

SEPTEMBER 2, 2018



Group Mini Project | Supervised Learning

Assessment

*Presented By: Group5 - Sankarsan Gayen, Karthik
N, Sasirekha Sathasivam, Satish Patil*

PGPBDML.B. Mar18


Case Study-

Campaign for selling personal loans

This case is about a bank (Thera Bank) which has a growing customer base. Majority of these customers are liability customers (depositors) with varying size of deposits. The number of customers who are also borrowers (asset customers) is quite small, and the bank is interested in expanding this base rapidly to bring in more loan business and in the process, earn more through the interest on loans. In particular, the 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.

The department wants to build a model that will help them identify the potential customers who have higher probability of purchasing the loan. This will increase the success ratio while at the same time reduce the cost of the campaign.

The file given below 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.

Data Set - [Bank Personal Loan Modelling.xlsx](#) 

Considering the information provided above, follow the steps given below --

1. Read the column description and ensure you understand each attribute well.
2. Study the data distribution in each attribute, share your findings.
3. Get the target column distribution. Your comments.
4. Split the data into training and test set in the ratio of 70:30 respectively.
5. Use a classification model to predict the likelihood of a liability customer buying personal loans.
6. Explain why you chose one model over the other

Solutions:

Predicting the personal loan borrowers for Thera Bank

#1. Read the column description and ensure you understand each attribute well:

- The Dataset has 13 columns 5000 rows, its a good representation of population:

Out of 13 Columns, "Personal Loan" is the target variable and remaining 12 are independent attributes
:

1. Age, Experience, Income, Zip Code, Family and Education: These columns represent customer demographic information.
2. CCAvg, Mortgage, Personal Loan, Securities Account, CD Account, Online, Creditcard: These columns represent customers relationship with bank.
3. Categorical Variables: The columns Family, Education, Personal Loan, Securities Account, CD Account, Online and CreditCard are categorical variables.
4. Experience: The variable Experience is having Negative values.
5. Income, CCAvg and Mortgage: These three columns have skewed distribution and there might be few outliers.

```
# Column descriptions

##      ID          Customer ID
##      Age         Customer's age in completed years

##      Experience   #years of professional experience

##      Income       Annual income of the customer ($000)

##      ZIPCode      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?

##      CreditCard    Does the customer use a credit card issued by UniversalBank?
```

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5000 entries, 0 to 4999
Data columns (total 13 columns):
Age                5000 non-null int64
Experience          5000 non-null int64
Income             5000 non-null int64
ZIP Code           5000 non-null int64
Family             5000 non-null int64
CCAvg              5000 non-null float64
Education          5000 non-null int64
Mortgage           5000 non-null int64
Personal Loan      5000 non-null int64
Securities Account 5000 non-null int64
CD Account         5000 non-null int64
Online             5000 non-null int64
CreditCard         5000 non-null int64
dtypes: float64(1), int64(12)
memory usage: 507.9 KB
```

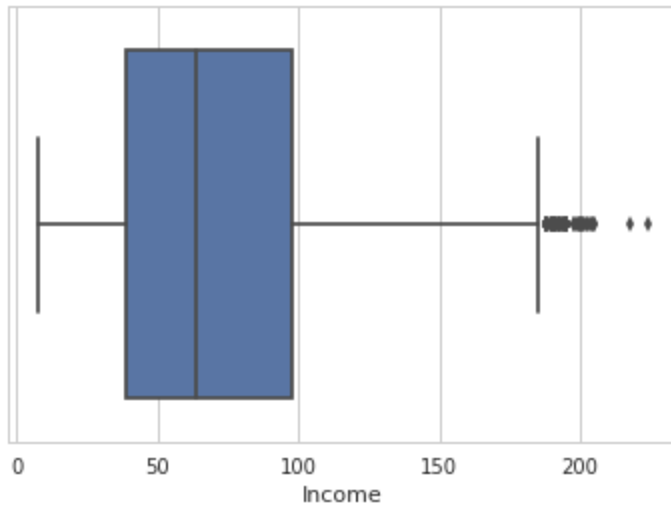
```
data.describe()
```

	Age	Experience	Income	ZIP Code	Family	CCAvg	Education	Mortgage	Personal Loan	Securities Account	CD Account	
count	5000.000000	5000.000000	5000.000000	5000.000000	5000.000000	5000.000000	5000.000000	5000.000000	5000.000000	5000.000000	5000.000000	5000
mean	45.338400	20.104600	73.774200	93152.503000	2.396400	1.937913	1.881000	56.498800	0.096000	0.104400	0.060400	0
std	11.463166	11.467954	46.033729	2121.852197	1.147663	1.747666	0.839869	101.713802	0.294621	0.305809	0.238250	0
min	23.000000	-3.000000	8.000000	9307.000000	1.000000	0.000000	1.000000	0.000000	0.000000	0.000000	0.000000	0
25%	35.000000	10.000000	39.000000	91911.000000	1.000000	0.700000	1.000000	0.000000	0.000000	0.000000	0.000000	0
50%	45.000000	20.000000	64.000000	93437.000000	2.000000	1.500000	2.000000	0.000000	0.000000	0.000000	0.000000	1
75%	55.000000	30.000000	98.000000	94608.000000	3.000000	2.500000	3.000000	101.000000	0.000000	0.000000	0.000000	1
max	67.000000	43.000000	224.000000	96651.000000	4.000000	10.000000	3.000000	635.000000	1.000000	1.000000	1.000000	1

Plotting Box plots for attributes "Income", "CCAvg" and "Mortgage":

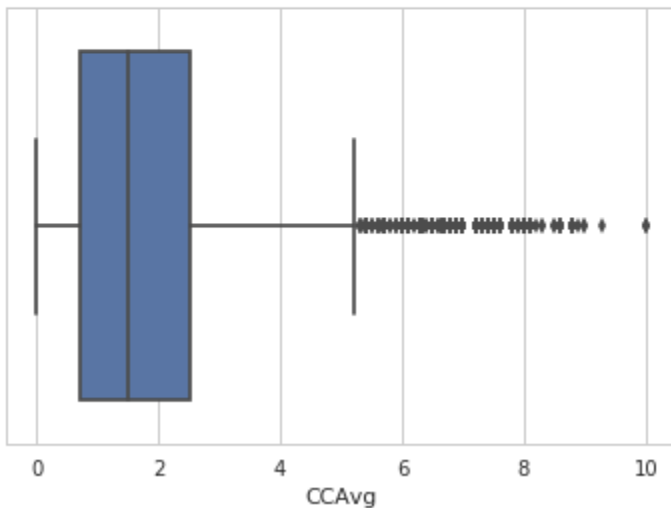
```
sns.set(style="whitegrid", color_codes=True)
px= sns.boxplot(x="Income", data= data)
```

/usr/local/anaconda/python3/lib/python3.6/site-packag
is a private function. Do not use.
box_data = remove_na(group_data)



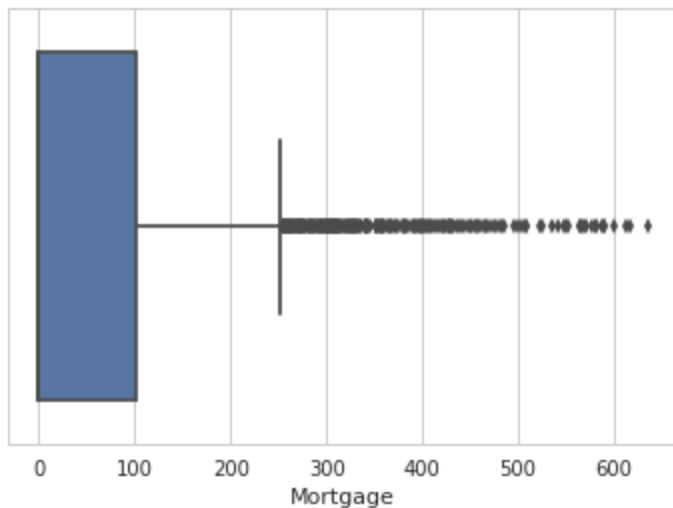
```
sns.set(style="whitegrid", color_codes=True)
px= sns.boxplot(x="CCAvg", data= data)
```

/usr/local/anaconda/python3/lib/python3.6/site-pa
is a private function. Do not use.
box_data = remove_na(group_data)



