```
from google.colab import drive
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True).
```

```
%cd /content/drive/MyDrive/Train.csv
```

```
[Errno 20] Not a directory: '/content/drive/MyDrive/Train.csv'
/content
```

```
!ls
```

```
DataForML.pkl  drive  Final_XGB_Model.pkl  sample_data
```

STEP 1

- Connecting pathway to csv file (Google Drive
- Removing duplicate rows

```
# Supressing the warning messages
import warnings
warnings.filterwarnings('ignore')
```

```
# Reading dataset
import pandas as pd
import numpy as np
EcommerceData = pd.read_csv('/content/drive/MyDrive/Train.csv', encoding="latin")
print('Shape before deleting duplicate values:',  EcommerceData.shape)

#Removing duplicate rows if any exists
EcommerceData=EcommerceData.drop_duplicates()
print('Shape After deleting duplicate values', EcommerceData.shape)

#printing Sample Data
#Start Observing the Quantitative/Categorical/Qualitative variables
EcommerceData.head(10)
```

```
Shape before deleting duplicate values: (2452, 8)
Shape After deleting duplicate values (2452, 8)
```

	Product	Product_Brand	Item_Category	Subcategory_1	Subcategory_2	Item_Rating	Date	Selling_Price
0	P-2610	B-659	bags wallets belts	bags	hand bags	4.3	2/3/2017	291
1	P-2453	B-3078	clothing	women s clothing	western wear	3.1	7/1/2015	897
2	P-6802	B-1810	home decor festive needs	showpieces	ethnic	3.5	1/12/2019	792
3	P-4452	B-3078	beauty and personal care	eye care	h2o plus eye care	4.0	12/12/2014	837
4	P-8454	B-3078	clothing	men s clothing	t shirts	4.3	12/12/2013	470
5	P-5597	B-1487	home decor festive needs	table decor handicrafts	showpieces	5.0	9/4/2020	746
6	P-8398	B-3078	footwear	women s footwear	casual shoes	4.1	4/12/2017	1798
7	P-10744	B-2830	kitchen dining	cookware	pots pans	3.1	1/12/2013	955
8	P-4042	B-1045	home decor festive needs	wall decor clocks	paintings	2.4	18/3/2019	21770
9	P-360	B-88	automotive	accessories spare parts	car interior exterior	2.3	10/5/2018	199

Next steps: Generate code with EcommerceData | View recommended plots | New interactive sheet

Observation Above : There are 2452 Ecommerce Price Predictions

Removing the object variables

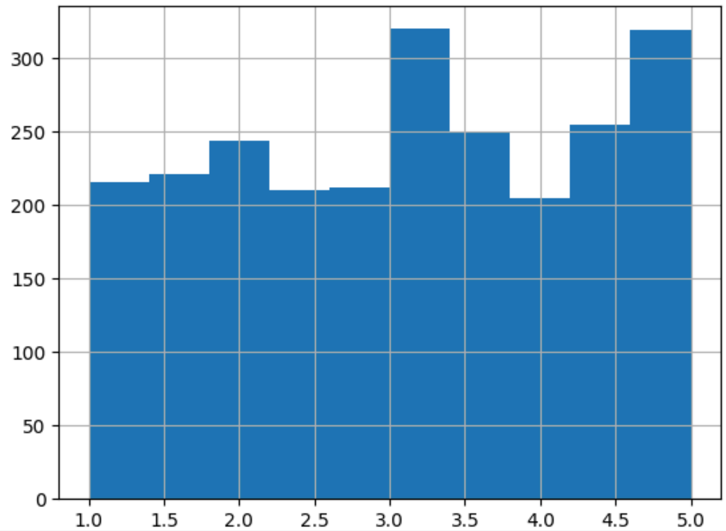
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

#List of all columns to remove
columns_to_remove = ['Product','Product_Brand','Item_Category','Subcategory_1','Subcategory_2','Date']

#Check existence of the columns
for col in columns_to_remove:
  if col in EcommerceData.columns:
    EcommerceData = EcommerceData.drop(columns=col)
    print(f"Removed column: {col}")
  else:
    print(f"Column not found in the DataFrame: {col}")

  #View data frame after attempt to drop columns
  print(EcommerceData.head())
```

```
Removed column: Product
  Product_Brand            Item_Category     Subcategory_1  \
0         B-659        bags wallets belts              bags
1        B-3078                  clothing  women s clothing
2        B-1810  home decor festive needs        showpieces
3        B-3078  beauty and personal care          eye care
4        B-3078                  clothing    men s clothing

      Subcategory_2  Item_Rating        Date  Selling_Price
0         hand bags          4.3    2/3/2017            291
1      western wear          3.1    7/1/2015            897
2            ethnic          3.5   1/12/2019            792
3  h2o plus eye care         4.0  12/12/2014            837
4          t shirts          4.3  12/12/2013            470
Removed column: Product_Brand
              Item_Category     Subcategory_1      Subcategory_2  Item_Rating  \
0        bags wallets belts              bags          hand bags          4.3
1                  clothing  women s clothing       western wear          3.1
2  home decor festive needs        showpieces             ethnic          3.5
3  beauty and personal care          eye care  h2o plus eye care          4.0
4                  clothing    men s clothing           t shirts          4.3

         Date  Selling_Price
0    2/3/2017            291
1    7/1/2015            897
2   1/12/2019            792
3  12/12/2014            837
4  12/12/2013            470
Removed column: Item_Category
      Subcategory_1      Subcategory_2  Item_Rating        Date  Selling_Price
0              bags          hand bags          4.3    2/3/2017            291
1  women s clothing       western wear          3.1    7/1/2015            897
2        showpieces             ethnic          3.5   1/12/2019            792
3          eye care  h2o plus eye care          4.0  12/12/2014            837
4    men s clothing           t shirts          4.3  12/12/2013            470
Removed column: Subcategory_1
      Subcategory_2  Item_Rating        Date  Selling_Price
0         hand bags          4.3    2/3/2017            291
1      western wear          3.1    7/1/2015            897
2            ethnic          3.5   1/12/2019            792
3  h2o plus eye care         4.0  12/12/2014            837
4          t shirts          4.3  12/12/2013            470
Removed column: Subcategory_2
   Item_Rating        Date  Selling_Price
0          4.3    2/3/2017            291
1          3.1    7/1/2015            897
2          3.5   1/12/2019            792
3          4.0  12/12/2014            837
4          4.3  12/12/2013            470
Removed column: Date
   Item_Rating  Selling_Price
0          4.3            291
1          3.1            897
2          3.5            792
3          4.0            837
4          4.3            470
```

STEP 4

```
%matplotlib inline
EcommerceData['Item_Rating'].hist()
```

<Axes: >



STEP 5

```
EcommerceData.head()
```

	Item_Rating	Selling_Price
0	4.3	291
1	3.1	897
2	3.5	792
3	4.0	837
4	4.3	470

Next steps: Generate code with EcommerceData View recommended plots New interactive sheet

```
EcommerceData.tail()
```

	Item_Rating	Selling_Price
2447	2.3	741
2448	1.9	1590
2449	1.9	995
2450	2.7	1598
2451	4.1	397

```
EcommerceData.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2452 entries, 0 to 2451
Data columns (total 2 columns):
 #   Column         Non-Null Count  Dtype  
---  ------         --------------  -----  
 0   Item_Rating    2452 non-null   float64
 1   Selling_Price  2452 non-null   int64  
dtypes: float64(1), int64(1)
memory usage: 38.4 KB
```

