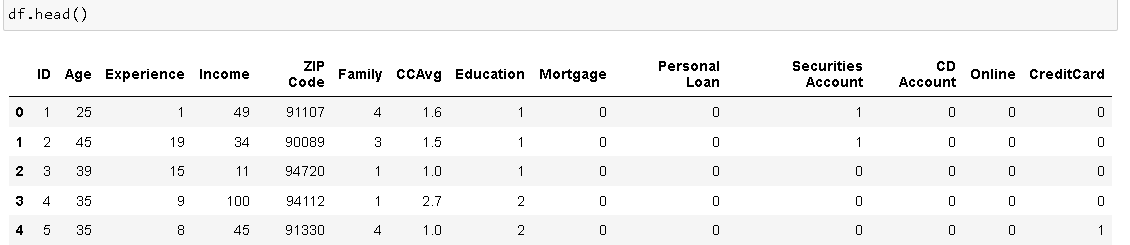
**1. Read the column description and ensure you understand each attribute well**

*df = pd.read\_csv(r'C:\Users\hp\Desktop\supervised learning\graded\Bank\_Personal\_Loan\_Modelling.csv')*

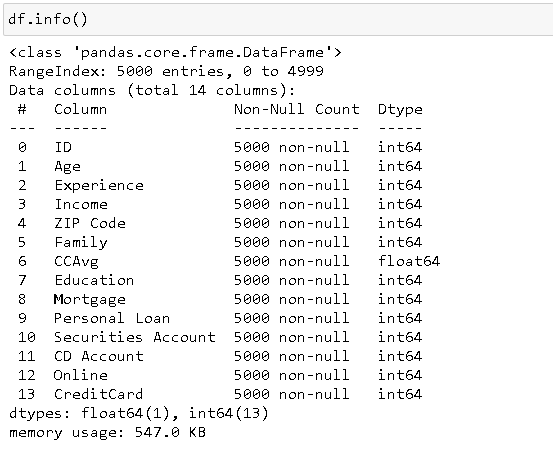
*df.head()*

**

*df.shape*

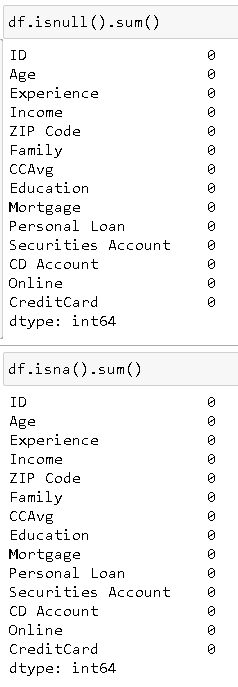
**

*df.info()*

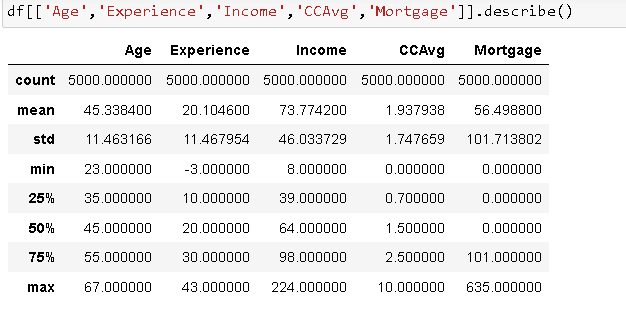
**

*df.isnull().sum()*

*df.isna().sum()*

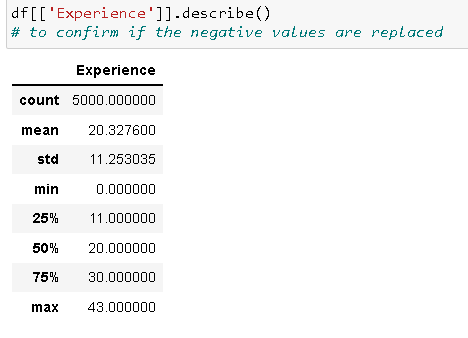
**

*df[['Age','Experience','Income','CCAvg','Mortgage']].describe()*

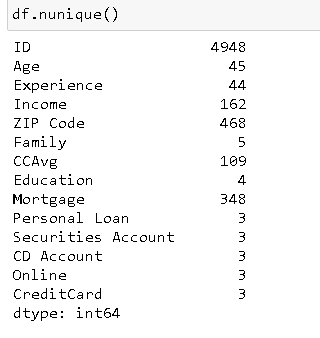
**

*df.loc[df['Experience'] < 0] = df['Experience'].median()*

*df[['Experience']].describe()*

**

*df.nunique()*



Approach:

* Read the csv file using read\_csv() function in pandas
* Df.shape returns rows and columns
* info() function returns number of entries and data types of each columns
* isna() and isnull() functions detect missing values and null values respectively
* describe() function: to return 5-point summary, mean, std dev and count of the attribute. We invoked it to confirm that the distribution of all the variables are as expected (we found out that ‘experience’ has some negative entries, so we replace them with the median value)
* dataframe.loc[condition]: locates the rows on certain conditions (in this case, where ‘Experience’ is negative). We can assign these rows with desired values. (in this case, median of ‘Experience i.e. dataframe[‘column’].median())
* nunique() provides total number of unique entries in each column (to confirm that number of unique values are as expected, especially in categorical columns)

Insights and Inferences:

* 5000 rows, 14 columns
* No missing values; no null values
* ‘Experience’ columns have some negative value (which we assume to be entry error), so we replace it with the median value
* ID's have to be unique. We only have 4948 unique IDs. It means we have 52 multiple/duplicate/wrong entries.
* Personal Loan, Securities account, online, etc., should have 2 categories, but they have 3 unique values. (we will detect the flaw in the next section and correct it)
* Significant right skew in Income, Mortgage and CCAvg is observed
* **Categorical columns:**
  + **Family**: Number of Family Members (ordered categorical attribute)
  + **Education**: (1:Undergrad); (2:Graduate); (3:Advanced/Professional) (can be taken as ordered categorical attribute)
  + **Securities Account**: customer having/not having securities account with the Bank (categorical, not ordered)
  + **CD Account**: customer having/not having Certificate of Diposit account with the Bank (categorical, not ordered)
  + **Online**: Customer using/ not using Internet banking (categorical, not ordered)
  + **Credit Card**: consumer using/not using Credit Card(categorical, not ordered)
* **Nominal Variables:**
  + **ID**: act as an identifire for a customer (must be unique)
  + **ZIP Code**: location of the customer's home address (qualitative categorical variable, can be used in prediction model as an attribute)
* **Numerical Columns:**
  + **Age**: Age of customer (ordered numerical attribute)
  + **Experience**: professional experience of customer (ordered numnerical attribute)
  + **Income**: Annual Income of Customer (ordered numerical attribute)
  + **CCAvg**: Avg. Annual expenditure via credit card(ordered numerical attribute)
  + **Mortgage**: value of house mortgage (ordered numerical attribute)
* **Target Variable:**
  + **Personal Loan**: The customer availing the loan (0=no loan; 1=loan availed)