```
sns.set(style="whitegrid", color_codes=True)
px= sns.boxplot(x="Mortgage", data= data)

/usr/local/anaconda/python3/lib/python3.6/site-
is a private function. Do not use.
box_data = remove_na(group_data)
```

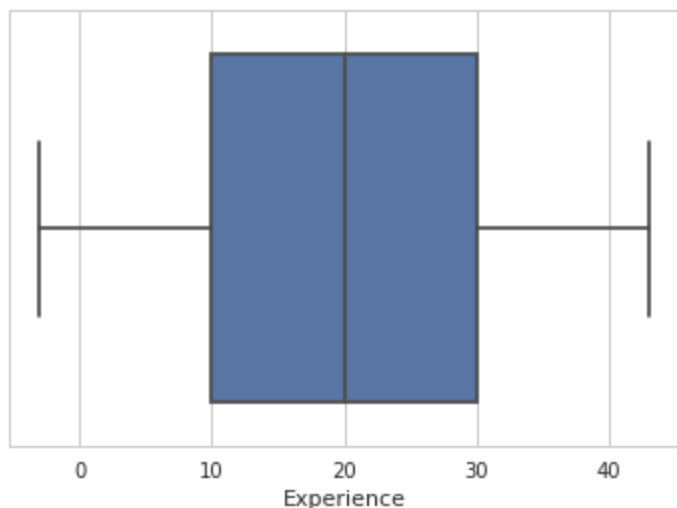


From the Box plots it is evident that the skewed distribution in the attributes "Income", "CCAvg" and "Mortgage" is not because of outliers. But the data itself is skewed.

Checking the distribution in "Experience" column, to check negative values in distribution.

```
sns.set(style="whitegrid", color_codes=True)
px= sns.boxplot(x="Experience", data= data, hue=True)