```
EcommerceData.describe(include='all')
```

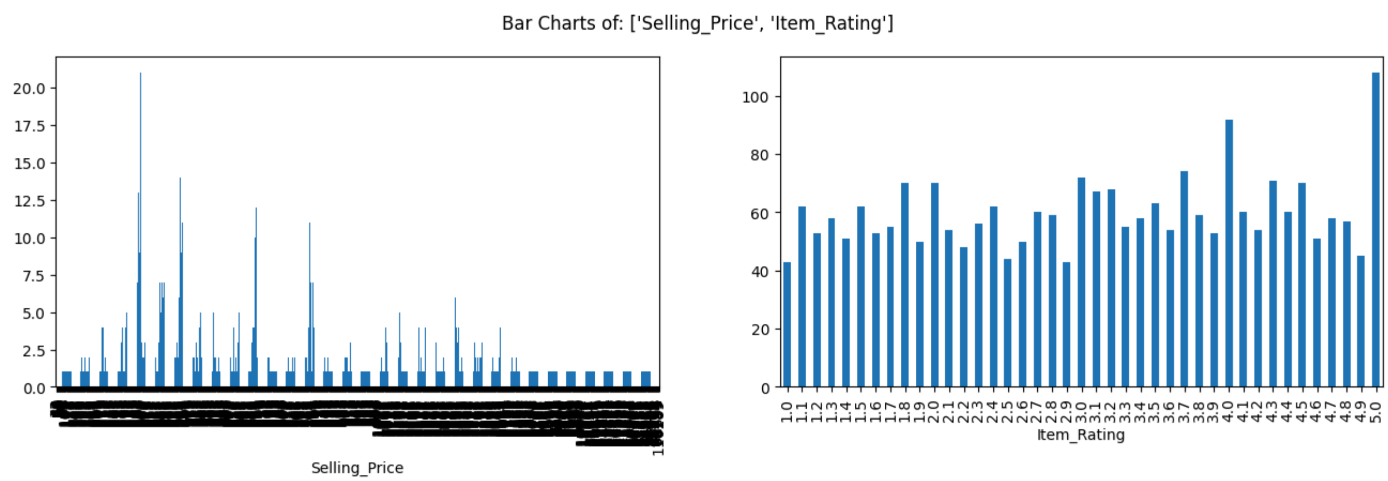
	Item_Rating	Selling_Price
count	2452.000000	2452.000000
mean	3.078467	2494.375612
std	1.187137	7115.256516
min	1.000000	33.000000
25%	2.000000	371.000000
50%	3.100000	596.000000
75%	4.100000	1195.250000
max	5.000000	116289.000000

```
EcommerceData.nunique()
```

	0
Item_Rating	41
Selling_Price	1095

STEP 8

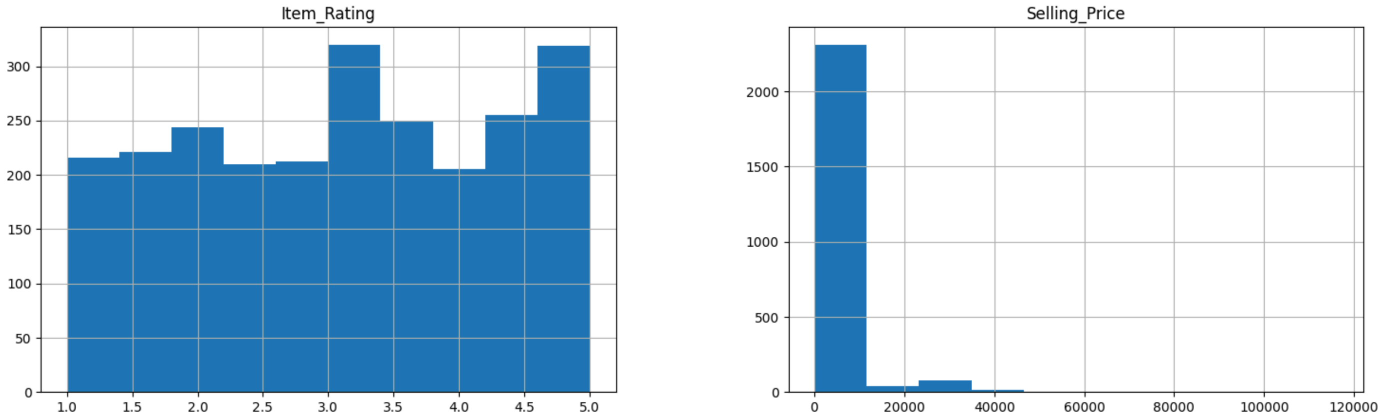
```
def PlotBarCharts(inpData, colsPlot):
    %matplotlib inline

    import matplotlib.pyplot as plt

    # Generating multiple subplots
    fig, subPlot = plt.subplots(nrows=1, ncols=len(colsPlot), figsize=(16,4))
    fig.suptitle('Bar Charts of: '+ str(colsPlot))

    for colName, plotNumber in zip(colsPlot, range(len(colsPlot))):
        if colName in inpData.columns:  # Check if column exists in DataFrame
            inpData.groupby(colName).size().plot(kind='bar', ax=subPlot[plotNumber])
        else:
            print(f"Column '{colName}' not found in DataFrame. Skipping.")


PlotBarCharts(inpData=EcommerceData, colsPlot=['Selling_Price','Item_Rating']) # Use 'colsPlot' to match the function definition
```



STEP 9

```
EcommerceData.hist(['Item_Rating','Selling_Price'], figsize=(18,5))
```

```
array([[<Axes: title={'center': 'Item_Rating'}>,
        <Axes: title={'center': 'Selling_Price'}>]], dtype=object)
```



STEP 10

```
EcommerceData['Selling_Price'][EcommerceData['Selling_Price']<60].sort_values(ascending=False)
```

	Selling_Price
1788	42
1950	33

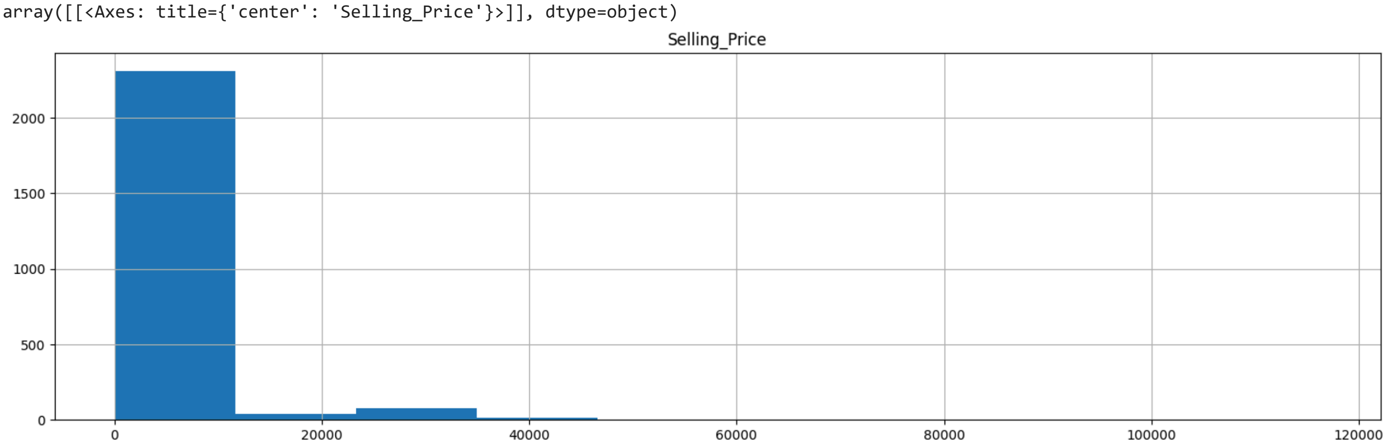
dtype: int64

```
EcommerceData['Selling_Price'][EcommerceData['Selling_Price']<60] = 51.13
```

STEP 11

```
EcommerceData.hist(['Selling_Price'], figsize=(18,5))
```

```
array([[<Axes: title={'center': 'Selling_Price'}>]], dtype=object)
```



STEP 12

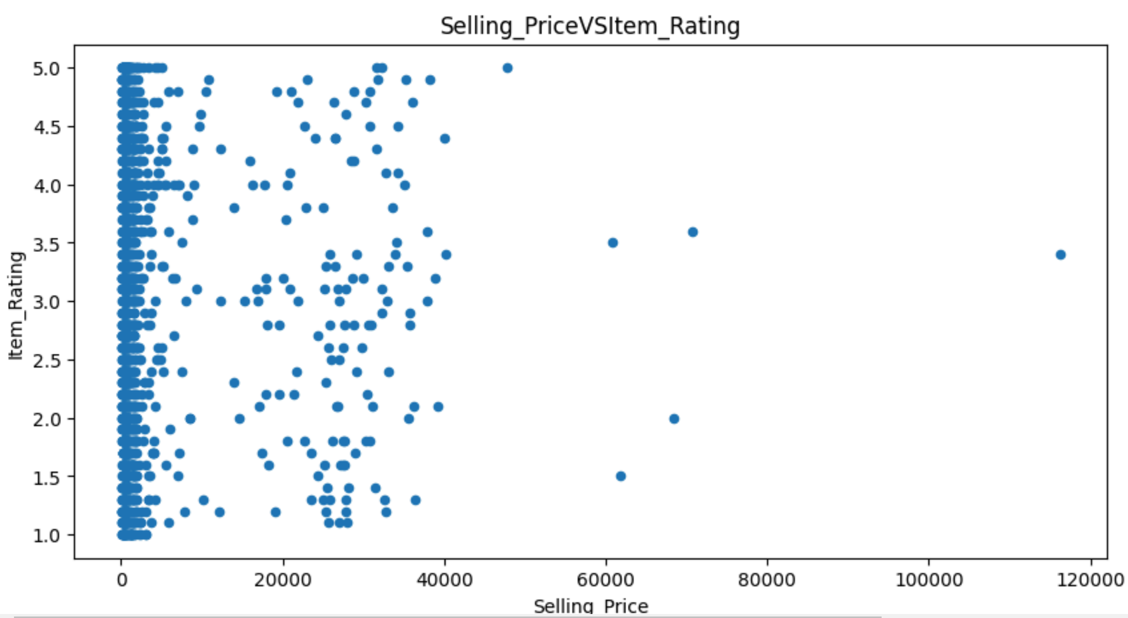
```
EcommerceData.isnull().sum()
```

	0
Item_Rating	0
Selling_Price	0

dtype: int64

STEP 13

