	1.5	2.50	10.0
Education	5000.0	1.881000	0.839869	1.0	1.00	2.0	3.00	3.0
Mortgage	5000.0	56.498800	101.713802	0.0	0.00	0.0	101.00	635.0
Personal Loan	5000.0	0.096000	0.294621	0.0	0.00	0.0	0.00	1.0
Securities Account	5000.0	0.104400	0.305809	0.0	0.00	0.0	0.00	1.0
CD Account	5000.0	0.060400	0.238250	0.0	0.00	0.0	0.00	1.0
Online	5000.0	0.596800	0.490589	0.0	0.00	1.0	1.00	1.0
CreditCard	5000.0	0.294000	0.455637	0.0	0.00	0.0	1.00	1.0

any(df['Experience'] < 0)

exp_med = df.loc[:,"Experience"].median()

df.loc[:, 'Experience'].replace([-1, -2, -3], [exp_med, exp_med, exp_med], inplace=True)

any(df['Experience'] < 0)

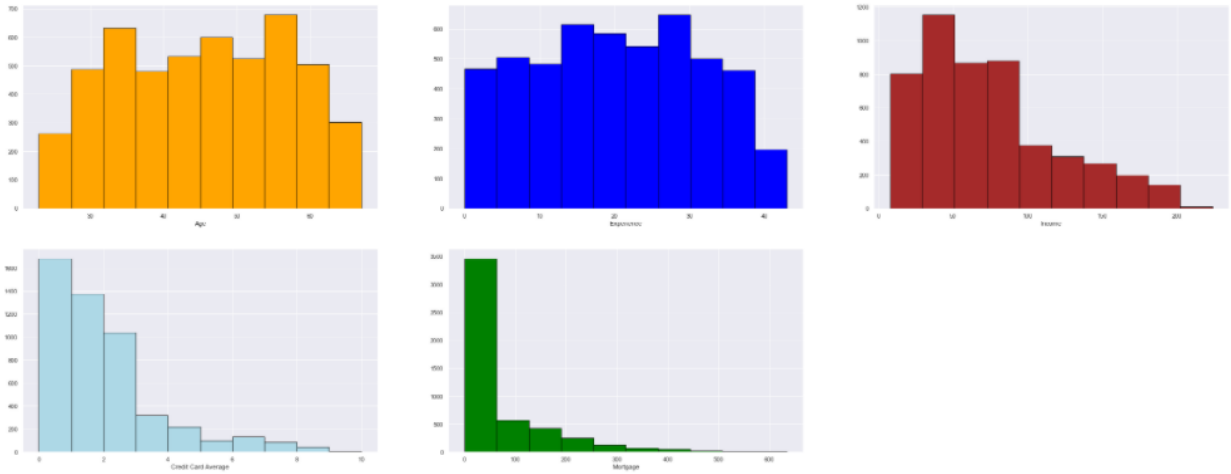
df.describe().T

Q2)Perform univariate analysis of each and every attribute - use an appropriate plot for a given attribute and mention your insights (5 points)

#Univariate Analysis of the continuous variables1

```
!pip install pandas_profiling
import pandas_profiling
df.profile_report()

plt.figure(figsize= (40.5,40.5))
plt.subplot(5,3,1)
plt.hist(df.Age, color='orange', edgecolor = 'black')
plt.xlabel('Age')

plt.subplot(5,3,2)
plt.hist(df.Experience, color='blue', edgecolor = 'black')
plt.xlabel('Experience')

plt.subplot(5,3,3)
plt.hist(df.Income, color='brown', edgecolor = 'black')
plt.xlabel('Income')

plt.subplot(5,3,4)
plt.hist(df.CCAvg, color='lightblue', edgecolor = 'black')
plt.xlabel('Credit Card Average')

plt.subplot(5,3,5)
plt.hist(df.Mortgage, color='green', edgecolor = 'black')
plt.xlabel('Mortgage')

plt.show()
```

OUTPUT:



OBSERVATION: Age and experience are distributed normally but Income Mortgage and CCAvg are highly left skewed

#Checking for Skewness of data

import statsmodels.api as sm

import scipy.stats as stats

Skewness = pd.DataFrame({'Skewness' :
[stats.skew(df.Age),stats.skew(df.Experience),stats.skew(df.Income),stats.skew(df.CCAvg)

stats.skew(df.Mortgage)]},index=['Age','Experience','Income','CCAvg','Mortgage'])

