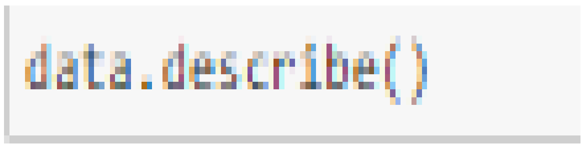
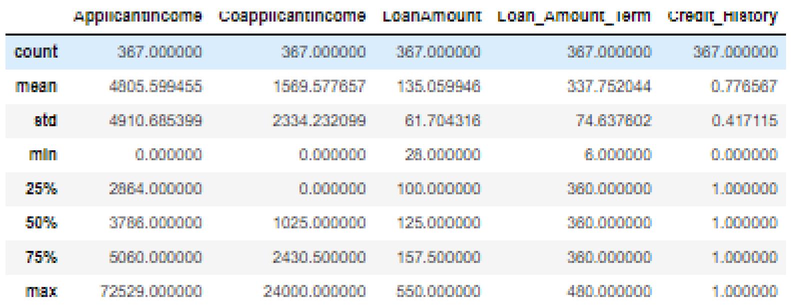
Descriptive Statistical:

Descriptive analysis is to study the basic features of data with the statistical process. Here pandas has a worthy function called describe. With this describe function we can understand the unique, top and frequent values of categorical features. And we can find mean, std, min, max and percentile values of continuous features.



```
data.describe()
```



	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amount_Term	Credit_History
count	367.000000	367.000000	367.000000	367.000000	367.000000
mean	4805.599455	1569.577657	135.059946	337.752044	0.776567
std	4910.685399	2334.232099	61.704316	74.637602	0.417115
min	0.000000	0.000000	28.000000	6.000000	0.000000
25%	2864.000000	0.000000	100.000000	360.000000	1.000000
50%	3786.000000	1025.000000	125.000000	360.000000	1.000000
75%	5060.000000	2430.500000	157.500000	360.000000	1.000000
max	72529.000000	24000.000000	550.000000	480.000000	1.000000

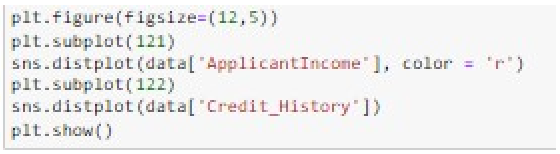
Visual Analysis:

Visual analysis is the process of using visual representations, such as charts, plots, and graphs, to explore and understand data. It is a way to quickly identify patterns, trends, and outliers in the data, which can help to gain insights and make informed decisions.

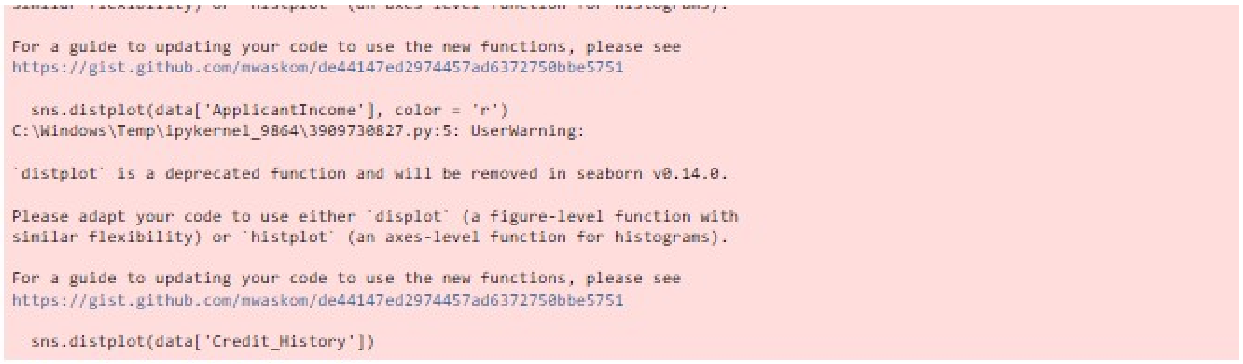
Univariate Analysis:

In simple words, univariate analysis is understanding the data with a single feature. Here we have displayed two different graphs such as distplot and countplot.