```
ContinuousCols = ['Selling_Price']

#Plotting Scatter Charts for each predictor vs the target variable
for predictor in ContinuousCols:
  EcommerceData.plot.scatter(x=predictor, y='Item_Rating', figsize=(10,5), title=predictor+"VS"+'Item_Rating')
```



STEP 14

```
#Calculating Correlation Matrix
ContinuousCols=['Item_Rating','Selling_Price'] # Only include numeric columns

#Creating the Correlation Matrix
CorrelationData=EcommerceData[ContinuousCols].corr()
CorrelationData
```

	Item_Rating	Selling_Price
Item_Rating	1.000000	-0.013686
Selling_Price	-0.013686	1.000000

Next steps: Generate code with CorrelationData | View recommended plots | New interactive sheet

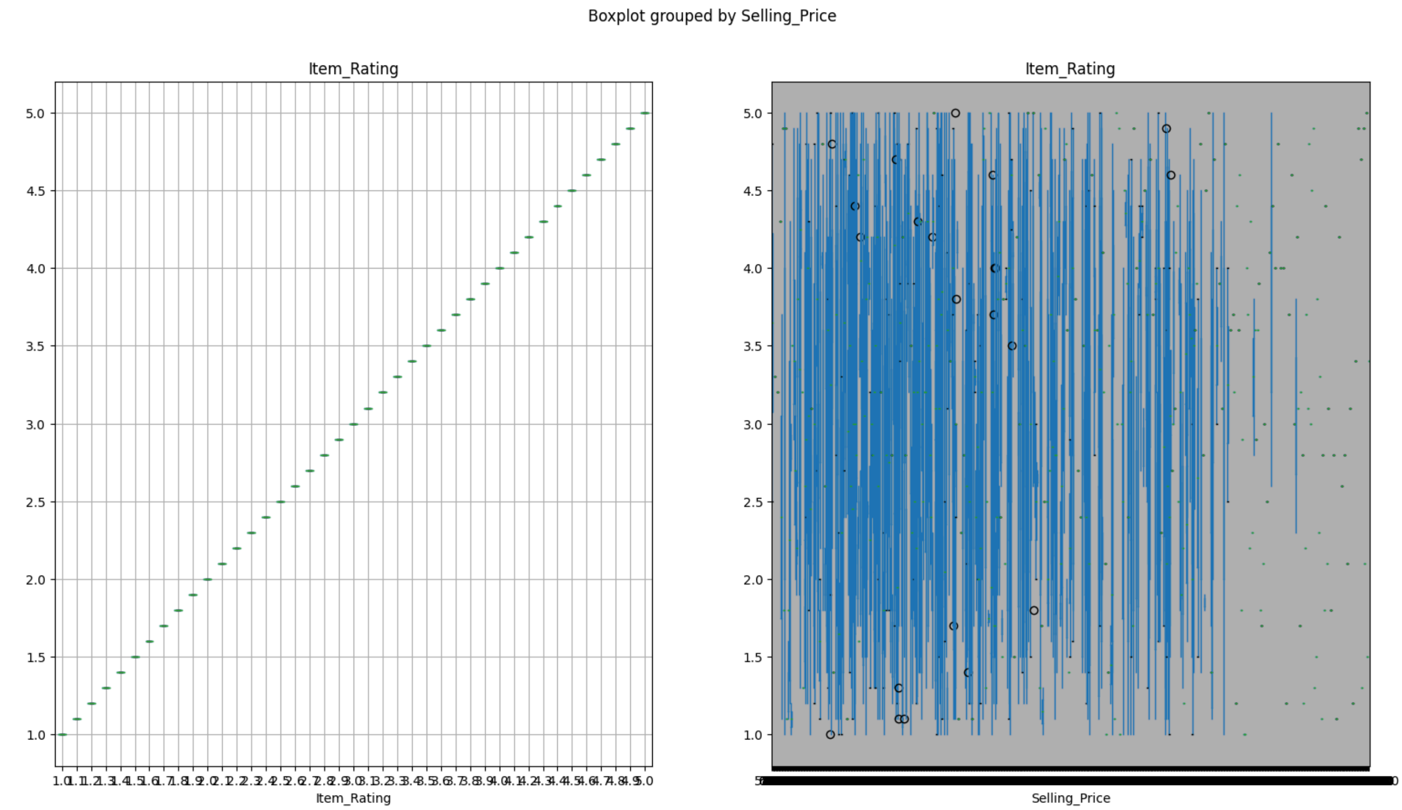
```
CorrelationData['Item_Rating'][abs(CorrelationData['Item_Rating'])>0.5]
```

	Item_Rating
Item_Rating	1.0

dtype: float64

** STEP 15**

```
CatrgoricalCols=['Item_Rating','Selling_Price']

import matplotlib.pyplot as plt
fig, PlotCanvas=plt.subplots(nrows=1, ncols=len(CatrgoricalCols), figsize=(18,10)) # Use CatrgoricalCols

#Creating box plots for each continuous predictor against the Target variable "Item_Rating"
for PredictorCol , i in zip(CatrgoricalCols, range(len(CatrgoricalCols))):
  EcommerceData.boxplot(column='Item_Rating', by=PredictorCol, figsize=(5,5), vert=True, ax=PlotCanvas[i]) # Correct typo: colimn -> column
```



STEP 16

```
def FunctionAnova (inpData, TargetVariable, CategoricalPredictorList):
  from scipy.stats import f_oneway

# Creating an empty list of final selected predictors
  SelectedPredictors=[]

  print('##### ANOVA Results ##### ‡ \n')
  for predictor in CategoricalPredictorList:
    # Check if the predictor column exists in the DataFrame
    if predictor in inpData.columns:
        CategoryGroupLists=inpData.groupby(predictor) [TargetVariable].apply(list)
        AnovaResults = f_oneway(*CategoryGroupLists)

    # If the ANOVA P-Value is <0.05, that means we reject HO
        if (AnovaResults [1] < 0.05):
          print(predictor, 'is correlated with', TargetVariable, '| P-Value:', AnovaResults [1])
          SelectedPredictors. append (predictor)
        else:
          print(predictor, 'is NOT correlated with', TargetVariable, '| P-Value:', AnovaResults [1])
    else:
```

```
        print(f"Skipping predictor '{predictor}' as it is not present in the DataFrame.")

  return (SelectedPredictors)

CategoricalPredictorList=['Product','Selling_Price']
FunctionAnova(inpData=EcommerceData,
              TargetVariable='Item_Rating', CategoricalPredictorList=CategoricalPredictorList)
```

```
##### ANOVA Results ##### ‡

Skipping predictor 'Product' as it is not present in the DataFrame.
Selling_Price is NOT correlated with Item_Rating | P-Value: 0.8437906633097767
[]
```

```
SelectedColumns=['Item_Rating', 'Selling_Price']
# Selecting final columns
DataForML=EcommerceData[SelectedColumns]
DataForML. head ()
```

	Item_Rating	Selling_Price
0	4.3	291.0
1	3.1	897.0
2	3.5	792.0
3	4.0	837.0
4	4.3	470.0

```
DataForML.to_pickle( 'DataForML.pkl')
```

STEP 17

```
# Treating all the nominal variables at once using dummy variables
DataForML_Numeric=pd.get_dummies (DataForML)

# Adding Target Variable to the data
DataForML_Numeric ['Item_Rating']=EcommerceData['Item_Rating']

# Printing sample rows
DataForML_Numeric.head ()
```

	Item_Rating	Selling_Price
0	4.3	291.0
1	3.1	897.0
2	3.5	792.0
3	4.0	837.0
4	4.3	470.0

STEP 18