Skewness

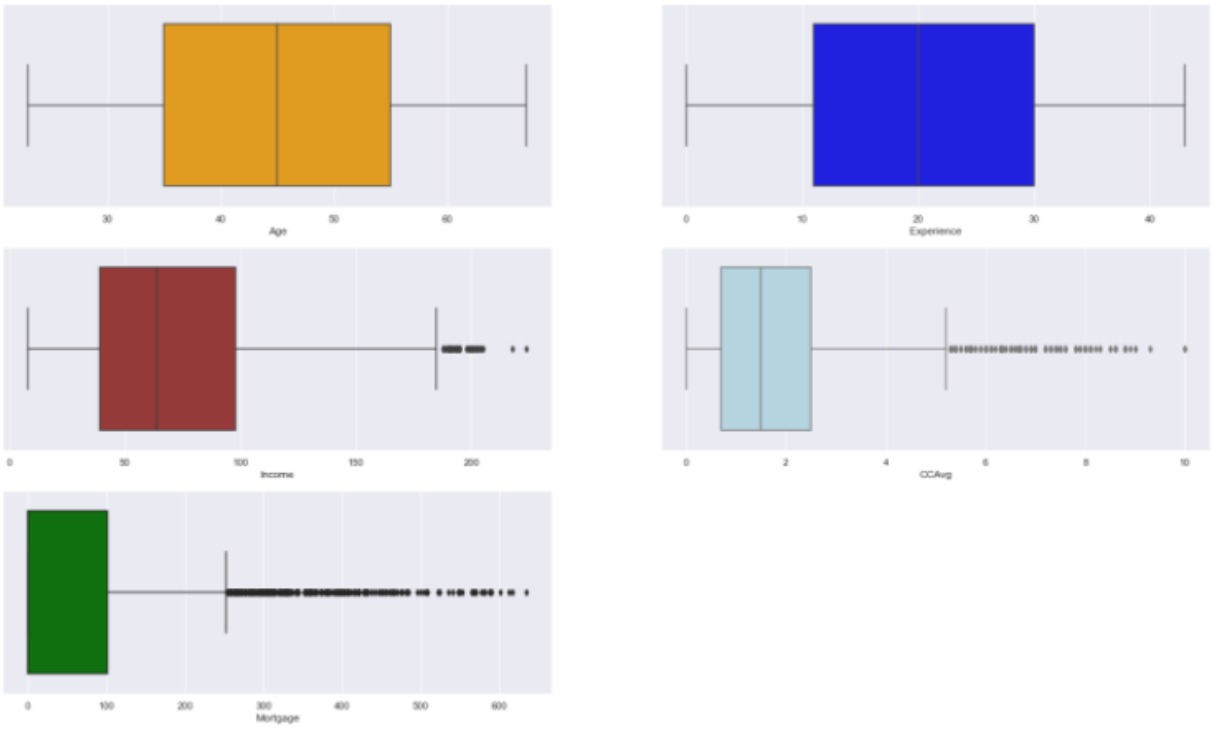
OUTPUT:

	Skewness
Age	-0.029332
Experience	-0.014096
Income	0.841086
CCAvg	1.597964
Mortgage	2.103371

#Univariate Analysis of the continuous variables2

```
plt.figure(figsize= (25,25))
plt.subplot(5,2,1)
sns.boxplot(x= df.Age, color='orange')

plt.subplot(5,2,2)
sns.boxplot(x= df.Experience, color='blue')

plt.subplot(5,2,3)
sns.boxplot(x= df.Income, color='brown')

plt.subplot(5,2,4)
sns.boxplot(x= df.CCAvg, color='lightblue')

plt.subplot(5,2,5)
sns.boxplot(x= df.Mortgage, color='green')
```

OUTPUT:



OBSERVATION: Age is normalised with majority between 35 to 55 years

Experience is also normalised with majority between 11years to 30 yeaars

Income is left skewed and is between 45K to 55K

CCAvg and Mortgagae are again left skewed

#Univariate Analysis of the categorical variables

```
plt.figure(figsize=(30,45))


plt.subplot(6,2,1)
df['Family'].value_counts().plot(kind="bar", align='center',color = 'blue',edgecolor = 'black')
plt.xlabel("Number of Family Members")
plt.ylabel("Count")
plt.title("Family Members Distribution")
```

```
plt.subplot(6,2,2)
df['Education'].value_counts().plot(kind="bar", align='center',color = 'pink',edgecolor = 'black')
plt.xlabel('Level of Education')
plt.ylabel('Count ')
plt.title('Education Distribution')


plt.subplot(6,2,3)
df['Securities Account'].value_counts().plot(kind="bar", align='center',color = 'violet',edgecolor = 'black')
plt.xlabel('Holding Securities Account')
plt.ylabel('Count')
plt.title('Securities Account Distribution')


plt.subplot(6,2,4)
df['CD Account'].value_counts().plot(kind="bar", align='center',color = 'yellow',edgecolor = 'black')
plt.xlabel('Holding CD Account')
plt.ylabel('Count')
plt.title("CD Account Distribution")


plt.subplot(6,2,5)
df['Online'].value_counts().plot(kind="bar", align='center',color = 'green',edgecolor = 'black')
plt.xlabel('Accessing Online Banking Facilities')
plt.ylabel('Count')
```