**2. Perform univariate analysis of each and every attribute - use an appropriate plot for a given attribute and mention your insights**

*print(df.Family.value\_counts())*

*print(df.Education.value\_counts())*

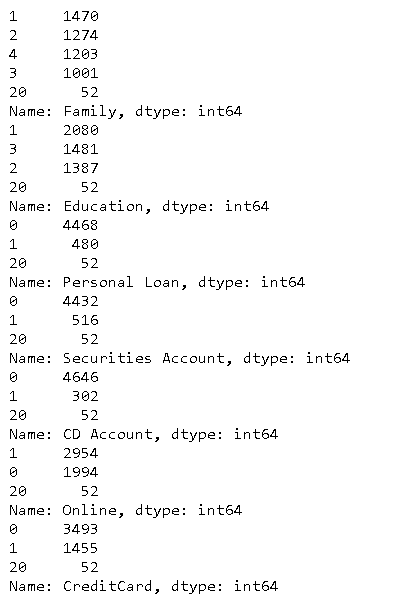
*print(df['Personal Loan'].value\_counts())*

*print(df['Securities Account'].value\_counts())*

*print(df['CD Account'].value\_counts())*

*print(df['Online'].value\_counts())*

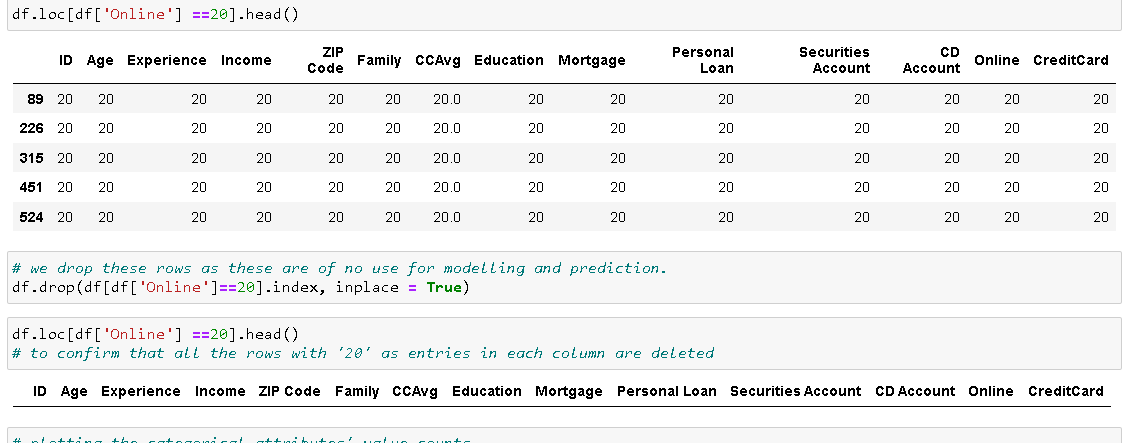
*print(df['CreditCard'].value\_counts())*