```
DataForML_Numeric.columns
```

```
Index(['Item_Rating', 'Selling_Price'], dtype='object')
```

```
TargetVariable='Item_Rating'
Predictors=['Item_Rating','Selling_Price'] # Removed 'datetime'

X=DataForML_Numeric[Predictors].values
y=DataForML_Numeric[TargetVariable].values

# Split the data into training and testing set
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=428)
```

STEP 19

```
### Standardization of data ###
from sklearn.preprocessing import StandardScaler, MinMaxScaler

PredictorScaler=MinMaxScaler()

PredictorScalerFit=PredictorScaler.fit(X)

X=PredictorScalerFit.transform (X)

from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)


# Sanity check for the sampled data
print(X_train.shape)
print(y_train.shape)
print(X_test.shape)
print(y_test.shape)
```

```
(1716, 2)
(1716,)
(736, 2)
(736,)
```

STEP 20

Multilinear Regression

```
#Multiple Linear Regression
from sklearn.linear_model import LinearRegression
RegModel = LinearRegression()

# Printing all the parameters of Linear regression
print (RegModel)

# Creating the model on Training Data
LREG=RegModel.fit(X_train,y_train)
prediction=LREG.predict(X_test)

from sklearn import metrics
# Measuring Goodness of fit in Training data
print('R2 Value:' ,metrics.r2_score(y_train, LREG.predict(X_train)))

#####################################################
print('\n##### Model Validation and Accuracy Calculations ###########')

#Printing some sample values of prediction
TestingDataResults=pd. DataFrame(data=X_test, columns=Predictors)
TestingDataResults [TargetVariable]=y_test
TestingDataResults [('Predicted'+TargetVariable)]=np. round (prediction) # Correct column name for predicted ratings

# Printing sample prediction values
print (TestingDataResults. head())

# Calculating the error for each row
TestingDataResults ['Selling_Price']=100 * ((abs(
 TestingDataResults ['Item_Rating']-TestingDataResults ['Predicted'+ TargetVariable] ))/TestingDataResults['Item_Rating']) # Use the correct 

MAPE=np.mean(TestingDataResults ['Selling_Price' ]) # Assuming 'low' was a typo and should be 'Selling_Price'
MedianMAPE=np. median (TestingDataResults ['Selling_Price']) # Assuming 'low' was a typo and should be 'Selling_Price'

Accuracy =100 - MAPE
MedianAccuracy=100- MedianMAPE
print('Mean Accuracy on test data:', Accuracy) # Can be negative sometimes due to outlier
print('Median Accuracy on test data:', MedianAccuracy)

# Defining a custom function to calculate accuracy
# Make sure there are no zeros in the Target variable if you are using MAPE
def Accuracy_Score(orig, pred):
  MAPE = np.mean (100 * (np. abs (orig-pred)/orig))
  #print ('#'*70, 'Accuracy:', 100-MAPE)
  return (100-MAPE)

# Custom Scoring MAPE calculation
from sklearn.metrics import make_scorer
```

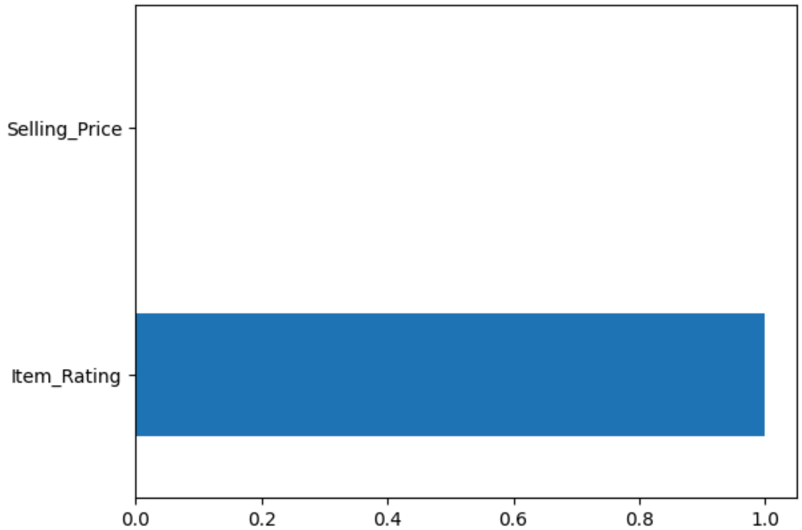
```
custom_Scoring=make_scorer (Accuracy_Score, greater_is_better=True)

# Importing cross validation function from sklearn
from sklearn.model_selection import cross_val_score

# Running 10-Fold Cross validation on a given algorithm
# Passing full data X and y because the K-fold will split the data and automatically choose train/test
Accuracy_Values=cross_val_score(RegModel, X, y, cv=10, scoring=custom_Scoring)
print('\nAccuracy values for 10-fold Cross Validation:\n', Accuracy_Values)
print('\nFinal Average Accuracy of the model:', round(Accuracy_Values.mean(),2))
```

```
LinearRegression()
R2 Value: 1.0

##### Model Validation and Accuracy Calculations ############
   Item_Rating  Selling_Price  PredictedItem_Rating
0         1.2       0.005531                   1.0
1         4.9       0.003784                   5.0
2         4.5       0.293956                   5.0
3         1.1       0.001806                   1.0
4         1.7       0.006881                   2.0
Mean Accuracy on test data: 90.66324889798534
Median Accuracy on test data: 91.30434782608695

Accuracy values for 10-fold Cross Validation:
 [100. 100. 100. 100. 100. 100. 100. 100. 100. 100.]

Final Average Accuracy of the model: 100.0
```

Decision Tree Regression

```
# Decision Trees (Multiple if-else statements!)
from sklearn.tree import DecisionTreeRegressor
RegModel = DecisionTreeRegressor (max_depth=5, criterion='friedman_mse') # Removed the extra space before 'friedman_mse'
# Good Range of Max_ depth = 2 to 20

# Printing all the parameters of Decision Tree
print (RegModel)

# Creating the model on Training Data
DT=RegModel. fit (X_train,y_train)
prediction=DT. predict(X_test)

from sklearn import metrics
# Measuring Goodness of fit in Training data
print('R2 Value: ',metrics.r2_score(y_train, DT.predict(X_train)))

# Plotting the feature importance for Top 10 most important columns
%matplotlib inline
feature_importances = pd.Series (DT. feature_importances_, index=Predictors)
feature_importances.nlargest(10).plot(kind='barh')
######################################################################################
print('\n##### Model Validation and Accuracy Calculations ###########')

# Printing some sample values of prediction
TestingDataResults=pd. DataFrame(data=X_test, columns=Predictors)
TestingDataResults [TargetVariable]=y_test
TestingDataResults [('Predicted'+TargetVariable)]=np. round (prediction)

#Printing sample prediction values
print(TestingDataResults. head ())

# Calculating the error for each row
TestingDataResults['Selling_Price'] = 100 * ( (abs(
 TestingDataResults['Item_Rating'] - TestingDataResults['Predicted' + TargetVariable]))/TestingDataResults['Item_Rating']) # Use the correct

MAPE = np.mean(TestingDataResults['Selling_Price'])
MedianMAPE=np. median (TestingDataResults ['Selling_Price'])

Accuracy =100 - MAPE
MedianAccuracy=100- MedianMAPE
print ('Mean Accuracy on test data:', Accuracy) # Can be negative sometimes due to outlier
print( 'Median Accuracy on test data:', MedianAccuracy)

# Defining a custom function to calculate accuracy
# Make sure there are no zeros in the Target variable if you are using MAPE
def Accuracy_Score(orig,pred):
  MAPE = np.mean (100 * (np-abs (orig-pred)/orig))
  #print ('#**70, 'Accuracy:', 100-MAPE)
  return ( 100-MAPE)

# Custom Scoring MAPE calculation
from sklearn.metrics import make_scorer
custom_Scoring = make_scorer(Accuracy_Score, greater_is_better=True)

# Importing cross validation function from sklearn
from sklearn.model_selection import cross_val_score

# Running 10-Fold Cross validation on a given algorithm
# Passing full data X and y because the K-fold will split the data and automatically choose train/tes
Accuracy_Values=cross_val_score(RegModel, X, y, cv=10, scoring=custom_Scoring)
print('\nAccuracy values for 10-fold Cross Validation: \n', Accuracy_Values)
print('InFinal Average Accuracy of the model:', round(Accuracy_Values.mean (), 2))
```

```
DecisionTreeRegressor(criterion='friedman_mse', max_depth=5)
R2 Value:  0.999277420777931