plt.title("Online Banking Distribution")

plt.subplot(6,2,6)

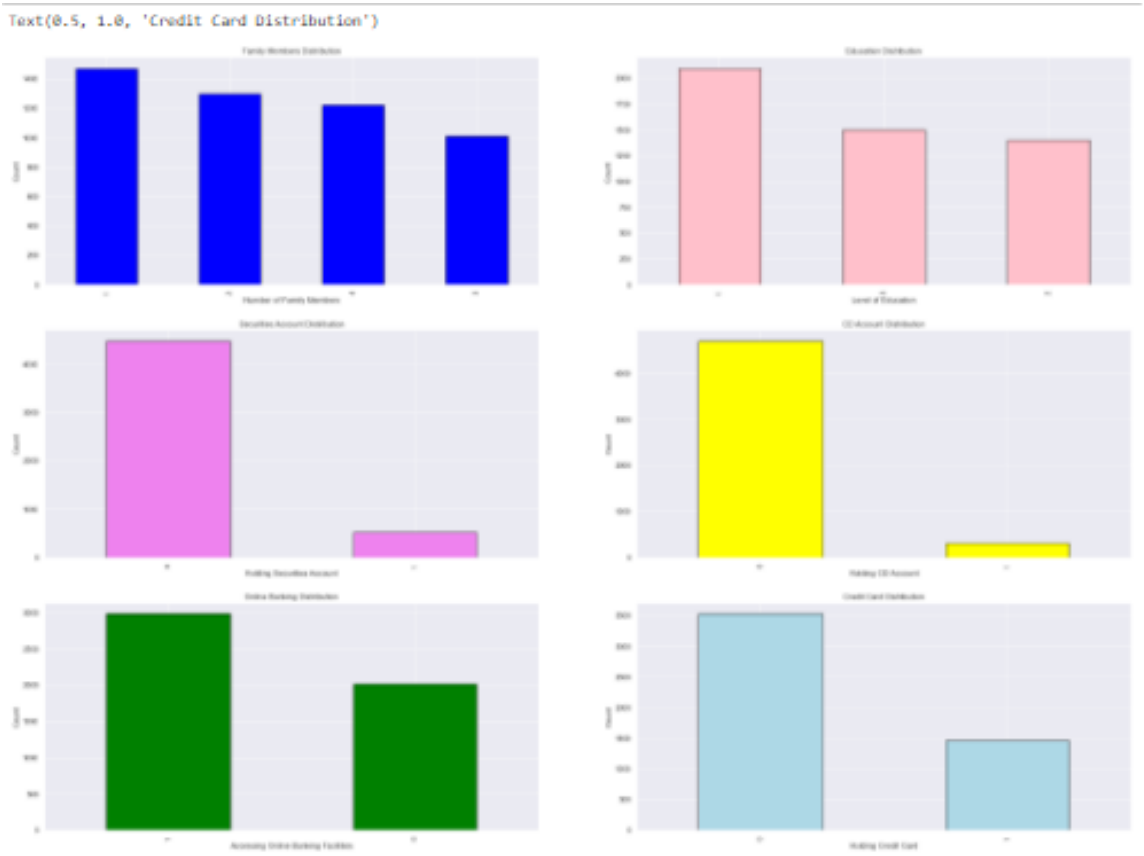
df['CreditCard'].value_counts().plot(kind="bar", align='center',color = 'lightblue',edgecolor = 'black')

plt.xlabel('Holding Credit Card')

plt.ylabel('Count')

plt.title("Credit Card Distribution")

OUTPUT:



OBSERVATION: Most of them are not having security and CD account

Q3)Perform correlation analysis among all the variables - you can use Pairplot and Correlation coefficients of every attribute with every other attribute (5 points)