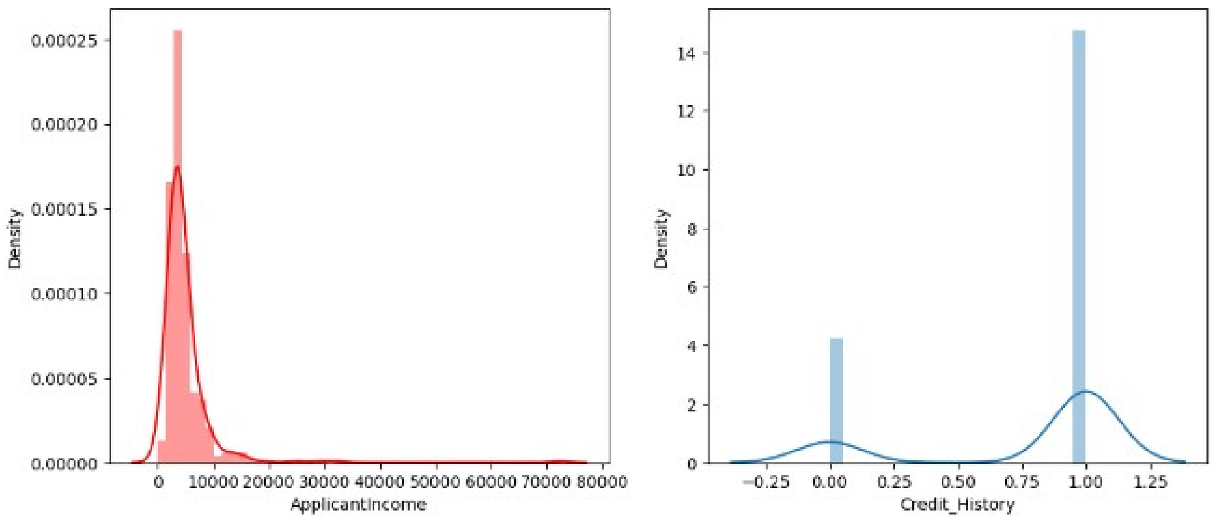
- The Seaborn package provides a wonderful function distplot. With the help of distplot, we can find the distribution of the feature. To make multiple graphs in a single plot, we use subplot.



```
plt.figure(figsize=(12,5))
plt.subplot(121)
sns.distplot(data['ApplicantIncome'], color = 'r')
plt.subplot(122)
sns.distplot(data['Credit_History'])
plt.show()
```

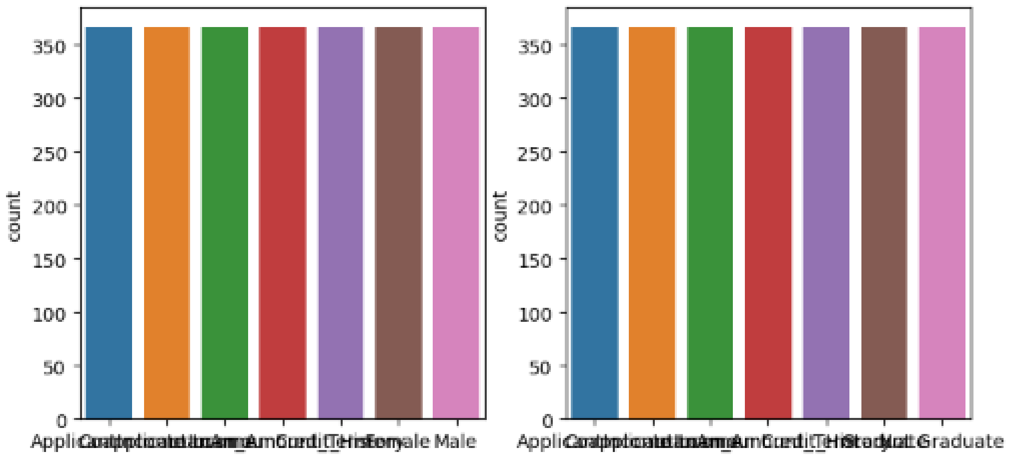


```
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(data['ApplicantIncome'], color = 'r')
C:\Windows\Temp\ipykernel_9864\3909730827.py:5: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(data['Credit_History'])
```