##### Model Validation and Accuracy Calculations ############
   Item_Rating  Selling_Price  PredictedItem_Rating
0          1.2       0.005531                   1.0
1          4.9       0.003784                   5.0
2          4.5       0.293956                   4.0
3          1.1       0.001806                   1.0
4          1.7       0.006881                   2.0
Mean Accuracy on test data: 90.66324889798534
Median Accuracy on test data: 91.30434782608695

Accuracy values for 10-fold Cross Validation:
 [nan nan nan nan nan nan nan nan nan nan]
InFinal Average Accuracy of the model: nan
```



Plotting Decision Tree

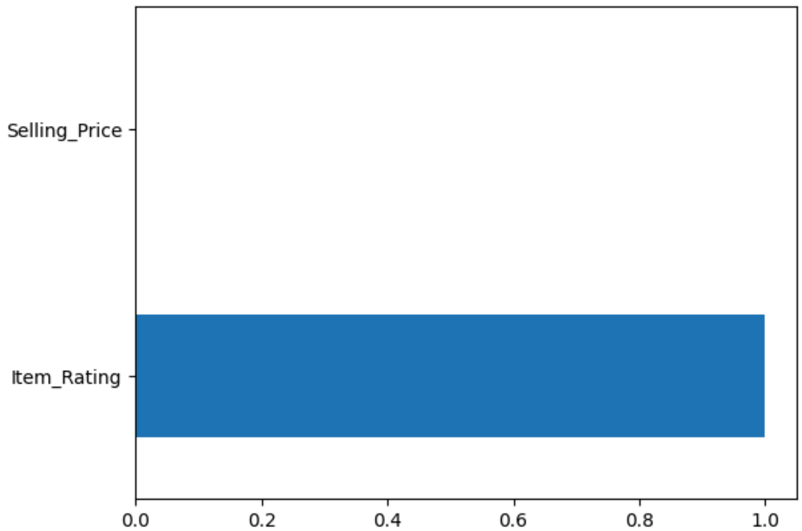
```
# Load libraries
from IPython.display import Image
# Import sklearn here
import sklearn
from sklearn import tree
import pydotplus
# Import the ensemble module explicitly
from sklearn import ensemble

# Ensure 'RegModel' is a RandomForestRegressor before proceeding
# Use sklearn.ensemble here instead of just sklearn
if isinstance(RegModel, ensemble.RandomForestRegressor): # Use ensemble here
    # Access the first tree in the forest as an example
    tree_to_visualize = RegModel.estimators_[0]

    # Create DOT data for the selected tree
    dot_data = tree.export_graphviz(tree_to_visualize, out_file=None,
                                    feature_names=Predictors)

    # printing the rules
    #print(dot_data)

    # Draw graph
    graph = pydotplus.graph_from_dot_data(dot_data)

    # Show graph
    Image(graph.create_png(), width=10000,height=5000)
    # Double-click on the graph to zoom in
else:
    print("Error: RegModel is not a RandomForestRegressor. Visualization skipped.")
```

```
Error: RegModel is not a RandomForestRegressor. Visualization skipped.
```

Random Forest Regression

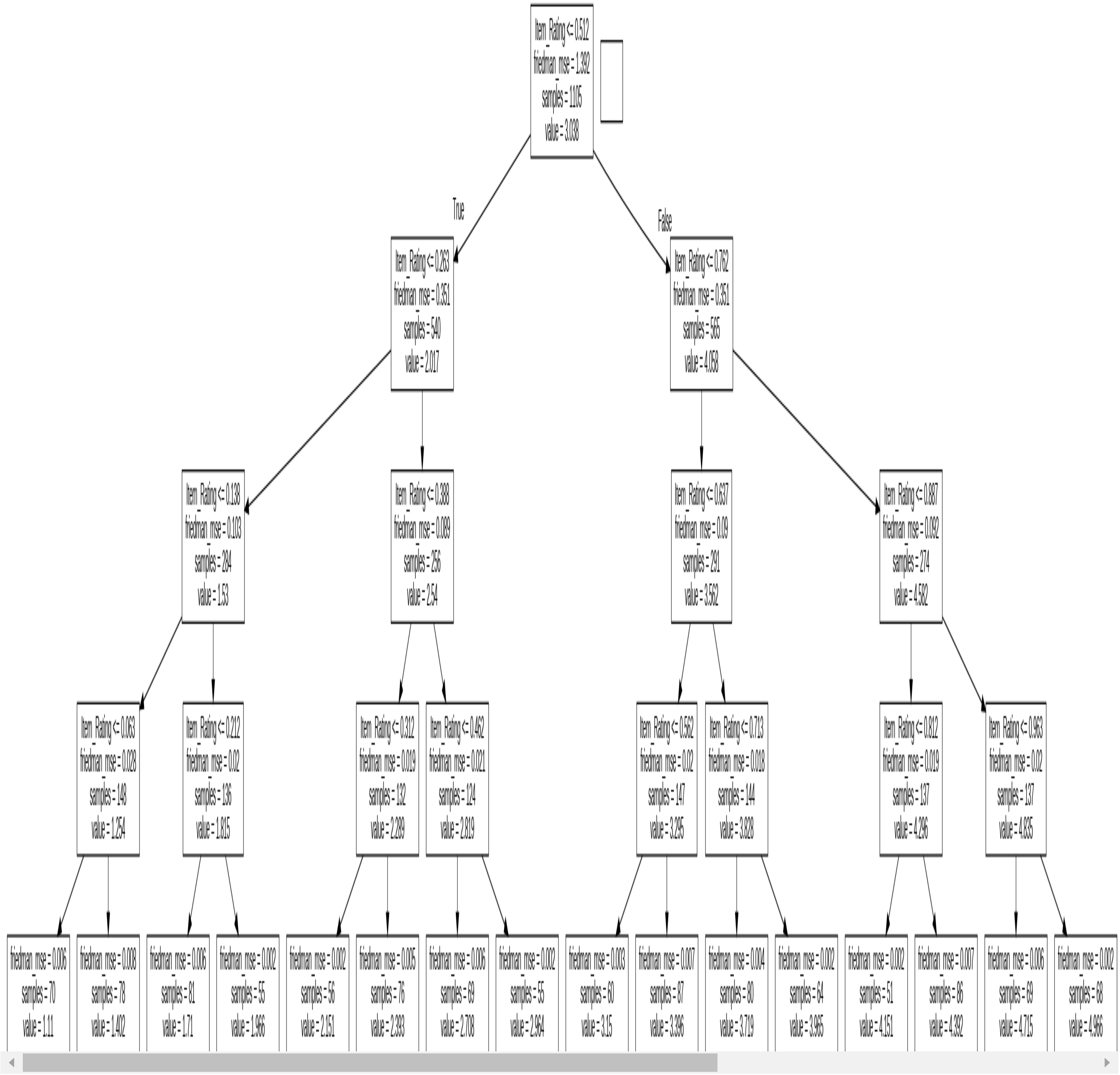
```
# Random Forest (Bagging of multiple Decision Trees)
from sklearn.ensemble import RandomForestRegressor
RegModel = RandomForestRegressor(max_depth=4, n_estimators=400,criterion='friedman_mse')
# Good range for max_depth: 2-10 and n_estimators: 100-1000

# Printing all the parameters of Random Forest
print(RegModel)

# Creating the model on Training Data
RF=RegModel.fit(X_train,y_train)
prediction=RF.predict(X_test)

from sklearn import metrics
# Measuring Goodness of fit in Training data
print('R2 Value:',metrics.r2_score(y_train, RF.predict(X_train)))

# Plotting the feature importance for Top 10 most important columns
%matplotlib inline
feature_importances = pd.Series(RF.feature_importances_, index=Predictors)
feature_importances.nlargest(10).plot(kind='barh')

###############################################################################
print('\n##### Model Validation and Accuracy Calculations ##########')

# Printing some sample values of prediction
TestingDataResults=pd.DataFrame(data=X_test, columns=Predictors)
TestingDataResults[TargetVariable]=y_test
TestingDataResults[('Predicted'+TargetVariable)]=np.round(prediction)

# Printing sample prediction values
print(TestingDataResults.head())

# Calculating the error for each row
# Use the correct column names from the TargetVariable
TestingDataResults['APE']=100 * ((abs(
  TestingDataResults[TargetVariable]-TestingDataResults['Predicted' + TargetVariable]))/TestingDataResults[TargetVariable])

MAPE=np.mean(TestingDataResults['APE'])
MedianMAPE=np.median(TestingDataResults['APE'])