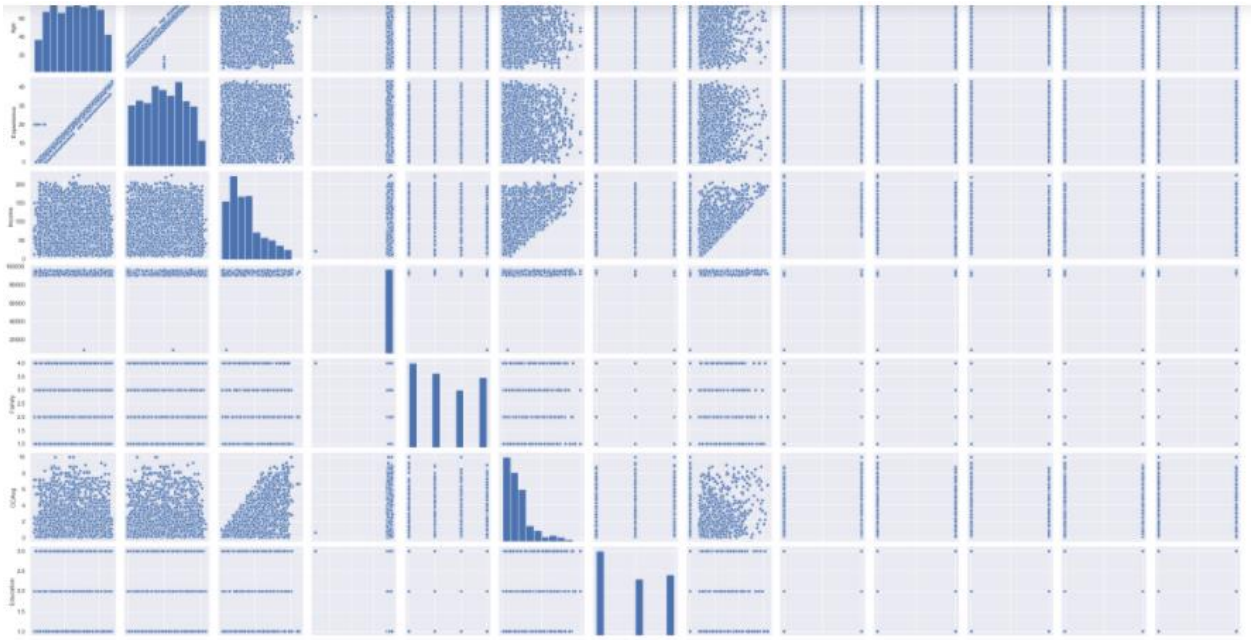
#Checking for correlation

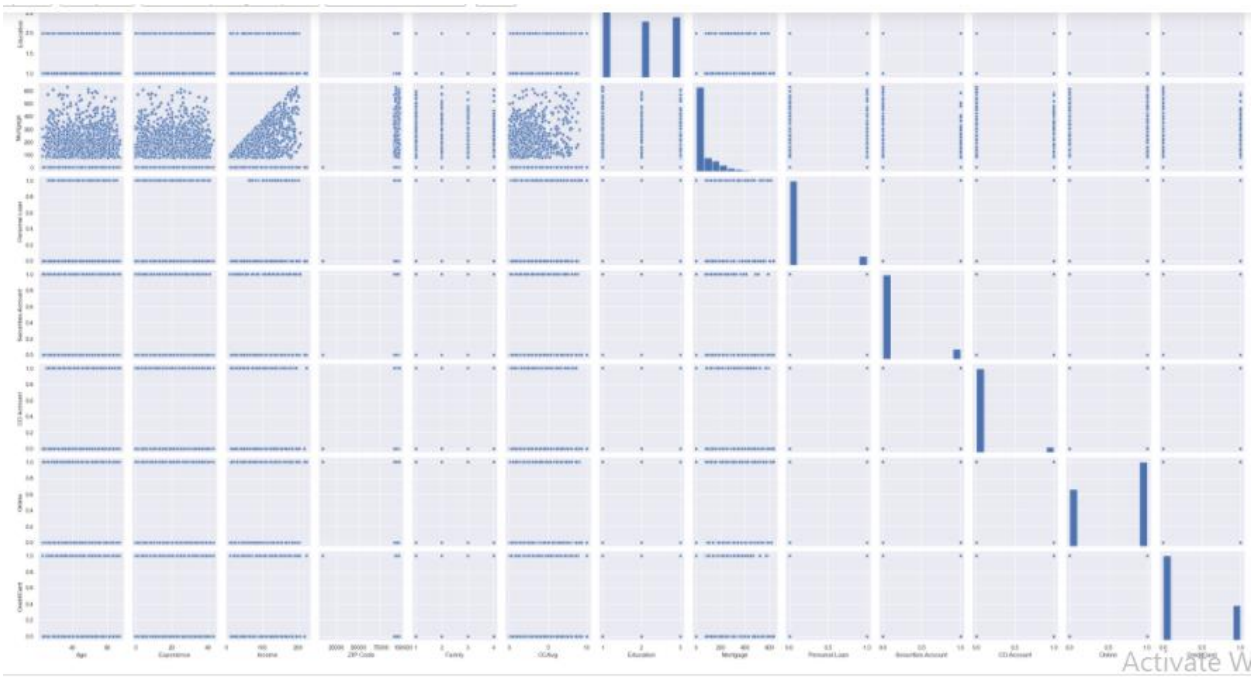
df[['Personal Loan', 'Age', 'Income', 'CCAvg', 'Mortgage']].corr()