/usr/local/anaconda/python3/lib/python3.6/site-package
is a private function. Do not use.
box_data = remove_na(group_data)
```



#2 studying data distribution in each attributes, Share your findings:

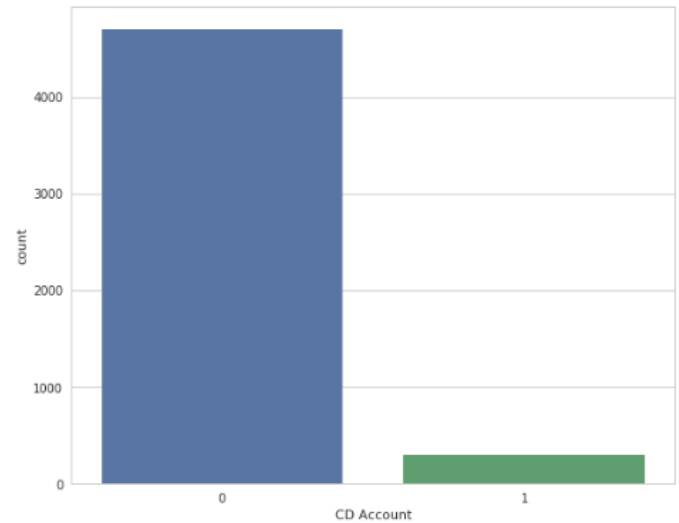
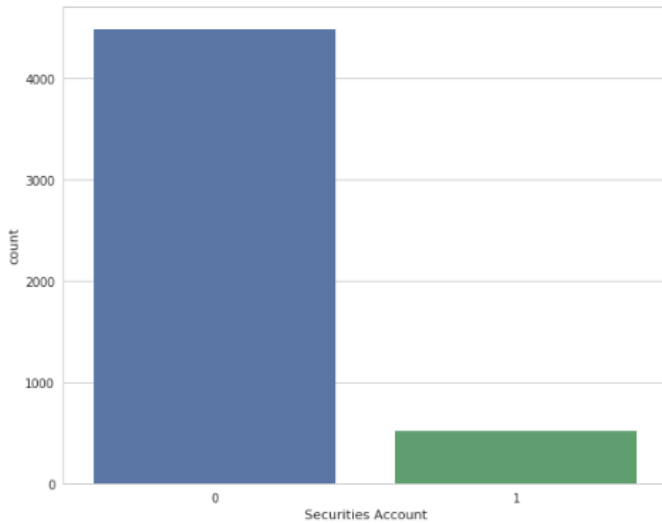
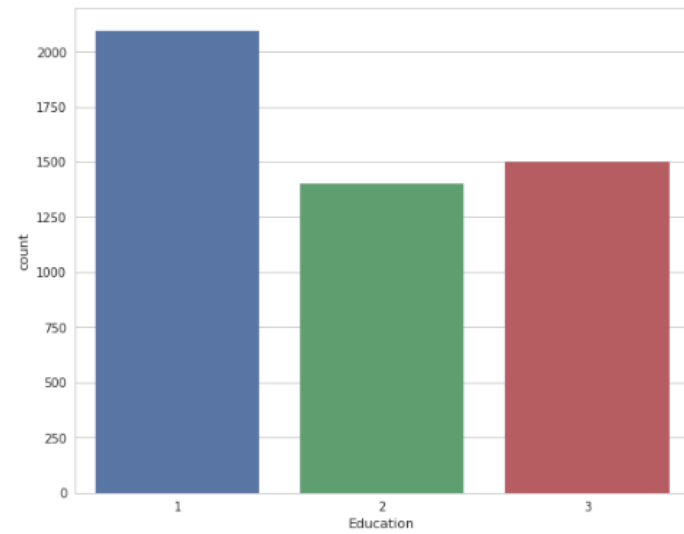
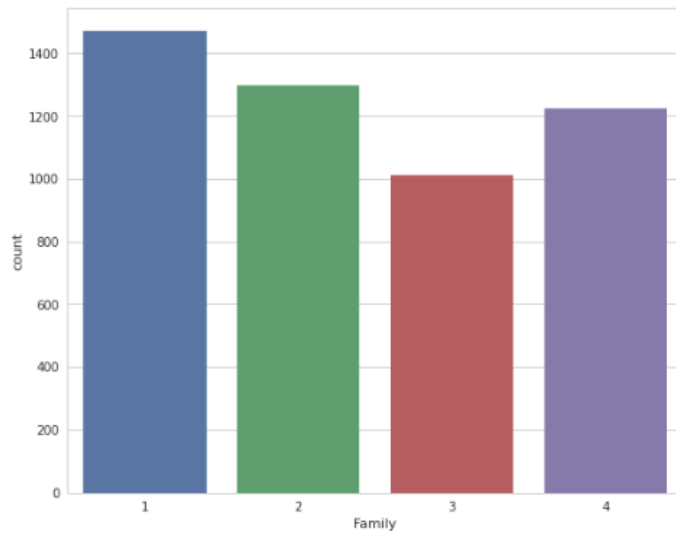
- **From the distribution of the independent attribute we can infer that :**
 - 1) Majority of the banks customers are having a family size of 1
 - 2) Majority of banks customers are Undergraduates followed by Advanced/Professional and Graduates.
 - 3) Majority of banks existing customers are not holding "Securities account" and "CD account"
 - 4) Majority of existing customers are using Online banking facility
 - 5) Majority of banks customers do not have a Credit card.

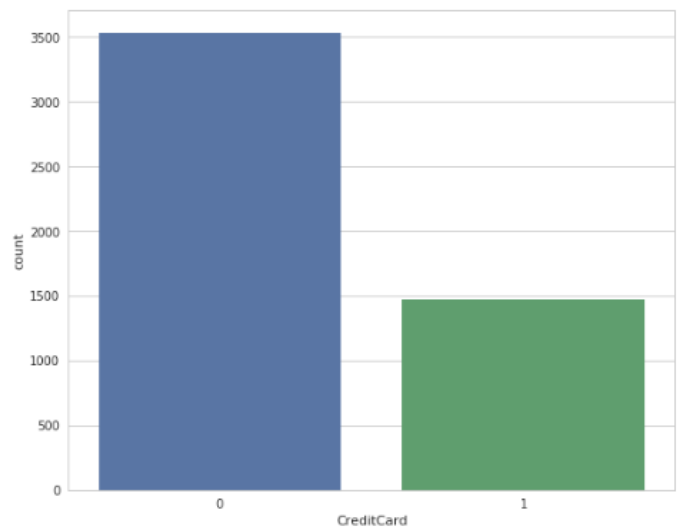
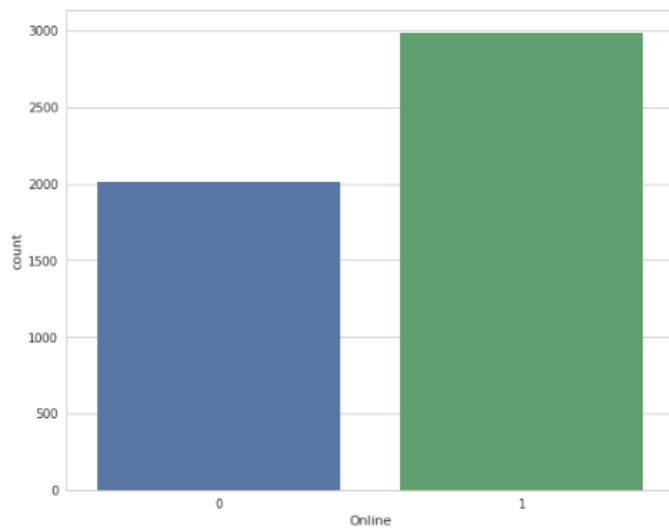
```

categorical = ["Family", "Education", "Securities Account", "CD Account", "Online", "CreditCard"]
fig, axs = plt.subplots(nrows=3, ncols=2, figsize=(20, 25))
for col, ax in zip(categorical[0:], axs.ravel()):
    sns.countplot(x=col, data=data, ax=ax)

```

/usr/local/anaconda/python3/lib/python3.6/site-packages/seaborn/categorical.py:1460: FutureWarning: remove_na is deprecated and is a private function. Do not use.
stat_data = remove_na(group_data)



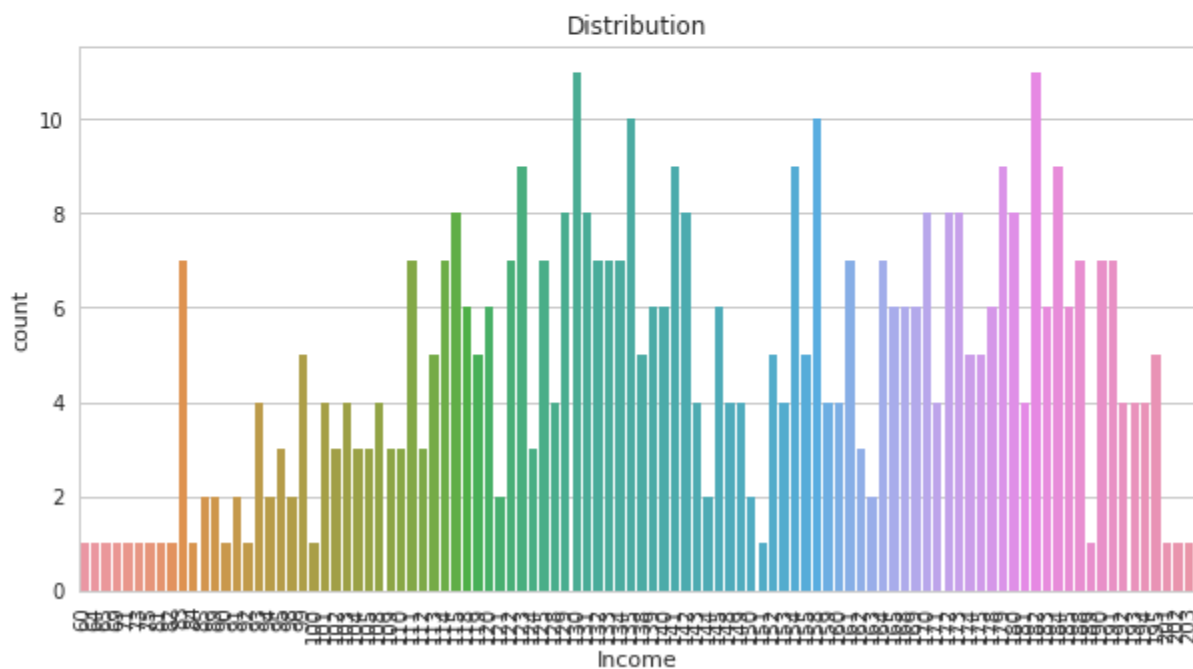


Observations based on the distribution of People who have taken the personal loan:

Relationship between Income and People who have taken personal loan:

```
from matplotlib import pyplot as plt
plt.figure(figsize=(10,5))
sns.countplot(x='Income', data=data_loan_yes)
plt.xticks(rotation='vertical')
plt.title('Distribution')
plt.show()
```

/usr/local/anaconda/python3/lib/python3.6/site-packages/seaborn/categorical.py:1
is a private function. Do not use.
stat_data = remove_na(group_data)

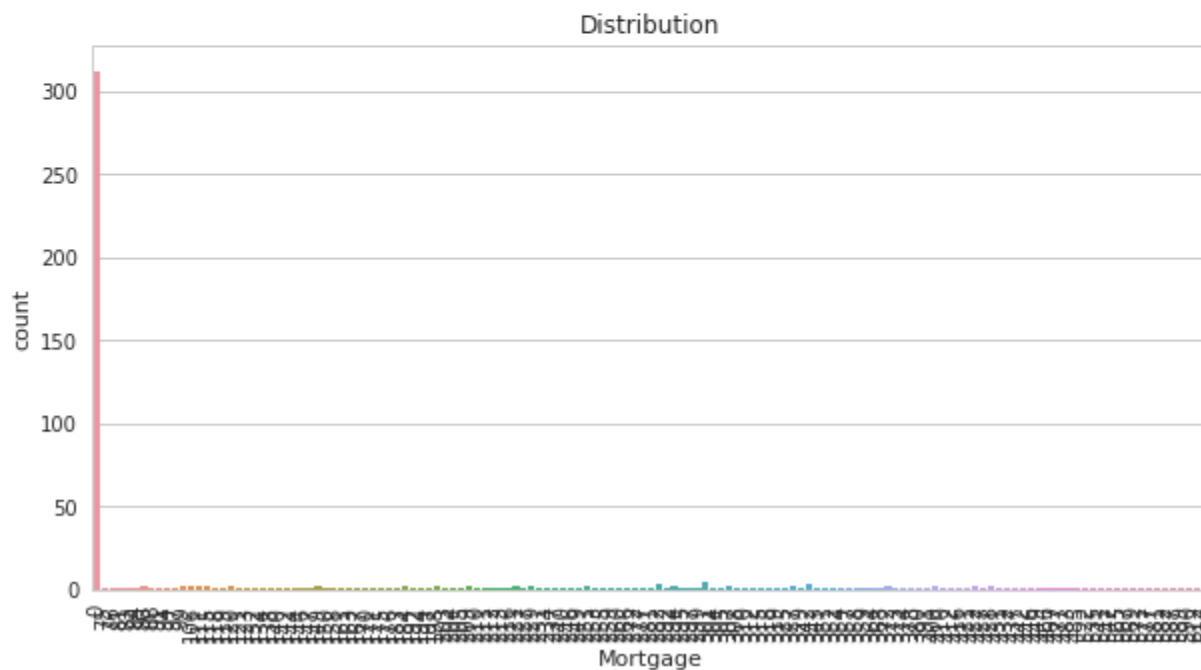


Income > 144k are more likely to take Personal Loan from above observation

```
# Relationship between Mortgage and People who have taken personal loan:
```

```
from matplotlib import pyplot as plt
plt.figure(figsize=(10,5))
sns.countplot(x='Mortgage', data=data_loan_yes)
plt.xticks(rotation='vertical')
plt.title('Distribution')
plt.show()
```

```
/usr/local/anaconda/python3/lib/python3.6/site-packages/seaborn/categorical.py:1460:
is a private function. Do not use.
    stat_data = remove_na(group_data)
```

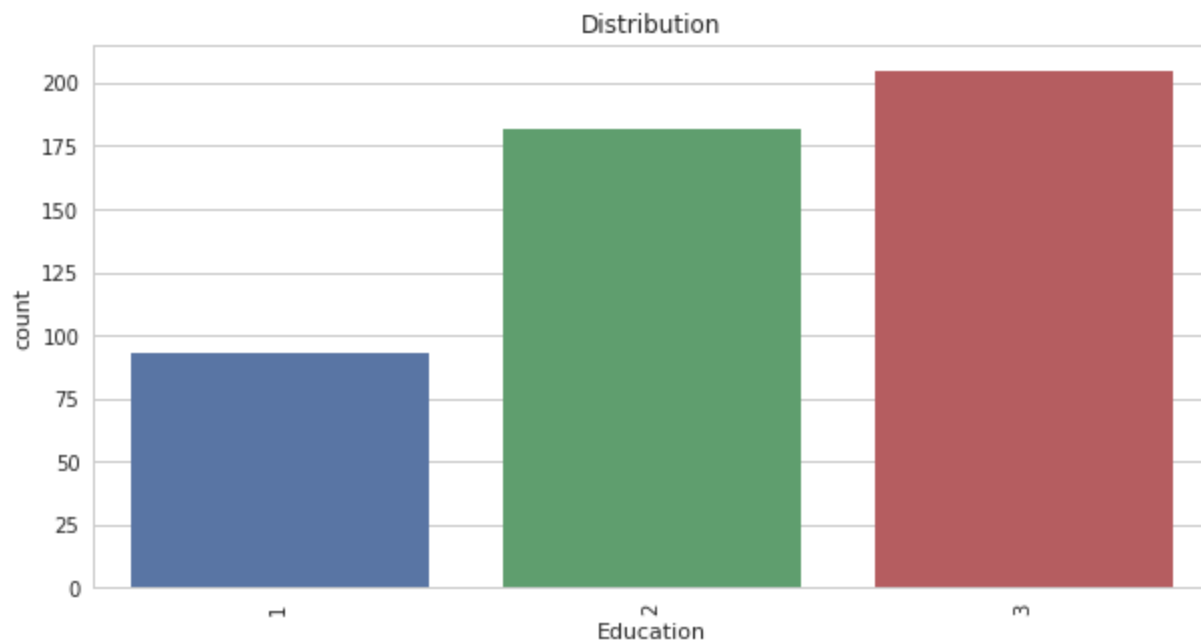


No Mortgage people are more prone to take personal loans from above observation