Accuracy =100 - MAPE
MedianAccuracy=100- MedianMAPE
print('Mean Accuracy on test data:', Accuracy) # Can be negative sometimes due to outlier
print('Median Accuracy on test data:', MedianAccuracy)


# Defining a custom function to calculate accuracy
# Make sure there are no zeros in the Target variable if you are using MAPE
def Accuracy_Score(orig,pred):
    MAPE = np.mean(100 * (np.abs(orig-pred)/orig))
    #print('#'*70,'Accuracy:', 100-MAPE)
    return(100-MAPE)

# Custom Scoring MAPE calculation
from sklearn.metrics import make_scorer
custom_Scoring=make_scorer(Accuracy_Score, greater_is_better=True)

# Importing cross validation function from sklearn
from sklearn.model_selection import cross_val_score

# Running 10-Fold Cross validation on a given algorithm
# Passing full data X and y because the K-fold will split the data and automatically choose train/test
Accuracy_Values=cross_val_score(RegModel, X , y, cv=10, scoring=custom_Scoring)
print('\nAccuracy values for 10-fold Cross Validation:\n',Accuracy_Values)
print('\nFinal Average Accuracy of the model:', round(Accuracy_Values.mean(),2))
```

```
RandomForestRegressor(criterion='friedman_mse', max_depth=4, n_estimators=400)
R2 Value: 0.9990687030437887

##### Model Validation and Accuracy Calculations ##########
   Item_Rating  Selling_Price  PredictedItem_Rating
0          1.2       0.005531                   1.0
1          4.9       0.003784                   5.0
2          4.5       0.293956                   4.0
3          1.1       0.001806                   1.0
4          1.7       0.006881                   2.0
Mean Accuracy on test data: 90.66324889798534
Median Accuracy on test data: 91.30434782608695

Accuracy values for 10-fold Cross Validation:
 [97.95079881 98.84208727 98.81883342 98.6043417  98.40321514 98.77406117
 98.00670342 98.61799918 98.29693688 98.7357674 ]

Final Average Accuracy of the model: 98.51
```



Plotting one of decision tree in Random Forest Regression

```
# Plotting a single Decision Tree from Random Forest
# Load libraries
from IPython.display import Image
from sklearn import tree
import pydotplus

# Create DOT data for the 6th Decision Tree in Random Forest
# Remove class_names as it's not applicable for Regression problems
dot_data = tree.export_graphviz(RegModel.estimators_[5] , out_file=None, feature_names=Predictors)

# Draw graph
graph = pydotplus.graph_from_dot_data(dot_data)

# Show graph
Image(graph.create_png(), width=3000,height=1000)
# Double click on the graph to zoom in
```



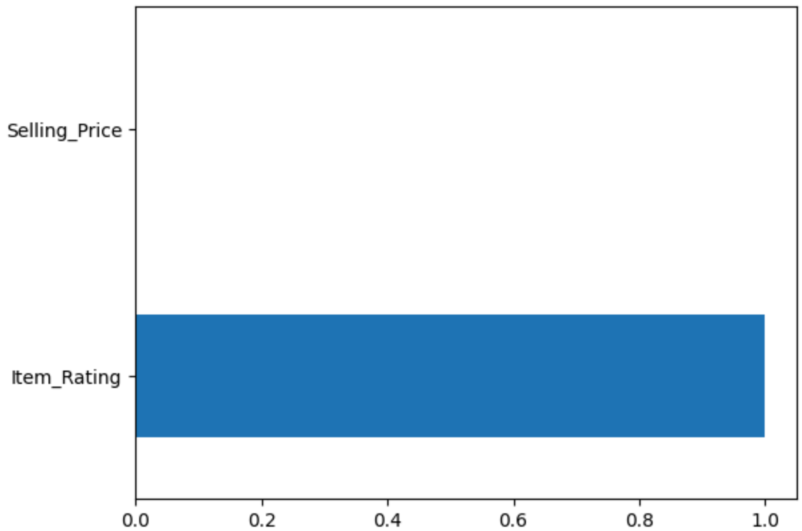
Step 21

AdaBoost Algorithm

```
# Adaboost (Boosting of multiple Decision Trees)
from sklearn.ensemble import AdaBoostRegressor
from sklearn.tree import DecisionTreeRegressor

# Choosing Decision Tree with 6 level as the weak learner
DTR=DecisionTreeRegressor(max_depth=3)
RegModel = AdaBoostRegressor(n_estimators=100, base_estimator=DTR ,learning_rate=0.04)

# Printing all the parameters of Adaboost
print(RegModel)

# Creating the model on Training Data
AB=RegModel.fit(X_train,y_train)
prediction=AB.predict(X_test)

from sklearn import metrics
# Measuring Goodness of fit in Training data
```

```
print('R2 Value:',metrics.r2_score(y_train, AB.predict(X_train)))

# Plotting the feature importance for Top 10 most important columns
%matplotlib inline
feature_importances = pd.Series(AB.feature_importances_, index=Predictors)
feature_importances.nlargest(10).plot(kind='barh')

###############################################################################
print('\n##### Model Validation and Accuracy Calculations ##########')

# Printing some sample values of prediction
TestingDataResults=pd.DataFrame(data=X_test, columns=Predictors)
TestingDataResults[TargetVariable]=y_test
TestingDataResults[('Predicted'+TargetVariable)]=np.round(prediction)

# Printing sample prediction values
print(TestingDataResults.head())

# Calculating the error for each row
TestingDataResults['APE']=100 * ((abs(
  TestingDataResults[TargetVariable]-TestingDataResults['Predicted' + TargetVariable]))/TestingDataResults[TargetVariable])

MAPE=np.mean(TestingDataResults['APE'])
MedianMAPE=np.median(TestingDataResults['APE'])

Accuracy =100 - MAPE
MedianAccuracy=100- MedianMAPE
print('Mean Accuracy on test data:', Accuracy) # Can be negative sometimes due to outlier
print('Median Accuracy on test data:', MedianAccuracy)


# Defining a custom function to calculate accuracy
# Make sure there are no zeros in the Target variable if you are using MAPE
def Accuracy_Score(orig,pred):
    MAPE = np.mean(100 * (np.abs(orig-pred)/orig))
    #print('#'*70,'Accuracy:', 100-MAPE)
    return(100-MAPE)

# Custom Scoring MAPE calculation
from sklearn.metrics import make_scorer
custom_Scoring=make_scorer(Accuracy_Score, greater_is_better=True)

# Importing cross validation function from sklearn
from sklearn.model_selection import cross_val_score

# Running 10-Fold Cross validation on a given algorithm
# Passing full data X and y because the K-fold will split the data and automatically choose train/test
Accuracy_Values=cross_val_score(RegModel, X , y, cv=10, scoring=custom_Scoring)
print('\nAccuracy values for 10-fold Cross Validation:\n',Accuracy_Values)
print('\nFinal Average Accuracy of the model:', round(Accuracy_Values.mean(),2))
```

```
AdaBoostRegressor(base_estimator=DecisionTreeRegressor(max_depth=3),
                  learning_rate=0.04, n_estimators=100)
R2 Value: 0.9857601279738215

##### Model Validation and Accuracy Calculations ##########
   Item_Rating  Selling_Price  PredictedItem_Rating
0         1.2       0.005531                   1.0
1         4.9       0.003784                   5.0
2         4.5       0.293956                   4.0
3         1.1       0.001806                   1.0
4         1.7       0.006881                   2.0
Mean Accuracy on test data: 90.66324889798534
Median Accuracy on test data: 91.30434782608695

Accuracy values for 10-fold Cross Validation:
 [94.38971833 95.49796108 94.63820477 94.89597801 95.19920678 95.31778595
 94.46664257 94.80794614 94.98814044 95.07609336]

Final Average Accuracy of the model: 94.93
```

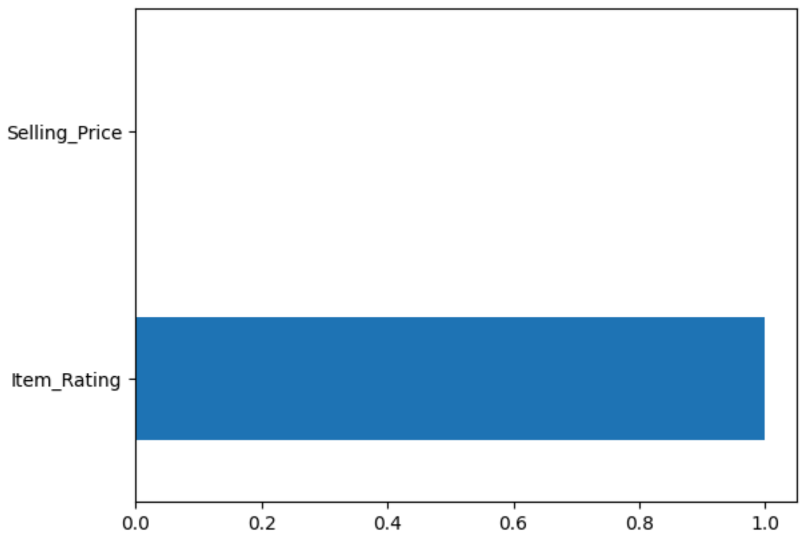


XG Boost Regressor