#Pairplot

sns.pairplot(df.iloc[:,1:])

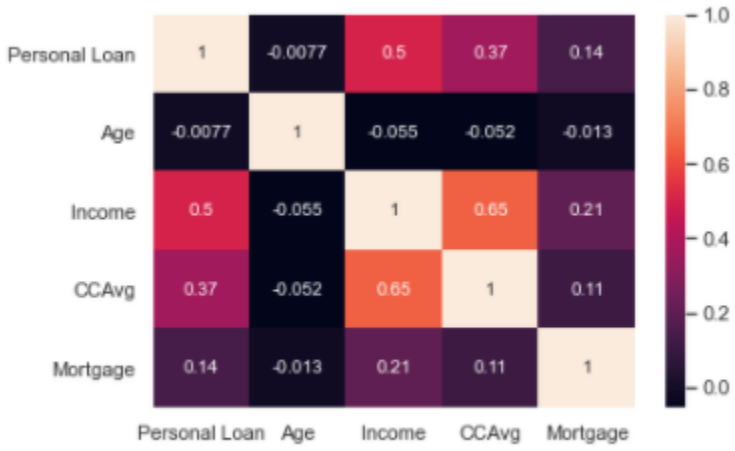
OUTPUT:





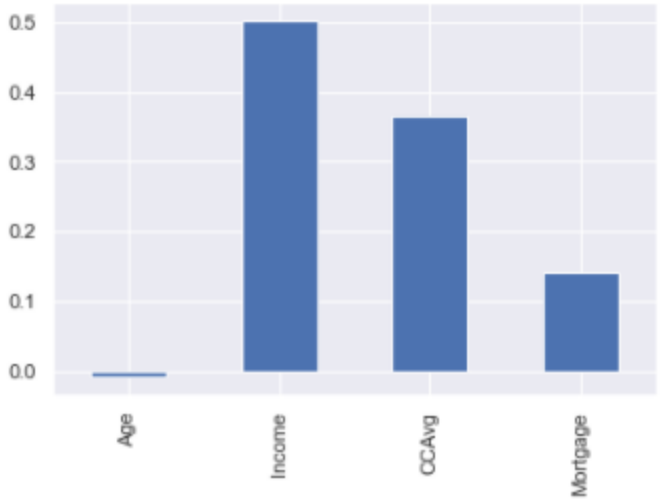
sns.heatmap(df[['Personal Loan', 'Age', 'Income', 'CCAvg', 'Mortgage']].corr(), annot = True)

OUTPUT:



df[['Personal Loan', 'Age', 'Income', 'CCAvg', 'Mortgage']].corr()['Personal Loan'][1:].plot.bar()

OUTPUT:



OBSERVATION: Income and CCAvg have some correlation with the Personal Loan, Age has hardly any correlation.

Q4). One hot encode the Education variable (3 points)

df_dummies= pd.get_dummies(df, prefix='Edu', columns=['Education']) #This function does One-Hot-Encoding on categorical text

returns the names of all the columns as a list

df_dummies.head()

OUTPUT:

```
Index(['ID', 'Age', 'Experience', 'Income', 'ZIP Code', 'Family', 'CCAvg',
       'Mortgage', 'Personal Loan', 'Securities Account', 'CD Account',
       'Online', 'CreditCard', 'Edu_1', 'Edu_2', 'Edu_3'],
      dtype='object')
```

df_dummies.columns

OUTPUT:

	ID	Age	Experience	Income	ZIP Code	Family	CCAvg	Mortgage	Personal Loan	Securities Account	CD Account	Online	CreditCard	Edu_1	Edu_2	Edu_3
0	1	25	1.0	49	91107	4	1.6	0	0	1	0	0	0	1	0	0
1	2	45	19.0	34	90089	3	1.5	0	0	1	0	0	0	1	0	0
2	3	39	15.0	11	94720	1	1.0	0	0	0	0	0	0	1	0	0
3	4	35	9.0	100	94112	1	2.7	0	0	0	0	0	0	0	1	0
4	5	35	8.0	45	91330	4	1.0	0	0	0	0	0	1	0	1	0

Q5). Separate the data into dependant and independent variables and create training and test sets out of them (X_train, y_train, X_test, y_test) (2 points)

#Dependant variable analysis

```
df["Personal Loan"].value_counts().to_frame()
```

```
pd.value_counts(df["Personal Loan"]).plot(kind="bar")
```

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

```
plt.subplot(3,1,1)
sns.scatterplot(df.CCAvg, df.Income, hue = df['Personal Loan'], palette= ['red','green'])
```

```
plt.subplot(3,1,2)
sns.scatterplot(df.Family, df.Income, hue = df['Personal Loan'], palette= ['violet','purple'])
```

```
plt.subplot(3,1,3)
sns.scatterplot(df.Income, df.Mortgage, hue = df['Personal Loan'], palette= ['yellow','green'])
```

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

```
plt.subplot(3,1,1)
sns.scatterplot(df.Age, df.Experience, hue = df['Personal Loan'], palette= ['yellow','violet'])
```

```
plt.subplot(3,1,2)
sns.scatterplot(df.Education, df.Income, hue = df['Personal Loan'], palette= ['blue','yellow'])

plt.subplot(3,1,3)
sns.scatterplot(df.Education, df.Mortgage, hue = df['Personal Loan'], palette= ['red','yellow'])



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

plt.subplot(2,2,1)
sns.countplot(x="Securities Account", data=df ,hue="Personal Loan")

plt.subplot(2,2,2)
sns.countplot(x='CD Account' ,data=df ,hue='Personal Loan')

sns.distplot(df[df["Personal Loan"] == 0]['Income'], color = 'b')
sns.distplot(df[df["Personal Loan"] == 1]['Income'], color = 'y')

#creating training and test sets out of them(X_train, y_train, X_test, y_test)

from sklearn.model_selection import train_test_split
train_set, test_set = train_test_split(df.drop(['ID','Experience'], axis=1), test_size=0.3 ,
random_state=100)



train_labels = train_set.pop('Personal Loan')
test_labels = test_set.pop('Personal Loan')
```

Q6) Use StandardScaler( ) from sklearn, to transform the training and test data into scaled values (fit the StandardScaler object to the train data and transform train and test da

```
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()

print(scaler.fit(df))

StandardScaler()

print(scaler.mean_)


print(scaler.transform(df))
```

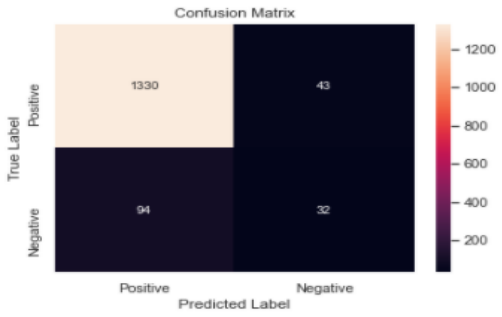
Q7)Write a function which takes a model, X_train, X_test, y_train and y_test as input and returns the accuracy,recall, precision, specificity, f1_score of the model trained on the train set and evaluated on the test set (5points)

```
from sklearn.datasets import make_classification

from sklearn.cross_validation import StratifiedShuffleSplit

from sklearn.metrics import accuracy_score, f1_score, precision_score, recall_score,
classification_report, confusion_matrix


# We use a utility to generate artificial classification data.

X, y = make_classification(n_samples=100, n_informative=10, n_classes=3)

sss = StratifiedShuffleSplit(y, n_iter=1, test_size=0.5, random_state=0)

for train_idx, test_idx in sss:

    X_train, X_test, y_train, y_test = X[train_idx], X[test_idx], y[train_idx], y[test_idx]

    svc.fit(X_train, y_train)

    y_pred = svc.predict(X_test)

    print(f1_score(y_test, y_pred, average="macro"))

    print(precision_score(y_test, y_pred, average="macro"))

    print(recall_score(y_test, y_pred, average="macro"))
```