```
# Relationship between Education and People who have taken personal Loan:
```

```
from matplotlib import pyplot as plt
plt.figure(figsize=(10,5))
sns.countplot(x='Education', data=data_loan_yes)
plt.xticks(rotation='vertical')
plt.title('Distribution')
plt.show()
```

```
/usr/local/anaconda/python3/lib/python3.6/site-packages/seaborn/categorical.py:1460: FutureWarning:
is a private function. Do not use.
  stat_data = remove_na(group_data)
```

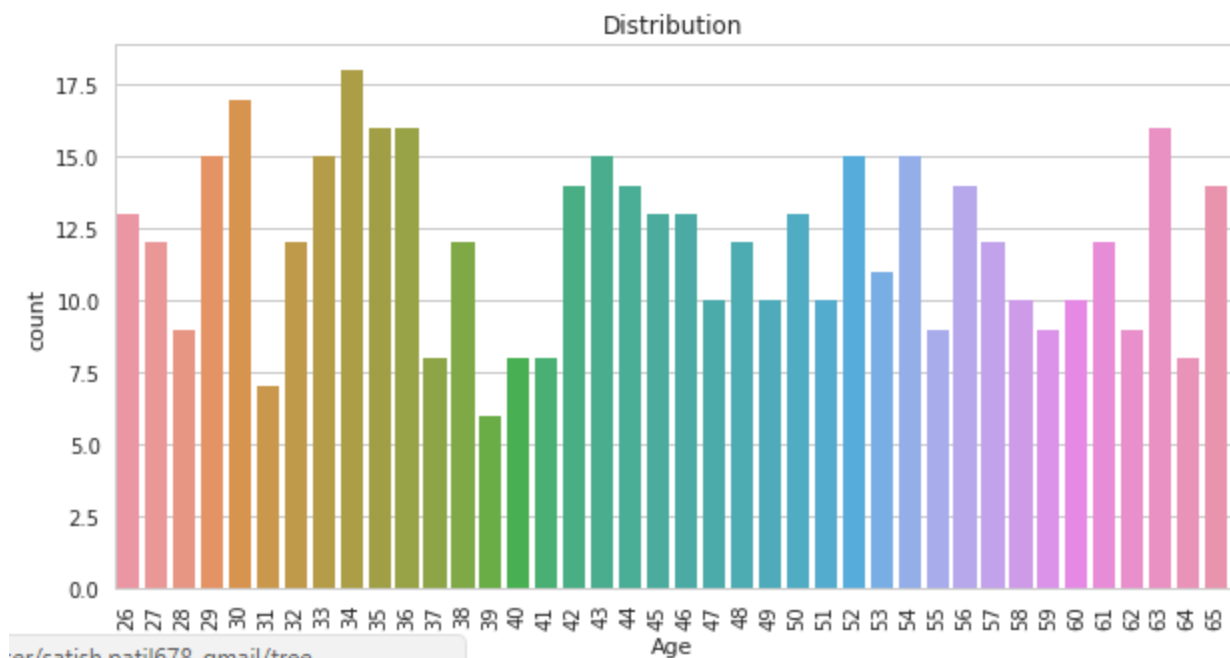


higher level of education are more likely to take personal loan followed by graduate from above observation.

```
# Relationship between Age and People who have taken personal loan:
```

```
from matplotlib import pyplot as plt
plt.figure(figsize=(10,5))
sns.countplot(x='Age', data=data_loan_yes)
plt.xticks(rotation='vertical')
plt.title('Distribution')
plt.show()
data_loan_yes.Age.mean()
```

```
/usr/local/anaconda/python3/lib/python3.6/site-packages/seaborn/categorical.py:1460: F
is a private function. Do not use.
stat_data = remove_na(group_data)
```



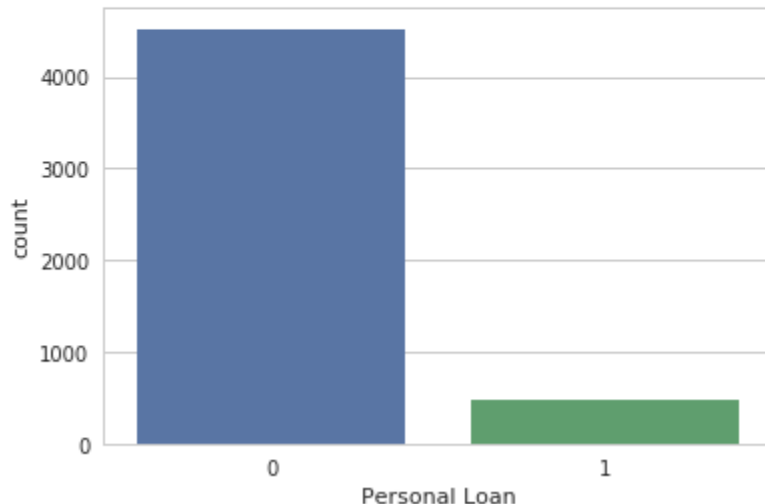
Age on higher side are more likely to take personal loan. The Average age of people who have taken loan is 45 years.

#3 Get the target column distribution. Your comments:

```
Target = data["Personal Loan"]
sns.countplot(x= Target, data=data)

/usr/local/anaconda/python3/lib/python3.6/site-packages/seaborn
is a private function. Do not use.
stat_data = remove_na(group_data)

<matplotlib.axes._subplots.AxesSubplot at 0x7fdf5eecb860>
```



It is evident from the distribution of Target variable("Personal Loan") that approximately only 10% of the customers have accepted the personal Loan.