**

*df.loc[df['Online'] ==20].head()*

*df.drop(df[df['Online']==20].index, inplace = True)*

*df.loc[df['Online'] ==20].head()*

**

*plt.figure(figsize= (13,22))*

*plt.subplot(7,2,1)*

*df.Family.value\_counts().plot(kind='bar')*

*plt.xlabel('Family')*

*plt.subplot(7,2,2)*

*df.Education.value\_counts().plot(kind='bar')*

*plt.xlabel('Education')*

*plt.subplot(7,2,3)*

*df['Personal Loan'].value\_counts().plot(kind='bar')*

*plt.xlabel('Personal Loan')*

*plt.subplot(7,2,4)*

*df['Securities Account'].value\_counts().plot(kind='bar')*

*plt.xlabel('Securities Account')*

*plt.subplot(7,2,5)*

*df['CD Account'].value\_counts().plot(kind='bar')*

*plt.xlabel('CD Account')*

*plt.subplot(7,2,6)*

*df['Online'].value\_counts().plot(kind='bar')*

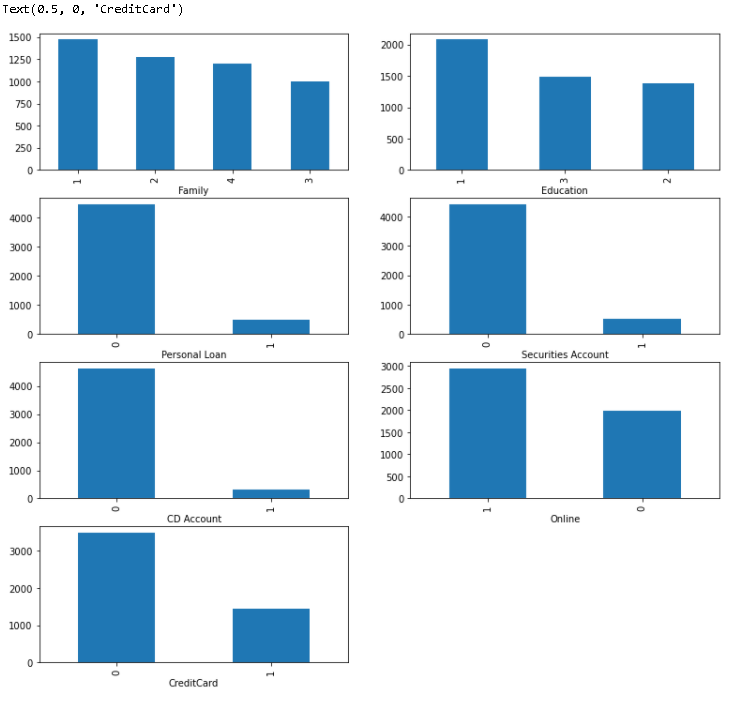
*plt.xlabel('Online')*

*plt.subplot(7,2,7)*

*df['CreditCard'].value\_counts().plot(kind='bar')*

*plt.xlabel('CreditCard')*

*plt.show()*

**

*plt.figure(figsize= (10,16))*

*plt.subplot(5,2,1)*

*sns.distplot(df['Age'])*

*plt.xlabel('Age')*

*plt.subplot(5,2,2)*

*sns.distplot(df['Experience'])*

*plt.xlabel('Experience')*

*plt.subplot(5,2,3)*

*sns.distplot(df['Income'])*

*plt.xlabel('Income')*

*plt.subplot(5,2,4)*

*sns.distplot(df['CCAvg'])*

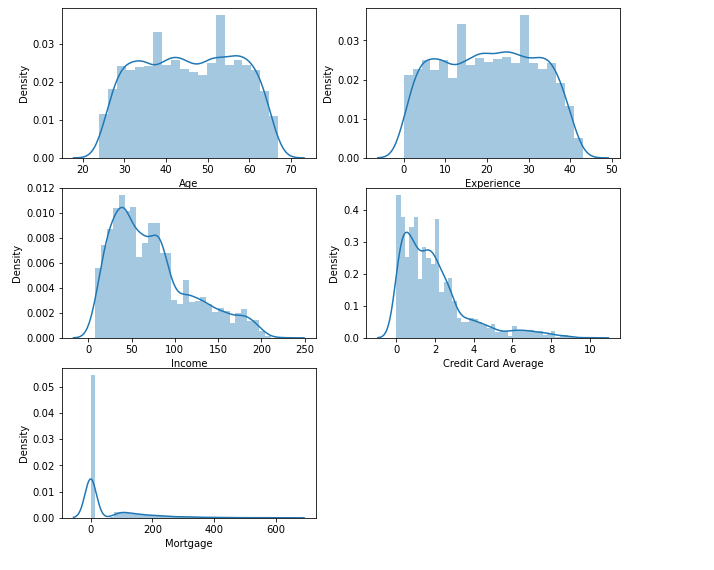
*plt.xlabel('Credit Card Average')*

*plt.subplot(5,2,5)*

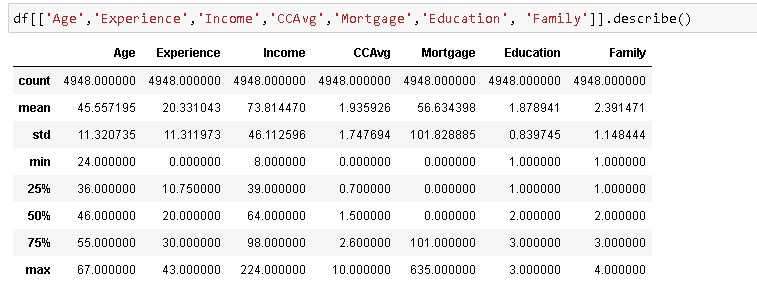
*sns.distplot(df['Mortgage'])*

*plt.xlabel('Mortgage')*

*plt.show()*

**

*df[['Age','Experience','Income','CCAvg','Mortgage','Education', 'Family']].describe()*

**

*plt.figure(figsize= (10,16))*

*plt.subplot(5,2,1)*

*sns.boxplot(df['Age'])*

*plt.xlabel('Age')*

*plt.subplot(5,2,2)*

*sns.boxplot(df['Experience'])*

*plt.xlabel('Experience')*

*plt.subplot(5,2,3)*

*sns.boxplot(df['Income'])*

*plt.xlabel('Income')*

*plt.subplot(5,2,4)*

*sns.boxplot(df['CCAvg'])*

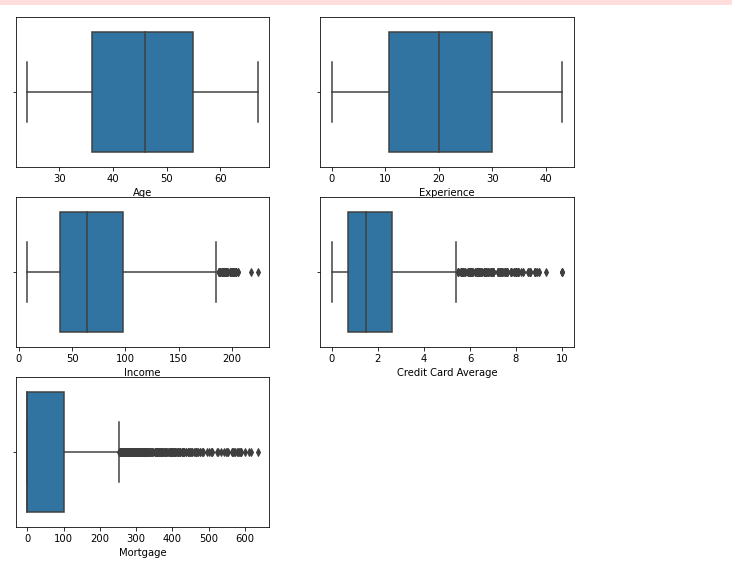
*plt.xlabel('Credit Card Average')*

*plt.subplot(5,2,5)*

*sns.boxplot(df['Mortgage'])*

*plt.xlabel('Mortgage')*

*plt.show()*

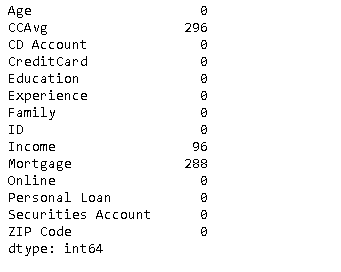
**

*Q1 = df.quantile(0.25)*

*Q3 = df.quantile(0.75)*

*IQR = Q3 - Q1*

*((df[['Income', 'CCAvg','Mortgage']] < (Q1 - 1.5 \* IQR)) | (df[['Income', 'CCAvg','Mortgage']] > (Q3 + 1.5 \* IQR))).sum()*

**

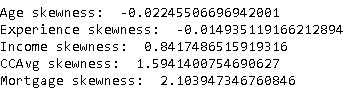
*print('Age skewness: ', stats.skew(df.Age))*

*print('Experience skewness: ',stats.skew(df.Experience))*

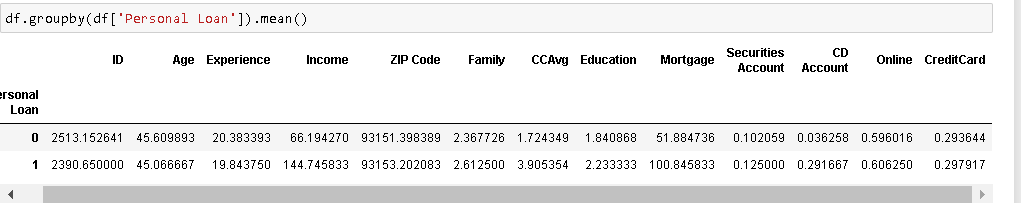
*print('Income skewness: ',stats.skew(df.Income))*

