```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from scipy.stats import skew
data = pd.read_csv("D:\IBM\Mall_Customers.csv")
data.head()
data.tail()
data.describe()
#univariate analysis
data.hist(figsize=(20,10), grid=False, layout=(2,4), bins=30)
plt.show()
plt.figure(figsize=(10,5))
sns.distplot(data['Age'])
plt.show()
sns.countplot(data['Age'])
sns.countplot(data['Gender'])
plt.hist(data['Annual Income (k$)'])
#Bivariate Analysis
sns.stripplot(x=data['Age'],y=data['Annual Income (k$)'])
fig, axes = plt.subplots(4,2, figsize=(15,10))
axes = axes.flatten()
for i in range(1,len(data.columns)-1):
    sns.scatterplot(x=data.iloc[:,i], y=data['Age'], ax=axes[i])
plt.show()
plt.figure(figsize=(10,5))
sns.boxenplot(y=data['Age'], x=data['Gender'])
plt.grid()
plt.show()
data.groupby('Gender')['Age'].describe()
plt.scatter(data['Age'],data['Annual Income (k$)'],color='blue')
plt.xlabel("Age")
plt.ylabel("Annual Income (k$)")
#Multivariate Analysis
sns.pairplot(data)
plt.show()
plt.figure(figsize=(10,5))
sns.heatmap(data.corr(), annot=True)
plt.show()
#Descriptive Statistics

#mean

data.mean()
# median

data.median()
#mode

data.mode()
data['Gender'].value_counts()
data.shape
data.isnull().sum()
#Missing values and deal with them
data.isna()
data.isna().any()
#skewness
```

```
data.skew()
print(sns.distplot(data['Age']))
data.kurt()
data.var()
data.std()
#Find the outliers and replace them outliers
sns.boxplot(data['Annual Income (k$)'])
qnt=data.quantile(q=(0.30,0.45))
qnt
iqr =qnt.loc[0.45]-qnt.loc[0.30] #iqr calculation
iqr
#lower extreme values
lower=qnt.loc[0.30]-1.5*iqr
lower
#upper extreme values
upper=qnt.loc[0.45]+1.5*iqr
upper
data['CustomerID']=np.where(data['CustomerID']>45,31,data['CustomerID'])
sns.boxplot(data['Annual Income (k$)'])
#Encoding Categorical Values
numeric_data = data.select_dtypes(include=[np.number])
categorical_data = data.select_dtypes(exclude=[np.number])
print("Number of numerical variables: ", numeric_data.shape[1])
print("Number of categorical variables: ", categorical_data.shape[1])
print("Number of categorical variables: ", categorical_data.shape[1])
Categorical_variables = list(categorical_data.columns)
Categorical_variables
data['Gender'].value_counts()
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
label = le.fit_transform(data['Gender'])
data["Gender"] = label
data['Gender'].value_counts()
#Scaling the data
X = data.drop("Age",axis=1)
Y = data['Age']
from sklearn.preprocessing import StandardScaler
object= StandardScaler()
scale = object.fit_transform(X)
print(scale)
#Clustering Algorithm
x = data.iloc[:, [3, 4]].values
from sklearn.cluster import KMeans
wcss_list= []
for i in range(1, 11):
    kmeans = KMeans(n_clusters=i, init='k-means++', random_state= 42)
    kmeans.fit(x)
    wcss_list.append(kmeans.inertia_)
plt.plot(range(1, 11), wcss_list)
plt.title('The Elobw Method Graph')
plt.xlabel('Number of clusters(k)')
plt.ylabel('wcss_list')
plt.show()
#training the K-means model on a dataset

kmeans = KMeans(n_clusters=5, init='k-means++', random_state= 42)
y_predict= kmeans.fit_predict(x)
```

```
plt.scatter(x[y_predict == 0, 0], x[y_predict == 0, 1], s = 100, c =
'blue', label = 'Cluster 1') #for first cluster
plt.scatter(x[y_predict == 1, 0], x[y_predict == 1, 1], s = 100, c =
'green', label = 'Cluster 2') #for second cluster
plt.scatter(x[y_predict== 2, 0], x[y_predict == 2, 1], s = 100, c =
'red', label = 'Cluster 3') #for third cluster
plt.scatter(x[y_predict == 3, 0], x[y_predict == 3, 1], s = 100, c =
'cyan', label = 'Cluster 4') #for fourth cluster
plt.scatter(x[y_predict == 4, 0], x[y_predict == 4, 1], s = 100, c =
'magenta', label = 'Cluster 5') #for fifth cluster
plt.scatter(kmeans.cluster_centers_[:, 0], kmeans.cluster_centers_[:, 1],
s = 300, c = 'yellow', label = 'Centroid')
plt.title('Clusters of customers')
plt.xlabel('Annual Income (k$)')
plt.ylabel('Spending Score (1-100)')
plt.legend()
plt.show()
#Split the data into dependent and independent variables.