Q8). Employ multiple Classification models (Logistic, K-NN, Naïve Bayes etc) and use the function from step 7 to train and get the metrics of the model (15 points)

```
logmodel = LogisticRegression()
logmodel.fit(X_Train,Y_Train)

predict = logmodel.predict(X_Test)
predictProb = logmodel.predict_proba(X_Test)


# Confusion Matrix
cm = confusion_matrix(Y_Test, predict)

class_label = ["Positive", "Negative"]
df_cm = pd.DataFrame(cm, index = class_label, columns = class_label)
sns.heatmap(df_cm, annot = True, fmt = "d")
plt.title("Confusion Matrix")
plt.xlabel("Predicted Label")
plt.ylabel("True Label")
plt.show()# Confusion Matrix
```

OUTPUT:



Classification Report

print(classification_report(Y_Test, predict))

OUTPUT:

	precision	recall	f1-score	support
0	0.93	0.97	0.95	1373
1	0.43	0.25	0.32	126
accuracy			0.91	1499
macro avg	0.68	0.61	0.63	1499
weighted avg	0.89	0.91	0.90	1499

Creating odd list of K for KNN

myList = list(range(1,20))

Subsetting just the odd ones

neighbors = list(filter(lambda x: x % 2 != 0, myList))

Empty list that will hold accuracy scores

ac_scores = []

Perform accuracy metrics for values from 1,3,5....19

for k in neighbors:

knn = KNeighborsClassifier(n_neighbors=k)

knn.fit(X_Train, Y_Train)

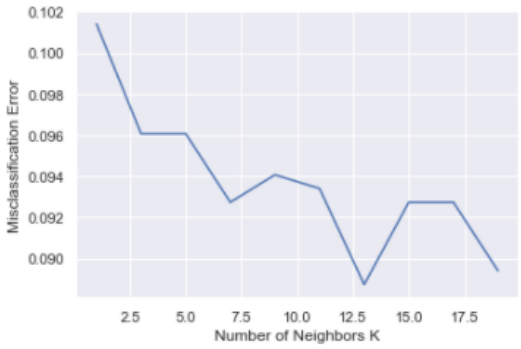
Predict the response

Y_Pred = knn.predict(X_Test)

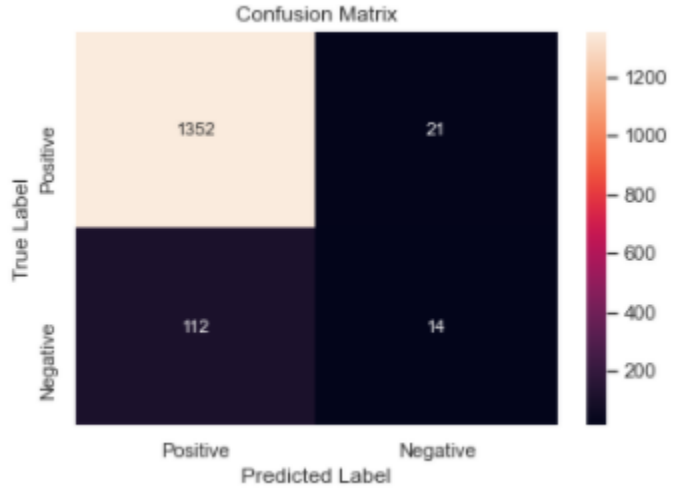
Evaluate accuracy

```
    scores = accuracy_score(Y_Test, Y_Pred)
    ac_scores.append(scores)

# Changing to misclassification error
MSE = [1 - x for x in ac_scores]

# Determining best k
optimal_k = neighbors[MSE.index(min(MSE))]
print("The optimal number of neighbors is %d" % optimal_k)



knn = KNeighborsClassifier(n_neighbors= 13 , weights = 'uniform', metric = 'euclidean')
knn.fit(X_Train, Y_Train)
predicted = knn.predict(X_Test)
from sklearn.metrics import accuracy_score
acc = accuracy_score(Y_Test, predicted)
print(acc)

plt.plot(neighbors, MSE)
plt.xlabel('Number of Neighbors K')
plt.ylabel('Misclassification Error')
```

OUTPUT:



OUTPUT:



Classification Report

print(classification_report(Y_Test, predicted))

	precision	recall	f1-score	support
0	0.92	0.98	0.95	1373
1	0.40	0.11	0.17	126
accuracy			0.91	1499
macro avg	0.66	0.55	0.56	1499
weighted avg	0.88	0.91	0.89	1499

#Naive Bayes

Model

naive_model = GaussianNB()

naive_model.fit(train_set, train_labels)

prediction = naive_model.predict(test_set)