- In our dataset we have some categorical features. With the count plot function, we are going to count the unique category in those features. We have created a dummy data frame with categorical features. With for loop and subplot we have plotted this below graph.
- From the plot we came to know, Applicants income is skewed towards left side, where as credit history is categorical with 1.0 and 0.0

```
one_hot1 = pd.get_dummies(data['Education'])
# Drop column B as it is now encoded
df1 = data.drop('Education',axis = 1)
# Join the encoded df
df1 = df1.join(one_hot1)
df2=pd.get_dummies(data)
# data['Education']=one_hot1

# one_hot2 = pd.get_dummies(data['Married'])
# # Drop column B as it is now encoded
# df1 = data.drop('Married',axis = 1)
# # Join the encoded df
# df1 = df1.join(one_hot1)
# data['Married']=one_hot2

# one_hot4 = pd.get_dummies(data['Dependents'])
# # Drop column B as it is now encoded
# df1 = data.drop(' Dependents',axis = 1)
# # Join the encoded df
# df1 = df1.join(one_hot1)
# data[' Dependents']=one_hot4

# one_hot5 = pd.get_dummies(data['Self_Employed'])
# # Drop column B as it is now encoded
# df1 = data.drop('Self_Employed',axis = 1)
# # Join the encoded df
# df1 = df1.join(one_hot1)
# data[' Self_Employed']=one_hot5

# one_hot6 = pd.get_dummies(data['Property_Area z'])
# # Drop column B as it is now encoded
# df1 = data.drop('Property_Area      ',axis = 1)
# # Join the encoded df
# df1 = df1.join(one_hot1)
# data[' Property_Area      ']=one_hot6

plt.figure(figsize=(18,4))
plt.subplot(1,4,1)
sns.countplot(df)
plt.subplot(1,4,2)
sns.countplot(df1)
plt.show()
```

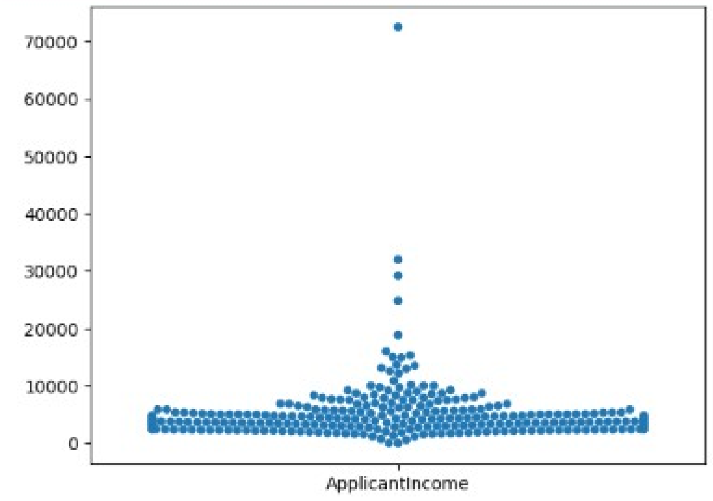


- Segmenting the gender column and married column based on bar graphs
- Segmenting the Education and Self-employed based on bar graphs ,for drawing insights such as educated people are employed
- Loan amount term based on the property area of a person holding

Multivariate Analysis:

In simple words, multivariate analysis is to find the relation between multiple features. Here we have used a swarm plot from the seaborn package.

```
C:\Users\GASCCS23\anaconda3\lib\site-packages\seaborn\categorical.py:3544: UserWarning: 33.2% of the points cannot be placed; y
ou may want to decrease the size of the markers or use stripplot.
  warnings.warn(msg, UserWarning)
```

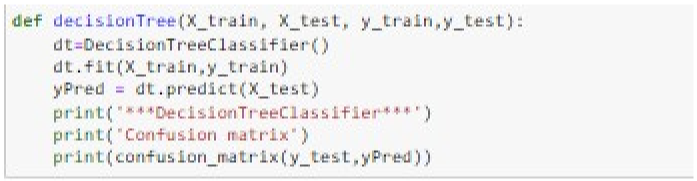


Training The Model In Multiple Algorithms:

Now our data is cleaned and it’s time to build the model. We can train our data on different algorithms. For this project we are applying four classification algorithms. The best model is saved based on its performance.