#target variable
y=data['Age']
y.head()
#independent
x=data.drop(columns=['Age'],axis=1)
x.head()
data=pd.get_dummies(data,columns=['Age'])
data.head()
#encoding
data = pd.get_dummies(data, drop_first=True)
data.head()
#Split the data into training and testing
from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test =
train_test_split(x,y,test_size=0.2,random_state=0)
x_train.shape
x_test.shape
y_train.shape
y_test.shape
#Build, test, train the Model
from sklearn.tree import DecisionTreeClassifier
#initializing the DT

model=DecisionTreeClassifier()
#encoding

data = pd.get_dummies(data, drop_first=True)
data.head()
X = data.drop('Gender', axis=1)
y = data['Gender']

from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33)

from sklearn.preprocessing import StandardScaler
ss = StandardScaler()

X_trains = ss.fit_transform(X_train)
X_tests = ss.transform(X_test)
#Base model
```

```
from sklearn.linear_model import LinearRegression
lr = LinearRegression()

lr.fit(X_trains, y_train)
pred = lr.predict(X_tests)

from sklearn.metrics import r2_score, roc_auc_score, mean_squared_error
rmse = np.sqrt(mean_squared_error(y_test, pred))
r2 = r2_score(y_test, pred)

print("The root mean Sq error calculated from the base model is:",rmse)
print("The r2-score is:",r2)
#selecting best feautre

from sklearn.feature_selection import RFE
lr = LinearRegression()
n = [{'n_features_to_select':list(range(1,10))}]
rfe = RFE(lr)

from sklearn.model_selection import GridSearchCV
gsearch = GridSearchCV(rfe, param_grid=n, cv=3)
gsearch.fit(X, y)

gsearch.best_params_lr = LinearRegression()
rfe = RFE(lr, n_features_to_select=8)
rfe.fit(X,y)

pd.DataFrame(rfe.ranking_, index=X.columns, columns=['Gender'])
#Measure the performance using Evaluation Metrics
from sklearn.neighbors import KNeighborsRegressor
from sklearn.ensemble import  RandomForestRegressor
from sklearn.linear_model import LinearRegression
from sklearn.ensemble import  GradientBoostingRegressor
from sklearn.linear_model import  Ridge
from sklearn.svm import SVR
from sklearn import model_selection
from sklearn.model_selection import cross_val_predict

models = [   SVR(),
             RandomForestRegressor(),
             GradientBoostingRegressor(),
             KNeighborsRegressor(n_neighbors = 4)]
results = []
names = ['SVM','Random Forest','Gradient Boost','K-Nearest Neighbors']
for model,name in zip(models,names):
    kfold = model_selection.KFold(n_splits=10)
    cv_results = model_selection.cross_val_score(model, X_train, y_train, 
cv=kfold)
    rmse = np.sqrt(mean_squared_error(y, cross_val_predict(model, X , y, 
cv=3)))
    results.append(rmse)
    names.append(name)
    msg = "%s: %f" % (name, rmse)
    print(msg)
```