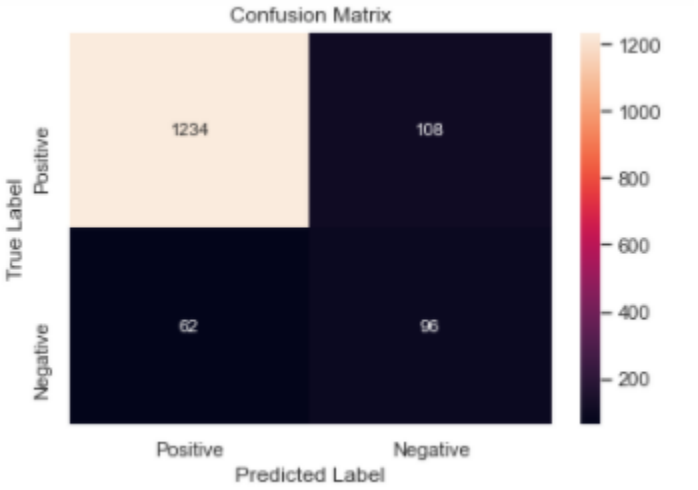
naive_model.score(test_set,test_labels)

Confusion Matrix

cm2 = confusion_matrix(test_labels, prediction)

```
class_label = ["Positive", "Negative"]
df_cm2 = pd.DataFrame(cm2, index = class_label, columns = class_label)
sns.heatmap(df_cm2, annot = True, fmt = "d")
plt.title("Confusion Matrix")
plt.xlabel("Predicted Label")
plt.ylabel("True Label")
plt.show()
```

OUTPUT:



Classififcation Report

```
print(classification_report(test_labels, prediction))
```

OUTPUT:

	precision	recall	f1-score	support
0	0.95	0.92	0.94	1342
1	0.47	0.61	0.53	158
accuracy			0.89	1500
macro avg	0.71	0.76	0.73	1500
weighted avg	0.90	0.89	0.89	1500

Q9)Create a dataframe with the columns - “Model”, “accuracy”, “recall”, “precision”, “specificity”, “f1_score”. Populate the dataframe accordingly (5 points)

```
LOG_reg=classification_report(Y_Test, predict)
print(classification_report(Y_Test, predict))
```

```
KNN=classification_report(Y_Test, predict)
print(classification_report(Y_Test, predicted))

Naive=classification_report(Y_Test, predict)
print(classification_report(test_labels, prediction))
```

OUTPUT:

```
LOG_reg=classification_report(Y_Test, predict)
print(classification_report(Y_Test, predict))

KNN=classification_report(Y_Test, predict)
print(classification_report(Y_Test, predicted))

Naive=classification_report(Y_Test, predict)
print(classification_report(test_labels, prediction))
```

Q10)Give your reasoning on which is the best model in this case (5 points)

```
models = []
models.append(('KNN', KNeighborsClassifier()))
models.append(('LR', LogisticRegression()))
models.append(('NB', GaussianNB()))

# Evaluate each model in turn
results = []
names = []
scoring = 'accuracy'
for name, model in models:
```

```
    kfold = model_selection.KFold(n_splits=10, random_state=12345)
    cv_results = model_selection.cross_val_score(model, X, Y, cv=kfold, scoring=scoring)
    results.append(cv_results)
    names.append(name)
    msg = "%s: %f (%f)" % (name, cv_results.mean(), cv_results.std())
    print(msg)

# Boxplot algorithm comparison
fig = plt.figure()
fig.suptitle('Algorithm Comparison')
ax = fig.add_subplot(111)
plt.boxplot(results)
ax.set_xticklabels(names)
plt.show()
```

Logistic Regression is the best Model as its accuracy is highest and recall is also good

In KNN the accuracy and recall is good but the confusion matrix tells it is not the accurate model for prediction

Naive Bayes has less accuracy and recall compared to other models

CONCLUSION:

Following can be concluded from the univariate analysis, the plots, skewness,model and also from the whole case study analysis:

1. The customer who having a CD account with the bank is more likely to buy personal loans from the bank**.**
2. The customer who uses or doesn't use a credit card issued is not affecting the purchase of personal loan.

3. The customer who uses or doesn’t use internet banking facilities is likely to not affect the personal loans purchase.
4. The customers who have or don’t have securities account with the bank is likely to not affect the chances of buying a personal loan.
5. Age of the customer is not affect the probability of buying the personal loan
6. Customers applicants who spend more on credit cards are seems to be more prone to buy personal loans.
7. Applicants with high incomes have more chances of buying the personal loan