```
data.corr()
```

	Age	Experience	Income	ZIP Code	Family	CCAvg	Education	Mortgage	Personal Loan	Securities Account	CD Account	Online	CreditCard
Age	1.000000	0.994215	-0.055269	-0.029216	-0.046418	-0.052030	0.041334	-0.012539	-0.007726	-0.000436	0.008043	0.013702	0.007681
Experience	0.994215	1.000000	-0.046574	-0.028626	-0.052563	-0.050089	0.013152	-0.010582	-0.007413	-0.001232	0.010353	0.013898	0.008967
Income	-0.055269	-0.046574	1.000000	-0.016410	-0.157501	0.645993	-0.187524	0.206806	0.502462	-0.002616	0.169738	0.014206	-0.002385
ZIP Code	-0.029216	-0.028626	-0.016410	1.000000	0.011778	-0.004068	-0.017377	0.007383	0.000107	0.004704	0.019972	0.016990	0.007691
Family	-0.046418	-0.052563	-0.157501	0.011778	1.000000	-0.109285	0.064929	-0.020445	0.061367	0.019994	0.014110	0.010354	0.011588
CCAvg	-0.052030	-0.050089	0.645993	-0.004068	-0.109285	1.000000	-0.136138	0.109909	0.366891	0.015087	0.136537	-0.003620	-0.006686
Education	0.041334	0.013152	-0.187524	-0.017377	0.064929	-0.136138	1.000000	-0.033327	0.136722	-0.010812	0.013934	-0.015004	-0.011014
Mortgage	-0.012539	-0.010582	0.206806	0.007383	-0.020445	0.109909	-0.033327	1.000000	0.142095	-0.005411	0.089311	-0.005995	-0.007231
Personal Loan	-0.007726	-0.007413	0.502462	0.000107	0.061367	0.366891	0.136722	0.142095	1.000000	0.021954	0.316355	0.006278	0.002802
Securities Account	-0.000436	-0.001232	-0.002616	0.004704	0.019994	0.015087	-0.010812	-0.005411	0.021954	1.000000	0.317034	0.012627	-0.015028
CD Account	0.008043	0.010353	0.169738	0.019972	0.014110	0.136537	0.013934	0.089311	0.316355	0.317034	1.000000	0.175880	0.278644
Online	0.013702	0.013898	0.014206	0.016990	0.010354	-0.003620	-0.015004	-0.005995	0.006278	0.012627	0.175880	1.000000	0.004210
CreditCard	0.007681	0.008967	-0.002385	0.007691	0.011588	-0.006686	-0.011014	-0.007231	0.002802	-0.015028	0.278644	0.004210	1.000000

From the correlation matrix we can infer that:

1)The attributes "Experience" and "Age" are having high positive correlation (correlation coefficient = 0.99)

2)The attributes "Income" and "CCAvg" has good positive correlation (correlation coefficient = 0.65)

The attributes that are having good correlation with Target variable("Personal Loan") are:

1) Income (0.5)

2) CCAvg (0.37)

3) CD Account (0.32)

The attribute "Zip Code" has having very least correlation with respect to Target variable (0.00011)

#4 Split the data into training and test set in the ratio of 70:30 respectively.

When we split the data into training and test set with 70:30, Training data = 3500 and Test data = 1500.

```
from sklearn.model_selection import train_test_split
from sklearn import metrics
```

```
X = data.drop("Personal Loan_1",axis=1)
Y = data["Personal Loan_1"]
```

```
# Splitting Data into 70% Training data and 30% Testing Data:
x_train, x_test, y_train, y_test = train_test_split(X, Y,train_size=0.7, test_size=0.3, random_state=0)
print(len(x_train))
print(len(x_test))
```

```
3500
1500
```

#5 Use a classification model to predict the likelihood of a liability customer buying personal loans.

Model-1 : Logistic Regression:

```
# Lets use the classification algorithm "LogisticRegression":
```

```
from sklearn.linear_model import LogisticRegression
logisticregression = LogisticRegression()
Model1 = logisticregression.fit(x_train, y_train)
```

```
print("Model score on training : "+str(Model1.score(x_train,y_train)))
print("Model score on test : "+str(Model1.score(x_test,y_test)))
```

```
Model score on training : 0.9588571428571429
Model score on test : 0.9553333333333334
```

```
y1_predict = Model1.predict(x_test)
print("Mean Absolute Error : " + str(metrics.mean_absolute_error(y1_predict, y_test)))
print("Confusion Matrix/n")
print(metrics.confusion_matrix(y_test, y1_predict))
print( metrics.accuracy_score(y_test, y1_predict) )
```

```
Mean Absolute Error : 2.0766666666666667
Confusion Matrix/n
[[1360  12]
 [ 55  73]]
0.9553333333333334
```

```
print(metrics.classification_report(y_test, y1_predict))
```

	precision	recall	f1-score	support
0	0.96	0.99	0.98	1372
1	0.86	0.57	0.69	128
avg / total	0.95	0.96	0.95	1500

ANALYZING THE CONFUSION MATRIX:

True Positives (TP): we correctly predicted the people have taken Personal loan 73

True Negatives (TN): we correctly predicted the people who have not taken Personal loan 1360

False Positives (FP): we incorrectly predicted that people have taken loan (a "Type I error") 12 Falsely predict positive Type I error

False Negatives (FN): we incorrectly predicted that people have not taken loan (a "Type II error") 55 Falsely predict negative Type II error

Model-2: K-Nearest neighbor's:

```
kNeighborsClassifier = KNeighborsClassifier(n_neighbors= 5 , weights = 'distance')
```

```
Model2 = kNeighborsClassifier.fit(x_train, y_train)
```

```
print("Model score on training : "+str(Model2.score(x_train,y_train)))
```

```
print("Model score on test : "+str(Model2.score(x_test,y_test)))
```

```
Model score on training : 1.0
```

```
Model score on test : 0.9113333333333333
```

```
y2_predict = Model2.predict(x_test)
```

```
print("Mean Absolute Error : " + str(metrics.mean_absolute_error(y2_predict, y_test)))
```

```
print("Confusion Matrix/n")
```

```
print(metrics.confusion_matrix(y_test, y2_predict))
```

```
print( metrics.accuracy_score(y_test, y2_predict) )
```

```
Mean Absolute Error : 8.047333333333333
```

```
Confusion Matrix/n
```

```
[[1325  47]
```

```
 [ 86  42]]
```

```
0.9113333333333333
```

```
print(metrics.classification_report(y_test, y2_predict))
```

	precision	recall	f1-score	support
0	0.94	0.97	0.95	1372
1	0.47	0.33	0.39	128
avg / total	0.90	0.91	0.90	1500

ANALYZING THE CONFUSION MATRIX:

True Positives (TP): we correctly predicted the people have taken Personal loan 42

True Negatives (TN): we correctly predicted the people who have not taken Personal loan 1325

False Positives (FP): we incorrectly predicted that people have taken loan (a "Type I error") 47 Falsely predict positive Type I error

False Negatives (FN): we incorrectly predicted that people have not taken loan (a "Type II error") 86 Falsely predict negative Type II error

Model-3: Naive bayes:

```
from sklearn.naive_bayes import GaussianNB
```

```
gaussianNB = GaussianNB()
```

```
Model3 = gaussianNB.fit(x_train, y_train)
```

```
print("Model score on training : "+str(Model3.score(x_train,y_train)))  
print("Model score on test : "+str(Model3.score(x_test,y_test)))
```

```
Model score on training : 0.8831428571428571  
Model score on test : 0.8886666666666667
```