*print('CCAvg skewness: ',stats.skew(df.CCAvg))*

*print('Mortgage skewness: ', stats.skew(df.Mortgage))*

**

*df.groupby(df['Personal Loan']).mean()*

**

*plt.figure(figsize= (13,40))*

*plt.subplot(11,2,1)*

*sns.barplot(x="Personal Loan", y="Family", data=df)*

*plt.xlabel('Family vs Personal Loan')*

*plt.subplot(11,2,2)*

*sns.barplot(x="Personal Loan", y="Education", data=df)*

*plt.xlabel('Education vs Personal Loan')*

*plt.subplot(11,2,3)*

*sns.barplot(x="Personal Loan", y="Securities Account", data=df)*

*plt.xlabel('Securities Account vs Personal Loan')*

*plt.subplot(11,2,4)*

*sns.barplot(x="Personal Loan", y="CD Account", data=df)*

*plt.xlabel('CD Account vs Personal Loan')*

*plt.subplot(11,2,5)*

*sns.barplot(x="Personal Loan", y="Online", data=df)*

*plt.xlabel('Online vs Personal Loan')*

*plt.subplot(11,2,6)*

*sns.barplot(x="Personal Loan", y="CreditCard", data=df)*

*plt.xlabel('CreditCard vs Personal Lon')*

*plt.subplot(11,2,7)*

*sns.barplot(x="Personal Loan", y="Age", data=df)*

*plt.xlabel('Age vs Personal Loan')*

*plt.subplot(11,2,8)*

*sns.barplot(x="Personal Loan", y="Experience", data=df)*

*plt.xlabel('Experience vs Personal Loan')*

*plt.subplot(11,2,9)*

*sns.barplot(x="Personal Loan", y="CCAvg", data=df)*

*plt.xlabel('CCAvg vs Personal Loan')*

*plt.subplot(11,2,10)*

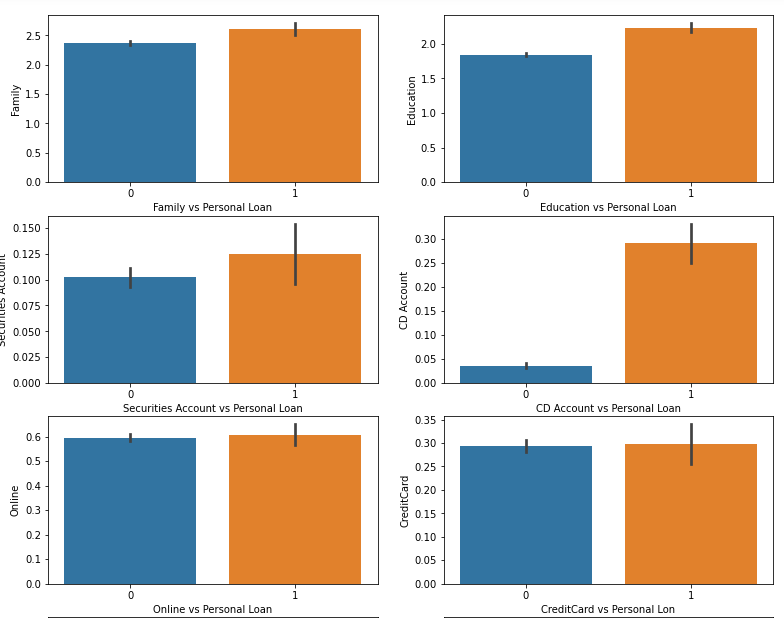
*sns.barplot(x="Personal Loan", y="Mortgage", data=df)*

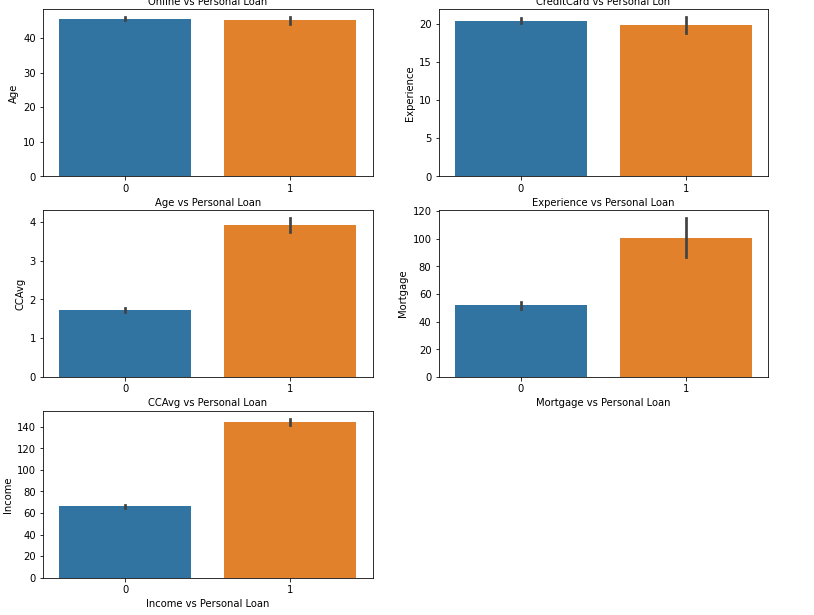
*plt.xlabel('Mortgage vs Personal Loan')*

*plt.subplot(11,2,11)*

*sns.barplot(x="Personal Loan", y="Income", data=df)*

*plt.xlabel('Income vs Personal Loan')*

**

**

*df=df.drop(columns= ['ID'])*

Approach:

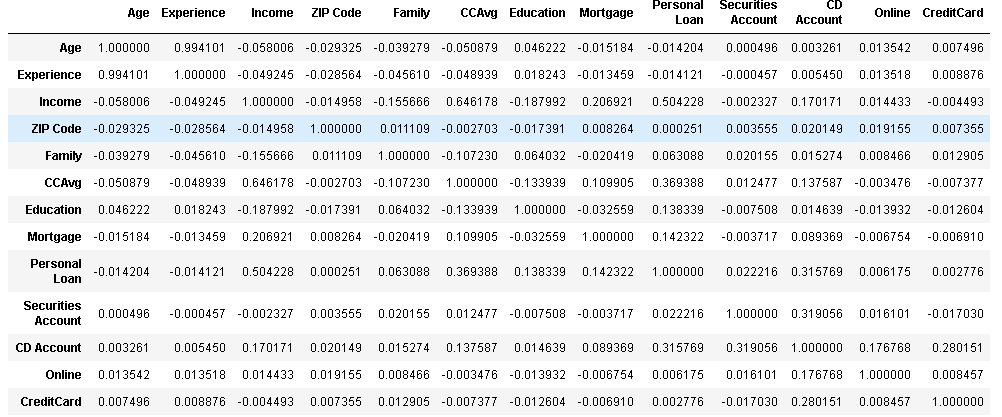
* value\_counts() function is used to print counts of distinct entries in a categorical column. (to get an idea of the distribution of categorical variables)
* Dataframe.loc[condition]: locates all the rows with the mentioned condition (in this case, those rows with entry 20 in each column). We drop these columns using (dataframe.drop(dataframe[<condition>].index, inplace=True)). Here, ‘inplace=True’ means that the selected rows are dropped from the original dataframe.
* plt.subplot() is used to plot multiple plots with a single call.
* Appending .plot(kind=<type>) after value\_counts(), creates a bar plot with values on x-axis and counts on the y-axis (eg. df.Family.value\_counts().plot(kind='bar')).
* plt.x/ylabel(‘<Label>’) is used to label the respective axis. It is mentioned after the plot function has been called.
* sns.distplot(<column>) plots the histogram with a normal fitting curve. We use it to see whether the distribution of the numerical variables are normal or not and to judge their skewness
* sns.boxplot(<column>) is used to plot boxplot of numerical attributes. It provides a visual form of 5 points summary and is very useful in spotting outliers. We can also get an idea whether the distribution is normal or not from this graph only.
* dataframe.quantile(<percentile>) gives us the value at the mentioned percentile. It is used to calculate interquantile range, to help in counting the outliers.
* (column<IQR OR column>IQR).sum() ) code prints the outliers for a column. We use this value to find that weather outlier is an oneoff event or a large number of values lie outside of the IQR.
* stats.skew(column) prints the skewness of a column.
* df.groupby(‘Personal Loan’).mean(): we used this code to find the mean of all the columns of df dataframe differentiated by whether they are availing Personal Loan or not.
* sns.barplot(x=’Personal Loan’, y=<y\_coulmn>, data=dataframe): it is a graphical presentation of the previous operation. Mean values of other columns are plotted in y axis against 0/1 value of ‘Personal Loan’.

Insights and Inferences:

* There were 52 rows that look bogus as all the entries in them was 20. So, we dropped them altogether.
* Customers availing Securities account, Credit Card, or CD account facility is far lower than customers not availing them.
* Customers using online services is higher than those who don’t
* Income, Mortgage and Credit Cad Average are right skewed. Age and Experience are normally distributed (but they have multiple peaks)
* Boxplots confirm the right skew in Income, Mortgage and CCAvg. Also, presence of outliers has been observed in these three columns. We need to count the outliers to comment whether they are flawed entries or are a result of the right skew
* Large proportion of customers have income between 45000 to 550000.
* Average spending on credit card is between 0 to 10000 and a large proportion of customers spend less than 2500.
* Customers with high income, higher CCAvg and high mortgage end up availing Personal Loans more than those with lower Income, CCAvg and Mortgage.
* Customers who avail Personal Loan are more likely to have securities account. (Same case with CD account).

**3. Perform correlation analysis among all the variables - you can use Pairplot and Correlation coefficients of every attribute with every other attribute**

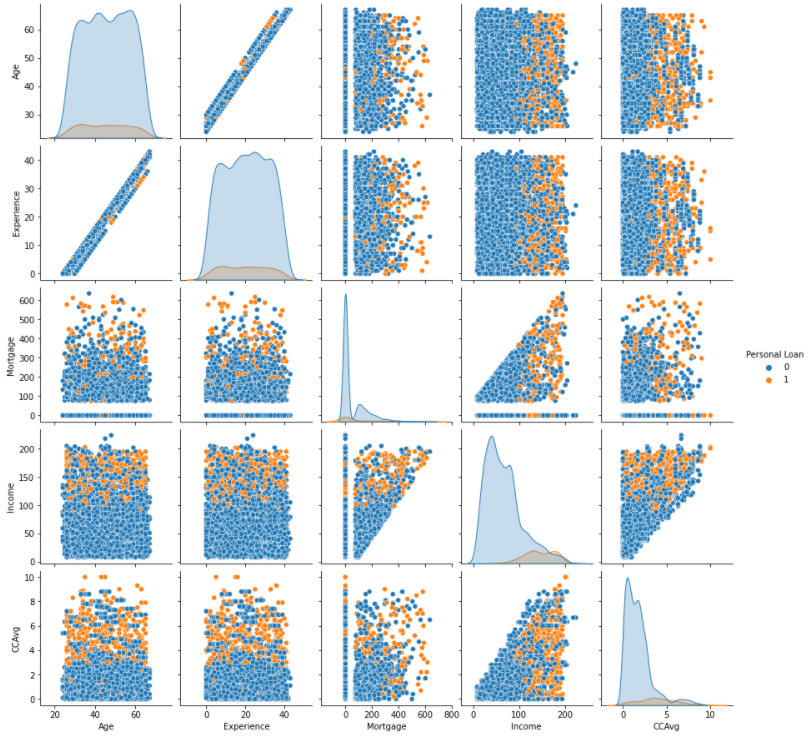
*df.corr()*

**

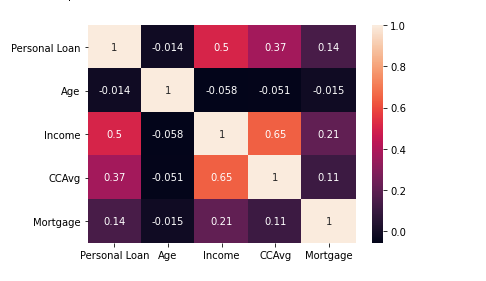
*sns.pairplot(df)*

*(refer Jupyter notebook for its output as it is not visually efficient, we use following visualisations )*

*sns.pairplot(df, vars=['Age', 'Experience', 'Mortgage', 'Income', 'CCAvg'], hue='Personal Loan')*

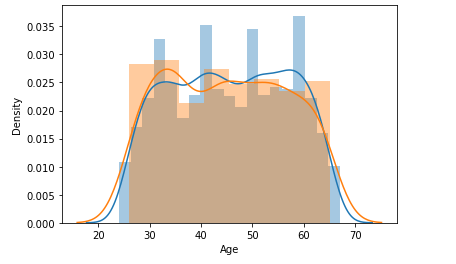
**

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

**

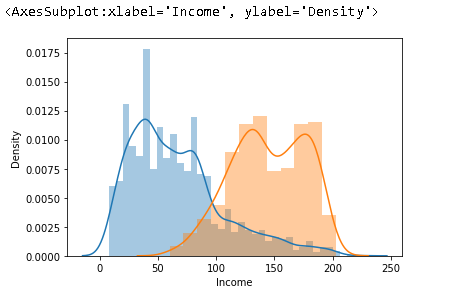
*sns.distplot(df[df["Personal Loan"] == 0]['Age'])*