```
# Xtreme Gradient Boosting (XGBoost)
from xgboost import XGBRegressor
RegModel=XGBRegressor(max_depth=2,
                      learning_rate=0.1,
                      n_estimators=1000,
                      objective='reg:linear',
                      booster='gbtree')

# Printing all the parameters of XGBoost
print(RegModel)

# Creating the model on Training Data
XGB=RegModel.fit(X_train,y_train)
prediction=XGB.predict(X_test)

from sklearn import metrics
# Measuring Goodness of fit in Training data
print('R2 Value:',metrics.r2_score(y_train, XGB.predict(X_train)))

# Plotting the feature importance for Top 10 most important columns
%matplotlib inline
feature_importances = pd.Series(XGB.feature_importances_, index=Predictors)
feature_importances.nlargest(10).plot(kind='barh')
###############################################################################
print('\n##### Model Validation and Accuracy Calculations ##########')

# Printing some sample values of prediction
TestingDataResults=pd.DataFrame(data=X_test, columns=Predictors)
TestingDataResults[TargetVariable]=y_test
TestingDataResults[('Predicted'+TargetVariable)]=np.round(prediction)

# Printing sample prediction values
print(TestingDataResults.head())
```

```
print(TestingDataResults.head())

# Calculating the error for each row
TestingDataResults['APE']=100 * ((abs(
  TestingDataResults[TargetVariable]-TestingDataResults['Predicted' + TargetVariable]))/TestingDataResults[TargetVariable])


MAPE=np.mean(TestingDataResults['APE'])
MedianMAPE=np.median(TestingDataResults['APE'])

Accuracy =100 - MAPE
MedianAccuracy=100- MedianMAPE
print('Mean Accuracy on test data:', Accuracy) # Can be negative sometimes due to outlier
print('Median Accuracy on test data:', MedianAccuracy)


# Defining a custom function to calculate accuracy
# Make sure there are no zeros in the Target variable if you are using MAPE
def Accuracy_Score(orig,pred):
    MAPE = np.mean(100 * (np.abs(orig-pred)/orig))
    #print('#'*70,'Accuracy:', 100-MAPE)
    return(100-MAPE)

# Custom Scoring MAPE calculation
from sklearn.metrics import make_scorer
custom_Scoring=make_scorer(Accuracy_Score, greater_is_better=True)

# Importing cross validation function from sklearn
from sklearn.model_selection import cross_val_score

# Running 10-Fold Cross validation on a given algorithm
# Passing full data X and y because the K-fold will split the data and automatically choose train/test
Accuracy_Values=cross_val_score(RegModel, X , y, cv=10, scoring=custom_Scoring)
print('\nAccuracy values for 10-fold Cross Validation:\n',Accuracy_Values)
print('\nFinal Average Accuracy of the model:', round(Accuracy_Values.mean(),2))
```

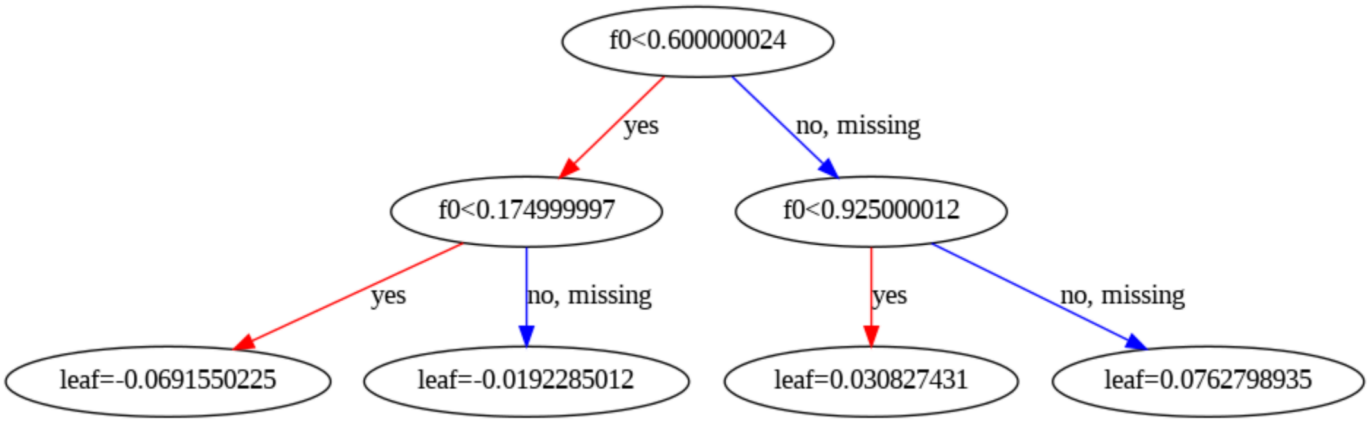
```
XGBRegressor(base_score=None, booster='gbtree', callbacks=None,
             colsample_bylevel=None, colsample_bynode=None,
             colsample_bytree=None, device=None, early_stopping_rounds=None,
             enable_categorical=False, eval_metric=None, feature_types=None,
             gamma=None, grow_policy=None, importance_type=None,
             interaction_constraints=None, learning_rate=0.1, max_bin=None,
             max_cat_threshold=None, max_cat_to_onehot=None,
             max_delta_step=None, max_depth=2, max_leaves=None,
             min_child_weight=None, missing=nan, monotone_constraints=None,
             multi_strategy=None, n_estimators=1000, n_jobs=None,
             num_parallel_tree=None, objective='reg:linear', ...)
R2 Value: 0.9999999379310271

##### Model Validation and Accuracy Calculations ##########
   Item_Rating  Selling_Price  PredictedItem_Rating
0          1.2       0.005531                   1.0
1          4.9       0.003784                   5.0
2          4.5       0.293956                   4.0
3          1.1       0.001806                   1.0
4          1.7       0.006881                   2.0
Mean Accuracy on test data: 90.66324889798534
Median Accuracy on test data: 91.30434782608695

Accuracy values for 10-fold Cross Validation:
 [99.99353035 99.9940059  99.99341373 99.99292925 99.99273238 99.99327859
 99.99221512 99.99230552 99.99445624 99.99324786]

Final Average Accuracy of the model: 99.99
```



Plotting single Decision Tree out of XG Boost

```
#Plotting a single Decision tree out of XGBoost
from xgboost import plot_tree
import matplotlib.pyplot as plt
fig, ax = plt.subplots(figsize=(20, 8))
plot_tree(XGB, num_trees=10, ax=ax)
```

```
<Axes: >
```



K-Nearest Neighbor(KNN)

```
#kNN
# K-Nearest Neighbor(KNN)
from sklearn.neighbors import KNeighborsRegressor
RegModel = KNeighborsRegressor(n_neighbors=3)

# Printing all the parameters of KNN
print(RegModel)

# Creating the model on Training Data
KNN=RegModel.fit(X_train,y_train)
prediction=KNN.predict(X_test)

from sklearn import metrics
# Measuring Goodness of fit in Training data
print('R2 Value:',metrics.r2_score(y_train, KNN.predict(X_train)))

# Plotting the feature importance for Top 10 most important columns
# The variable importance chart is not available for KNN

###############################################################################
print('\n##### Model Validation and Accuracy Calculations ##########')

# Printing some sample values of prediction
TestingDataResults=pd.DataFrame(data=X_test, columns=Predictors)
TestingDataResults[TargetVariable]=y_test
TestingDataResults[('Predicted'+TargetVariable)]=np.round(prediction)

# Printing sample prediction values
print(TestingDataResults.head())

# Calculating the error for each row
TestingDataResults['APE']=100 * ((abs(
  TestingDataResults[TargetVariable]-TestingDataResults['Predicted' + TargetVariable]))/TestingDataResults[TargetVariable])

MAPE=np.mean(TestingDataResults['APE'])
MedianMAPE=np.median(TestingDataResults['APE'])