```
y3_predict = Model3.predict(x_test)  
print("Mean Absolute Error : " + str(metrics.mean_absolute_error(y3_predict,  
print("Confusion Matrix/n")  
print(metrics.confusion_matrix(y_test, y3_predict))  
print( metrics.accuracy_score(y_test, y3_predict) )
```

```
Mean Absolute Error : 19.584666666666667  
Confusion Matrix/n  
[[1257  115]  
 [  52   76]]  
0.8886666666666667
```

```
print(metrics.classification_report(y_test, y3_predict))
```

	precision	recall	f1-score	support
0	0.96	0.92	0.94	1372
1	0.40	0.59	0.48	128

ser/satish.natil678@gmail/tree

ANALYZING THE CONFUSION MATRIX:

True Positives (TP): we correctly predicted the people have taken Personal loan as 76

True Negatives (TN): we correctly predicted the people who have not taken Personal loan as 1257

False Positives (FP): we incorrectly predicted that people have taken loan (a "Type I error") 115 Falsely predict positive Type I error

False Negatives (FN): we incorrectly predicted that people have not taken loan (a "Type II error") 52 Falsely predict negative Type II error

Model-4: Decision tree classifier:

```
from sklearn.tree import DecisionTreeClassifier
```

```
decisionTreeClassifier = DecisionTreeClassifier(criterion = 'entropy' )
```

```
Model4 = decisionTreeClassifier.fit(x_train, y_train)
```

```
print("Model score on training : "+str(Model4.score(x_train,y_train)))  
print("Model score on test : "+str(Model4.score(x_test,y_test)))
```

```
Model score on training : 1.0  
Model score on test : 0.988
```

```
y4_predict = Model4.predict(x_test)  
print("Mean Absolute Error : " + str(metrics.mean_absolute_error(y4_predict, y_test)))  
print("Confusion Matrix/n")  
print(metrics.confusion_matrix(y_test, y4_predict))  
print( metrics.accuracy_score(y_test, y4_predict) )
```

```
Mean Absolute Error : 1.028  
Confusion Matrix/n  
[[1366    6]  
 [  12  116]]  
0.988
```

```
print(metrics.classification_report(y_test, y4_predict))
```

	precision	recall	f1-score	support
0	0.99	1.00	0.99	1372
1	0.95	0.91	0.93	128
avg / total	0.99	0.99	0.99	1500

ANALYZING THE CONFUSION MATRIX:

True Positives (TP): we correctly predicted the people have taken Personal loan as 116

True Negatives (TN): we correctly predicted the people who have not taken Personal loan as 1365

False Positives (FP): we incorrectly predicted that people have taken loan (a "Type I error") 6 Falsely predict positive Type I error

False Negatives (FN): we incorrectly predicted that people have not taken loan (a "Type II error") 12 Falsely predict negative Type II error

Conclusion :

From the above analysis all models overall accuracy is good, But when we look at precision and recall value, Recall accuracy is low, Which indicated that our models are overfitting on training data due to imbalance dataset. Even without any model one can say that the given test set belongs to Majority class with 90% confidence.

But Still Decision tree models accuracy is good in terms of overall, precision and recall as trees won't effect much from imbalance data. But its overfitted due to max depth of a tree.

Using Regularizing techniques on decision we can overcome of overfitting

From the data set , we have only 480 records which are positive and rest as negative.

This is Imbalanced Class data with 1:9 ratio

```
# Let us Look at the target column which is 'Personal Loan' to understand how the data is distributed amongst the various values
# Most are not negatives. The ratio is almost 1:9 in favor of class 0. The model's ability to predict class 0 will
# be better than predicting class 1.
data.groupby(["Personal Loan_1"]).count()
```

	Age	Income	CCAvg	Mortgage	Family_2	Family_3	Family_4	Education_2	Education_3	Securities Account_1	CD Account_1	Online_1	CreditCard_1
Personal Loan_1													
0	4520	4520	4520	4520	4520	4520	4520	4520	4520	4520	4520	4520	4520
1	480	480	480	480	480	480	480	480	480	480	480	480	480

We have many ways to handle to imbalance data sets we are considering 2 methods

Case 1: Changing model parameters or using Ensemble techniques with trees.

Case 2: Upsampling of minority class data.

Case 1 :

Regularizing the Decision Tree

```
#dt_model = DecisionTreeClassifier(criterion = 'entropy', class_weight={0:.5,1:.5}, max_depth = 5, min_samples_leaf=5 )  
#dt_model.fit(train_set, train_labels)
```

```
model5 = DecisionTreeClassifier(criterion = 'entropy', max_depth = 5)  
model5.fit(x_train, y_train)
```

```
print("Model score on training : "+str(model5.score(x_train,y_train)))  
print("Model score on test : "+str(model5.score(x_test,y_test)))
```

```
Model score on training : 0.9888571428571429  
Model score on test : 0.988
```

```
y5_predict = model5.predict(x_test)  
print("Mean Absolute Error : " + str(metrics.mean_absolute_error(y5_predict, y_test)))  
print("Confusion Matrix/n")  
print(metrics.confusion_matrix(y_test, y5_predict))  
print( metrics.accuracy_score(y_test, y5_predict) )
```

```
Mean Absolute Error : 0.6893333333333334  
Confusion Matrix/n  
[[1368    4]  
 [  14  114]]  
0.988
```

```
print(metrics.classification_report(y_test, y5_predict))
```

	precision	recall	f1-score	support
0	0.99	1.00	0.99	1372
1	0.97	0.89	0.93	128
avg / total	0.99	0.99	0.99	1500

ANALYZING THE CONFUSION MATRIX:

True Positives (TP): we correctly predicted the people have taken Personal loan as 114

True Negatives (TN): we correctly predicted the people who have not taken Personal loan as 1368

False Positives (FP): we incorrectly predicted that people have taken loan (a "Type I error") 4
False Negatives (FN): we incorrectly predicted that people have not taken loan (a "Type II error") 14

False Negatives (FN): we incorrectly predicted that people have not taken loan (a "Type II error") 14 Falsely predict negative Type II error

```
#Validating model with K-Fold Validation|
from sklearn.model_selection import cross_val_score
scores = cross_val_score(model5, x_train, y_train, cv=5)
print(scores)
```

```
[0.98002853 0.98145506 0.98714286 0.98998569 0.98283262]
```

Gradient Boosting Models with XGBoost:

XGBoost takes care of Imbalance in datasets and helps in prediction accuracy on unseen data

```
from xgboost import XGBClassifier

model6 = XGBClassifier(n_estimators=1000, learning_rate=0.05)
# Add silent=True to avoid printing out updates with each cycle
model6.fit(x_train, y_train, verbose=False)
```