Decision Tree Model:

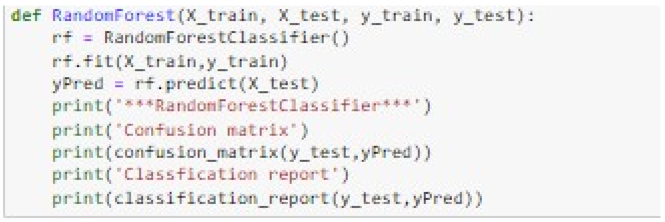
A function named decisionTree is created and train and test data are passed as the parameters. Inside the function, DecisionTreeClassifier algorithm is initialised and training data is passed to the model with the .fit() function. Test data is predicted with .predict() function and saved in a new variable. For evaluating the model, a confusion matrix and classification report is done.



```
def decisionTree(X_train, X_test, y_train,y_test):
    dt=DecisionTreeClassifier()
    dt.fit(X_train,y_train)
    yPred = dt.predict(X_test)
    print('***DecisionTreeClassifier***')
    print('Confusion matrix')
    print(confusion_matrix(y_test,yPred))
```

Random Forest Model:

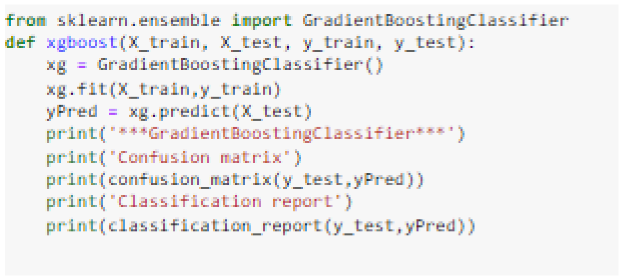
A function named randomForest is created and train and test data are passed as the parameters. Inside the function, RandomForestClassifier algorithm is initialised and training data is passed to the model with .fit() function. Test data is predicted with .predict() function and saved in a new variable. For evaluating the model, a confusion matrix and classification report is done.



```
def RandomForest(X_train, X_test, y_train, y_test):
    rf = RandomForestClassifier()
    rf.fit(X_train,y_train)
    yPred = rf.predict(X_test)
    print('***RandomForestClassifier***')
    print('Confusion matrix')
    print(confusion_matrix(y_test,yPred))
    print('Classfication report')
    print(classification_report(y_test,yPred))
```

KNN Model:

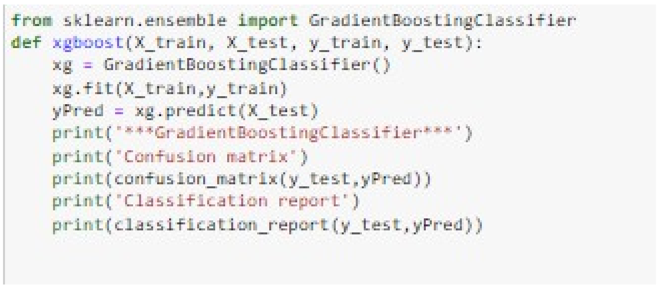
A function named KNN is created and train and test data are passed as the parameters. Inside the function, KNeighborsClassifier algorithm is initialised and training data is passed to the model with .fit() function. Test data is predicted with .predict() function and saved in new variable. For evaluating the model, confusion matrix and classification report is done.



```
from sklearn.ensemble import GradientBoostingClassifier
def xgboost(X_train, X_test, y_train, y_test):
    xg = GradientBoostingClassifier()
    xg.fit(X_train,y_train)
    yPred = xg.predict(X_test)
    print('***GradientBoostingClassifier***')
    print('Confusion matrix')
    print(confusion_matrix(y_test,yPred))
    print('Classification report')
    print(classification_report(y_test,yPred))
```

Xgboost Model:

A function named xgboost is created and train and test data are passed as the parameters. Inside the function, GradientBoostingClassifier algorithm is initialised and training data is passed to the model with .fit() function. Test data is predicted with .predict() function and saved in new variable. For evaluating the model, confusion matrix and classification report is done.



```
from sklearn.ensemble import GradientBoostingClassifier
def xgboost(X_train, X_test, y_train, y_test):
    xg = GradientBoostingClassifier()
    xg.fit(X_train,y_train)
    yPred = xg.predict(X_test)
    print('***GradientBoostingClassifier***')
    print('Confusion matrix')
    print(confusion_matrix(y_test,yPred))
    print('Classification report')
    print(classification_report(y_test,yPred))
```

ANN Model:

Building and training an Artificial Neural Network (ANN) using the Keras library with TensorFlow as the backend. The ANN is initialised as an instance of the Sequential class, which is a linear stack of layers. Then, the input layer and two hidden layers are added to the model using the Dense class, where the number of units and activation function are specified. The output layer is also added using the Dense class with a sigmoid activation function. The model is then compiled with the Adam optimizer, binary cross-entropy loss function, and accuracy metric. Finally, the model is fit to the training data with a batch size of 100, 20% validation split, and 100 epochs.