*sns.distplot(df[df["Personal Loan"] == 1]['Age'])*

**

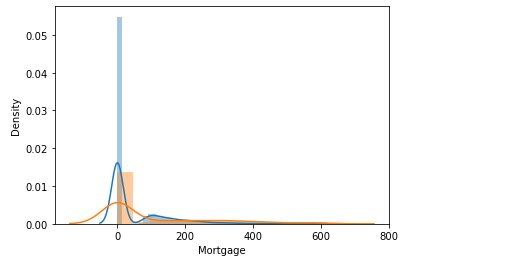
*sns.distplot(df[df["Personal Loan"] == 0]['Income'])*

*sns.distplot(df[df["Personal Loan"] == 1]['Income'])*

**

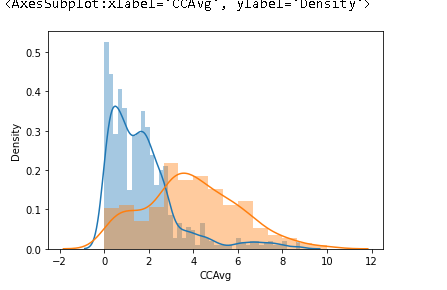
*sns.distplot(df[df["Personal Loan"] == 0]['Mortgage'])*

*sns.distplot(df[df["Personal Loan"] == 1]['Mortgage'])*

**

*sns.distplot(df[df["Personal Loan"] == 0]['CCAvg'])*

*sns.distplot(df[df["Personal Loan"] == 1]['CCAvg'])*

**

*plt.figure(figsize= (13,40))*

*plt.subplot(11,2,1)*

*sns.countplot(x="Family", data=df ,hue="Personal Loan")*

*plt.xlabel('Family vs Personal Loan')*

*plt.subplot(11,2,2)*

*sns.countplot(x="Education", data=df ,hue="Personal Loan")*

*plt.xlabel('Education vs Personal Loan')*

*plt.subplot(11,2,3)*

*sns.countplot(x="Securities Account", data=df ,hue="Personal Loan")*

*plt.xlabel('Securities Account vs Personal Loan')*

*plt.subplot(11,2,4)*

*sns.countplot(x="CD Account", data=df ,hue="Personal Loan")*

*plt.xlabel('CD Account vs Personal Loan')*

*plt.subplot(11,2,5)*

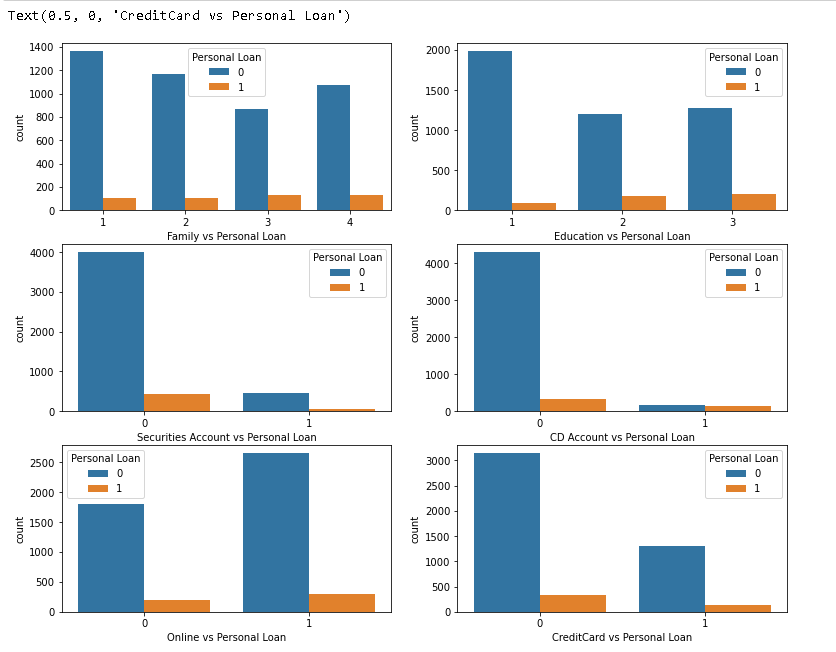
*sns.countplot(x="Online", data=df ,hue="Personal Loan")*

*plt.xlabel('Online vs Personal Loan')*

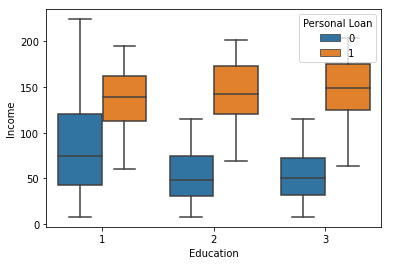
*plt.subplot(11,2,6)*

*sns.countplot(x="CreditCard", data=df ,hue="Personal Loan")*

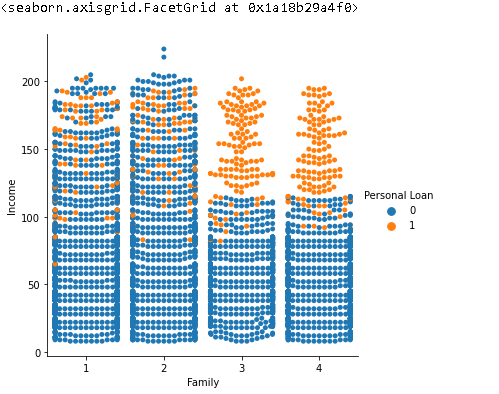
*plt.xlabel('CreditCard vs Personal Loan')*

**

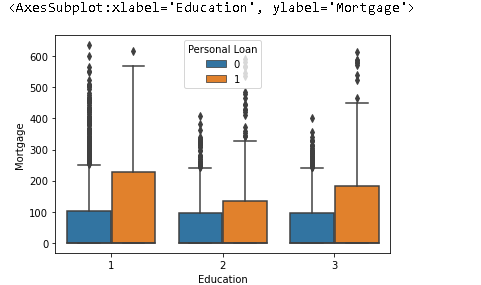
*sns.boxplot(x='Education', y='Income', hue='Personal Loan',data=df)*