```
print("Model score on training : "+str(model6.score(x_train,y_train)))
print("Model score on test : "+str(model6.score(x_test,y_test)))
```

```
Model score on training : 0.9994285714285714
Model score on test : 0.9833333333333333
```

```
y6_predict = model6.predict(x_test)
print("Mean Absolute Error : " + str(metrics.mean_absolute_error(y6_predict, y_test)))
print("Confusion Matrix/n")
print(metrics.confusion_matrix(y_test, y6_predict))
print( metrics.accuracy_score(y_test, y6_predict) )
```

```
Mean Absolute Error : 1.0326666666666666
Confusion Matrix/n
[[1366    6]
 [  19 109]]
0.9833333333333333
```

```
print(metrics.classification_report(y_test, y6_predict))
```

	precision	recall	f1-score	support
0	0.99	1.00	0.99	1372
1	0.95	0.85	0.90	128
avg / total	0.98	0.98	0.98	1500

```
#Validating model with K-Fold Validation  
from sklearn.model_selection import cross_val_score  
scores = cross_val_score(model6, x_train, y_train, cv=5)  
print(scores)
```

```
[0.98573466 0.9871612 0.98571429 0.98569385 0.981402 ]
```

Case 2:

Therefore, if we were to always predict 0 - people who will not take loan, we'd achieve an accuracy more than 90%.

On the contrary, for people who will take loan, hardly we will achieve 10% accuracy.

Up Sampling is one technique which can be used to make this data unbiased

```
# Separate majority and minority classes
df_majority = data[data['Personal Loan_1']==0]
df_minority = data[data['Personal Loan_1']==1]

from sklearn.utils import resample
# Upsample minority class
df_minority_upsampled = resample(df_minority,
                                replace=True,      # sample with replacement
                                n_samples=4520,    # to match majority class
                                random_state=123)  # reproducible results

# Combine majority class with upsampled minority class
df_upsampled = pd.concat([df_majority, df_minority_upsampled])

# Display new class counts
df_upsampled['Personal Loan_1'].value_counts()

1    4520
0    4520
Name: Personal Loan_1, dtype: int64

Sampled_X = df_upsampled.drop("Personal Loan_1",axis=1)
Sampled_Y = df_upsampled["Personal Loan_1"]

x_train, x_test, y_train, y_test = train_test_split(Sampled_X, Sampled_Y,train_size=0.7, test_size=0.3, random_state=0)
```

We can Iterate the process of using all the model with Sampled_X and Sampled_Y and check the accuracy of a model post upsampling.

Let's try on one Example with Upsampled Data (Logistic Regression)

```
# Lets use the classification algorithm "LogisticRegression":
logisticregressionOnSampledData = LogisticRegression()
Model7 = logisticregressionOnSampledData.fit(x_train, y_train)
```

```
print("Model score on training : "+str(Model7.score(x_train,y_train)))
print("Model score on test : "+str(Model7.score(x_test,y_test)))
```

```
Model score on training : 0.9054993678887484
Model score on test : 0.8971238938053098
```

```
y7_predict = Model7.predict(x_test)
print("Mean Absolute Error : " + str(metrics.mean_absolute_error(y7_predict, y_test)))
print("Confusion Matrix/n")
print(metrics.confusion_matrix(y_test, y7_predict))
print( metrics.accuracy_score(y_test, y7_predict) )
```

```
Mean Absolute Error : 12.46570796460177
Confusion Matrix/n
[[1220  132]
 [ 147 1213]]
0.8971238938053098
```

```
print(metrics.classification_report(y_test, y7_predict))
```

	precision	recall	f1-score	support
0	0.89	0.90	0.90	1352
1	0.90	0.89	0.90	1360
avg / total	0.90	0.90	0.90	2712

```
#Validating model with K-Fold Validation
from sklearn.model_selection import cross_val_score
scores = cross_val_score(Model7, x_train, y_train, cv=5)
print(scores)
```

```
[0.90916272 0.91153239 0.90995261 0.90197628 0.89802372]
```

Need to repeat the process on KNN, Naive Base and Decesion Tree

#6 Explain why you chose one model over the other:

Conclusion:

Decision Tree is best suited model for the given dataset over other models. Because of imbalance in dataset, Decision Tree classification is not effected much as compared to Logistic, KNN and Naive Base.

There are many techniques to use for this kind of data set. We have taken 2 cases: 1) By Changing Model parameters 2) Upsampling of Minarity Class data.

We can see the Overall accuracy, Precesion , Recall and f1-score are good in Decision Tree , XGBClassifier.

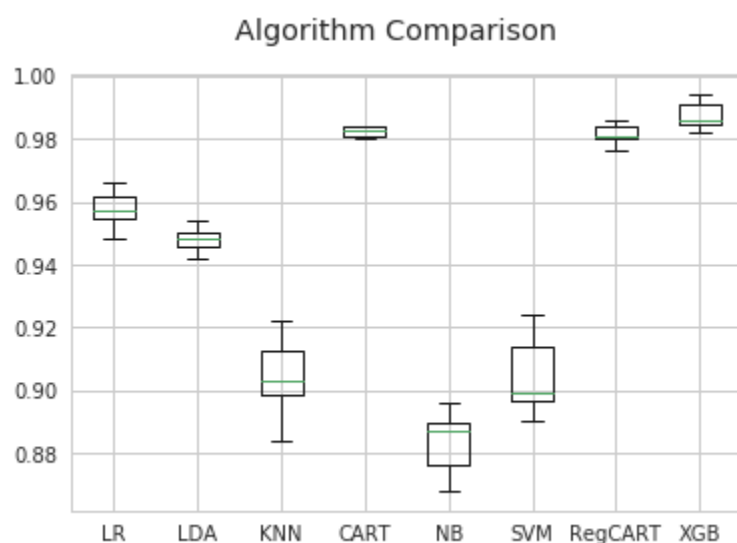
```
import matplotlib.pyplot as plt
from sklearn import model_selection
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
from sklearn.svm import SVC
```

```
models = []
models.append(('LR', Model1))
models.append(('LDA', LinearDiscriminantAnalysis()))
models.append(('KNN', Model2))
models.append(('CART', Model4))
models.append(('NB', Model3))
models.append(('SVM', SVC()))
models.append(('RegCART', model5))
models.append(('XGB', model6))

# 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()
```

LR: 0.957600 (0.005276)
LDA: 0.946400 (0.006917)
KNN: 0.904400 (0.010151)
CART: 0.983400 (0.003904)
NB: 0.883000 (0.009220)
SVM: 0.905200 (0.011496)
RegCART: 0.981400 (0.003470)

XGB: 0.987400 (0.003800)



Gradient Boosting Models with XGBoost (XGB) has the overall highest accuracy and is a good predictor for an unseen data over production environment.