Accuracy =100 - MAPE
MedianAccuracy=100- MedianMAPE
print('Mean Accuracy on test data:', Accuracy) # Can be negative sometimes due to outlier
print('Median Accuracy on test data:', MedianAccuracy)

# Defining a custom function to calculate accuracy
# Make sure there are no zeros in the Target variable if you are using MAPE
def Accuracy_Score(orig,pred):
    MAPE = np.mean(100 * (np.abs(orig-pred)/orig))
    #print('#'*70,'Accuracy:', 100-MAPE)
    return(100-MAPE)
```

```
# Custom Scoring MAPE calculation
from sklearn.metrics import make_scorer
custom_Scoring=make_scorer(Accuracy_Score, greater_is_better=True)

# Importing cross validation function from sklearn
from sklearn.model_selection import cross_val_score

# Running 10-Fold Cross validation on a given algorithm
# Passing full data X and y because the K-fold will split the data and automatically choose train/test
Accuracy_Values=cross_val_score(RegModel, X , y, cv=10, scoring=custom_Scoring)
print('\nAccuracy values for 10-fold Cross Validation:\n',Accuracy_Values)
print('\nFinal Average Accuracy of the model:', round(Accuracy_Values.mean(),2))
```

```
KNeighborsRegressor(n_neighbors=3)
R2 Value: 0.9999089175502753

##### Model Validation and Accuracy Calculations ##########
   Item_Rating  Selling_Price  PredictedItem_Rating
0         1.2       0.005531                   1.0
1         4.9       0.003784                   5.0
2         4.5       0.293956                   5.0
3         1.1       0.001806                   1.0
4         1.7       0.006881                   2.0
Mean Accuracy on test data: 90.66324889798534
Median Accuracy on test data: 91.30434782608695

Accuracy values for 10-fold Cross Validation:
 [99.74647244 99.85940599 99.90990411 99.83739468 99.90529663 99.9312526
 99.93367127 99.92445448 99.89414247 99.84828355]

Final Average Accuracy of the model: 99.88
```

Support Vector Machine (SVM) Regressor

```
# Support Vector Machines(SVM)
from sklearn import svm
RegModel = svm.SVR(C=50, kernel='rbf', gamma=0.01)

# Printing all the parameters
print(RegModel)

# Creating the model on Training Data
SVM=RegModel.fit(X_train,y_train)
prediction=SVM.predict(X_test)

from sklearn import metrics
# Measuring Goodness of fit in Training data
print('R2 Value:',metrics.r2_score(y_train, SVM.predict(X_train)))

# Plotting the feature importance for Top 10 most important columns
# The built in attribute SVM.coef_ works only for linear kernel
%matplotlib inline
#feature_importances = pd.Series(SVM.coef_[0], index=Predictors)
#feature_importances.nlargest(10).plot(kind='barh')

###########################################################################
print('\n##### Model Validation and Accuracy Calculations ##########')

# Printing some sample values of prediction
TestingDataResults=pd.DataFrame(data=X_test, columns=Predictors)
TestingDataResults[TargetVariable]=y_test
TestingDataResults[('Predicted'+TargetVariable)]=np.round(prediction)

# Printing sample prediction values
print(TestingDataResults.head())

# Calculating the error for each row
TestingDataResults['APE']=100 * ((abs(
  TestingDataResults[TargetVariable]-TestingDataResults['Predicted' + TargetVariable]))/TestingDataResults[TargetVariable])

MAPE=np.mean(TestingDataResults['APE'])
MedianMAPE=np.median(TestingDataResults['APE'])

Accuracy =100 - MAPE
MedianAccuracy=100- MedianMAPE
print('Mean Accuracy on test data:', Accuracy) # Can be negative sometimes due to outlier
```

```
print('Median Accuracy on test data:', MedianAccuracy)

# Defining a custom function to calculate accuracy
# Make sure there are no zeros in the Target variable if you are using MAPE
def Accuracy_Score(orig,pred):
    MAPE = np.mean(100 * (np.abs(orig-pred)/orig))
    #print('#'*70,'Accuracy:', 100-MAPE)
    return(100-MAPE)

# Custom Scoring MAPE calculation
from sklearn.metrics import make_scorer
custom_Scoring=make_scorer(Accuracy_Score, greater_is_better=True)

# Importing cross validation function from sklearn
from sklearn.model_selection import cross_val_score

# Running 10-Fold Cross validation on a given algorithm
# Passing full data X and y because the K-fold will split the data and automatically choose train/test
Accuracy_Values=cross_val_score(RegModel, X , y, cv=10, scoring=custom_Scoring)
print('\nAccuracy values for 10-fold Cross Validation:\n',Accuracy_Values)
print('\nFinal Average Accuracy of the model:', round(Accuracy_Values.mean(),2))
```

```
SVR(C=50, gamma=0.01)
R2 Value: 0.9975753755843485

##### Model Validation and Accuracy Calculations ##########
   Item_Rating  Selling_Price  PredictedItem_Rating
0          1.2       0.005531                   1.0
1          4.9       0.003784                   5.0
2          4.5       0.293956                   4.0
3          1.1       0.001806                   1.0
4          1.7       0.006881                   2.0
Mean Accuracy on test data: 90.66324889798534
Median Accuracy on test data: 91.30434782608695

Accuracy values for 10-fold Cross Validation:
 [97.5112176  97.99082655 97.9923486  97.87370383 97.94091817 97.99333807
 97.47018289 97.82214203 97.89073368 97.84767878]

Final Average Accuracy of the model: 97.83
```

STEP 21

Model Deployment

```
# Separate Target Variable and Predictor Variables
TargetVariable='Item_Rating'

# Selecting the final set of predictors for the deployment
# Based on the variable importance charts of multiple algorithms above
Predictors=['Selling_Price']

X=DataForML_Numeric[Predictors].values
y=DataForML_Numeric[TargetVariable].values

### Sandardization of data ###
from sklearn.preprocessing import StandardScaler, MinMaxScaler
# Choose either standardization or Normalization
# On this data Min Max Normalization produced better results

# Choose between standardization and MinMAx normalization
#PredictorScaler=StandardScaler()
PredictorScaler=MinMaxScaler()

# Storing the fit object for later reference
PredictorScalerFit=PredictorScaler.fit(X)

# Generating the standardized values of X
X=PredictorScalerFit.transform(X)

print(X.shape)
print(y.shape)
```

```
(2452, 1)
(2452,)
```

Cross Validation

```
# Importing cross validation function from sklearn
from sklearn.model_selection import cross_val_score

# choose from different tunable hyper parameters
from xgboost import XGBRegressor
RegModel=XGBRegressor(max_depth=2,
                      learning_rate=0.1,
                      n_estimators=1000,
                      objective='reg:linear',
                      booster='gbtree')

# Running 10-Fold Cross validation on a given algorithm
# Passing full data X and y because the K-fold will split the data and automatically choose train/test
Accuracy_Values=cross_val_score(RegModel, X , y, cv=10, scoring=custom_Scoring)
print('\nAccuracy values for 10-fold Cross Validation:\n',Accuracy_Values)
print('\nFinal Average Accuracy of the model:', round(Accuracy_Values.mean(),2))
```

```
Accuracy values for 10-fold Cross Validation:
 [47.85892455 56.39302407 56.21418827 56.0352065  56.18893321 55.43157763
 45.40816829 54.36641636 54.64114095 55.84435462]

Final Average Accuracy of the model: 53.84
```

STEP 22

Retraining Model using 100% data

```
# Training the model on 100% Data available
Final_XGB_Model=RegModel.fit(X,y)
```

STEP 23

Saving Model as Serialized file

```
import pickle
import os

# Saving the Python objects as serialized files can be done using pickle library
# Here let us save the Final model
with open('Final_XGB_Model.pkl', 'wb') as fileWriteStream:
    pickle.dump(Final_XGB_Model, fileWriteStream)
    # Don't forget to close the filestream!
    fileWriteStream.close()

print('pickle file of Predictive Model is saved at Location:',os.getcwd())
```

```
pickle file of Predictive Model is saved at Location: /content
```

STEP 24

Creating Python Function

```
from re import IGNORECASE
# This Function can be called from any from any front end tool/website

def FunctionPredictResult(InputData):
    import pandas as pd
    Num_Inputs=InputData.shape[0]

    # Making sure the input data has same columns as it was used for training the model
    # Also, if standardization/normalization was done, then same must be done for new input

    # Appending the new data with the Training data
    DataForML=pd.read_pickle('DataForML.pkl')
    #InputData=InputData.append(DataForML, ignore_index=True)
    InputData = pd.concat([InputData, DataForML], ignore_index=True)
```