**

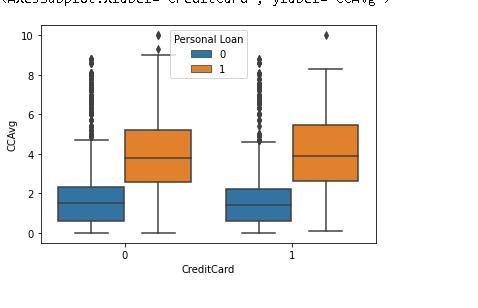
*sns.catplot(x='Family', y='Income', hue='Personal Loan', data = df, kind='swarm')*

**

*sns.boxplot(x="Education", y='Mortgage', hue="Personal Loan", data=df)*

**

*sns.boxplot(x="CreditCard", y='CCAvg', hue="Personal Loan", data=df)*

**

Approach:

* dataframe.corr() is used to plot correlation between different columns.
* sns.pairplot(dataframe) to plot scatter plot between different columns and Normal histogram between same columns. (it is also helpful in visual presentation of corr() results)
* sns.pairplot(dataframe, var=columns, hue=column): We can selectively plot the scatter plot to get deeper insights. Added to it we can add “Hue” to the plot so as to differentiate between the customers who availed the loan vs those who didn’t.
* sns.distplot(df[df["Personal Loan"] == 0][<column>])

sns.distplot(df[df["Personal Loan"] == 1][<column>])

It effectively plots two normal histogram plots of the <column> differentiated by the ‘Personal Loan’ status of the customers. Helps in determining the distribution and range of the two categories of the customers

* sns.countplot(x=<column>, data=dataframe ,hue="Personal Loan"): plots the count plot of categories of categorical variables. Each category has two count bars- one for customers not availing loan and other one for customers availing loans
* sns.boxplot(x=<categorical column>, y=<numerical column>, hue='Personal Loan',data=dataframe): plots boxplots for the numerical column on the basis of their categories and weather they have availed personal loan or not.
* sns.catplot(x=<categorical column>, y=<numerical column>, hue='Personal Loan',data=dataframe): similar to previous criterion, only difference is that here scatter plot is plotted

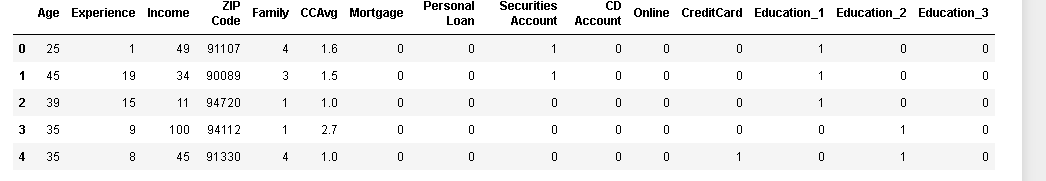
Insights and Inferences:

* Age and Experience are heavily correlated (0.99)
* Income and CC Avg are also correlated to a large extent (but we will still use both of them in model building as both of them are not totally correlated)
* We can actually drop one of age or experience during the model building
* Simple pairplot visualisation was not very efficient (as it is not visible clearly, and there are a lot of useless graphs). So we have used a pairplot with a limited number of columns.
* Mortgage and CCAvg are related to income
* loan availing customers have higher income and higher CCAvg
* Proportion of customers availing personal loan increases with education level and availability of CD account.
* Box plots infer that as income and education improves, propensity of a customer to avail personal loan increases
* Catplot between family and income suggest that larger families with high income almost certainly will avail a personal loan
* Boxplot between Mortgage and education suggest that those who have higher mortgage usually avail personal loan in every level of education
* Having a credit card doesn’t affect loan availing behaviour, however those with higher credit card spending do avail loan more frequently

**4. One hot encode the Education variable**

*df = pd.get\_dummies(df, columns = ['Education'])*

*df.head()*

**

*df=df.drop(['Experience'], axis=1)*

*df['ZIP Code'] = df['ZIP Code'].astype('category',copy=False)*

*df.info()*

Approach:

* One-hot encoding can be done by get\_dummies function or by OneHotEncoder function. Here we have used get\_dummies on education column for its simplicity.

Insights and Inferences:

* Since Experience and age are highly correlated (0.99), we drop one to reduce the complexity of the model.

###### **5. Separate the data into dependent and independent variables and create training and test sets out of them (X\_train, y\_train, X\_test, y\_test)**

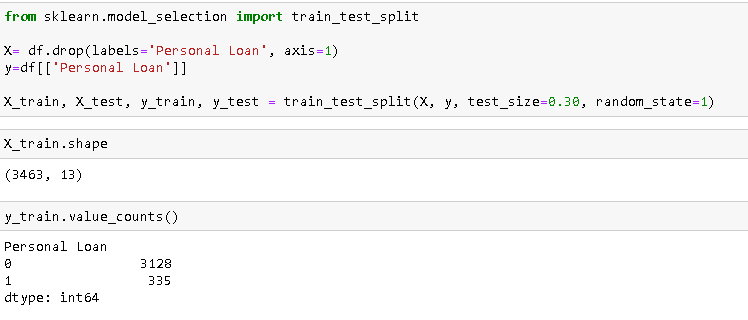
*X= df.drop(labels='Personal Loan', axis=1)*

*y=df[['Personal Loan']]*

*X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.30, random\_state=1)*

*X\_train.shape*

*y\_train.value\_counts()*

**

Approach:

* First we create X dataframe that contains all the columns except ‘Personal Loan’ that is the target attribute. Then we create y dataframe which contains the target attribute “Personal Loan”
* We use train\_test\_split to split the dataframes into test dataset(30%) and train dataset (70%).

Insights and Inferences:

###### **6. 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 data using this object, making sure that the test set does not influence the values of the train set**

*from sklearn.preprocessing import StandardScaler*

*scaler = StandardScaler()*

*X\_train = scaler.fit\_transform(X\_train)*

*X\_test = scaler.transform(X\_test)*

Approach:

* We use X\_train to fit the standardscaler() and then transform both X\_test and X\_Train

Insights and Inferences:

**7. 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**

*def regressor(regtype ,X\_train, X\_test, y\_train, y\_test):*

*regtype.fit(X\_train, y\_train)*

*y\_pred = regtype.predict(X\_test)*

*accuracy\_test=accuracy\_score(y\_test, y\_pred)*

*accuracy\_train=accuracy\_score(y\_train, regtype.predict(X\_train))*

*precision=precision\_score(y\_test,y\_pred)*

*recall=recall\_score(y\_test, y\_pred)*

*f1=f1\_score(y\_test, y\_pred)*

*tn, fp, fn, tp = confusion\_matrix(y\_test, y\_pred).ravel()*

*specificity = tn / (tn+fp)*

*cm=confusion\_matrix(y\_test,y\_pred)*

*return accuracy\_test, accuracy\_train, precision, recall, specificity, f1, cm*

Approach:

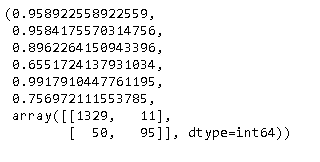
* Function has five arguments : regtype (name of the regression and the function required to call it), X\_train, X\_test, y\_train and y\_test.
* First we fit the X\_train and y\_Train using the given technique of regression.
* We use it to predict the target variable in the X\_test dataset and store it in y\_pred.
* Then to get different scores, we use y\_test (original dataset containing actual data) and y\_pred (containing predicted data)
* Accuracy, Precision ,recall and f1\_score can be received by calling their respective functions.
* For specificity, we need to have true positives, false positives, true negatives, and false negatives so that we can directly calculate it using the formula (true\_positive)/(true\_negative+false\_positive).
* In this case tp (true positive) is the value 1 (availing personal loan) and true negative is 0 (not availing personal loan)
* We calculate accuracy for both test and train set to ensure if there is no drastic difference between the two.
* Following points are returned by the function: accuracy\_test, accuracy\_train, precision, recall, specificity, f1\_score and confusion matrix.
* We print confusion matrix so that we can recalculate accuracy, precision, recall to ensure that their respective functions return correct values.

Insights and Inferences:

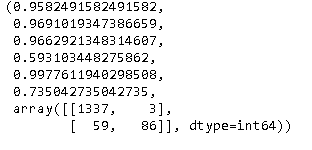
* (positive=1 i.e., customer availing the loan)
* precision: TP/(TP+FP)= ratio of true positives to total number of positive claims
* recall: TP/(TP+FN)= of all customers availing the loan, how many were correctly predicted
* F1\_score: (2\*precision\*recall)/(precision+recall)= weighted combination of precision (how many instances it classifies correctly), recall (robustness) (it does not miss a significant number of customers)
* Specificity: TN/(TN+FP)=of all customers not availing the loan, how many did we correctly predict

**8. 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**

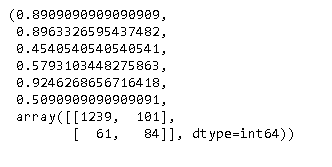
*regressor(LogisticRegression(), X\_train,X\_test,y\_train,y\_test)*

**

*regressor(KNeighborsClassifier(),X\_train,X\_test,y\_train,y\_test)*

**

*regressor(GaussianNB(), X\_train,X\_test,y\_train,y\_test)*

****

Approach:

* We call the function by calling the function and entering the required arguments.

Insights and Inferences:

**9. Create a dataframe with the columns - “Model”, “accuracy”, “recall”, “precision”, “specificity”, “f1\_score”. Populate the dataframe accordingly**

*data = {'Model':['Gaussian Naive Bayes'],*

*'Accuracy\_test':regressor(GaussianNB(), X\_train,X\_test,y\_train,y\_test)[0],*

*'Accuracy\_train':regressor(GaussianNB(), X\_train,X\_test,y\_train,y\_test)[1],*

*'Precision':regressor(GaussianNB(), X\_train,X\_test,y\_train,y\_test)[2],*

*'Recall':regressor(GaussianNB(), X\_train,X\_test,y\_train,y\_test)[3],*

*'Specificity':regressor(GaussianNB(), X\_train,X\_test,y\_train,y\_test)[4],*

*'F1\_Score':regressor(GaussianNB(), X\_train,X\_test,y\_train,y\_test)[5]}*

*model\_dataframe = pd.DataFrame(data)*

*data = {'Model':'Logistic Regression',*

*'Accuracy\_test':regressor(LogisticRegression(), X\_train,X\_test,y\_train,y\_test)[0],*

*'Accuracy\_train':regressor(LogisticRegression(), X\_train,X\_test,y\_train,y\_test)[1],*

*'Precision':regressor(LogisticRegression(), X\_train,X\_test,y\_train,y\_test)[2],*

*'Recall':regressor(LogisticRegression(), X\_train,X\_test,y\_train,y\_test)[3],*

*'Specificity':regressor(LogisticRegression(), X\_train,X\_test,y\_train,y\_test)[4],*

*'F1\_Score':regressor(LogisticRegression(), X\_train,X\_test,y\_train,y\_test)[5]}*

*model\_dataframe= model\_dataframe.append(data, ignore\_index = True)*

*data = {'Model':'K-nearest Neighbours classification',*

*'Accuracy\_test':regressor(KNeighborsClassifier(), X\_train,X\_test,y\_train,y\_test)[0],*

*'Accuracy\_train':regressor(KNeighborsClassifier(), X\_train,X\_test,y\_train,y\_test)[1],*

*'Precision':regressor(KNeighborsClassifier(), X\_train,X\_test,y\_train,y\_test)[2],*

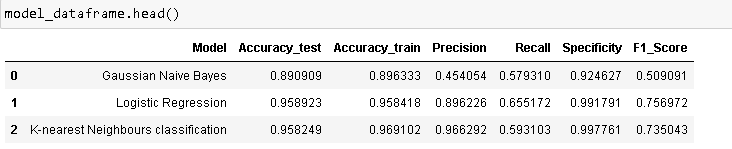
*'Recall':regressor(KNeighborsClassifier(), X\_train,X\_test,y\_train,y\_test)[3],*

*'Specificity':regressor(KNeighborsClassifier(), X\_train,X\_test,y\_train,y\_test)[4],*

*'F1\_Score':regressor(KNeighborsClassifier(), X\_train,X\_test,y\_train,y\_test)[5]}*

*model\_dataframe= model\_dataframe.append(data, ignore\_index = True)*

*model\_dataframe.head()*

**

Approach:

* We create a dataframe model\_dataframe and create a list (named data) that contains entries with respect to the required column titles.
* The list is then appended to the model\_dataframe dataset.

Insights and Inferences:

**10. Give your reasoning on which is the best model in this case**

Insights and Inferences:

* **All three models perform similarly on both the test and train dataset (in terms of accuracy). Precision of Gaussian NB is very low**
* **Precision for KNN is highest (logistic regression is not so far behind)**
* **Recall for Logistic regression is highest (KNN is not so far behind)**
* **Specificity is the measure of correctly predicting those customers who didn't avail the loan (which is not of prime interest here, however it is comparable for both KNN and logistic regression)**
* **We will judge on the basis of Recall and Precision (its weighted average is given by F1\_Score)**
* **Since the F1\_Score for Logistic Regression is the highest, we decided that it is the best model in this case.**
* **Also, KNN is a distance based model which not ideal to classify the target.**