```
#%% md
# **ST1/ST1G Assignment 9 (Capstone Programming Project)**
#%% md
### This Project is based on E-commerce Price Prediction available from Kaggle repository (https://www.kaggle.com/datasets/bhuwanesh340/ecommerce-price-prediction/data ).


*   It contains the details of 2452 products.
*   Project task is to create a machine learning model which can predict the price of products based on its attributes.
* For solving this problem, I will approach the task , with a step by step approach to create a data analysis and prediction model based on (machine learning/AI algorithms, regression algorith for example) available from different Python packages, modules and classes.

#%% md
### Step 1: Reading the data Reading the data with python
#%%
# Supressing the warning messages
import warnings
warnings.filterwarnings('ignore')
#%%
# Reading the dataset
import pandas as pd
df=pd.read_csv('D:\PricePrediction\Train.csv', encoding='latin')
print('Shape before deleting duplicate values:', df.shape)

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

```

```
# Printing sample data
# Start observing the Quantitative/Categorical/Qualitative variables
df.head(10)

#%% md
## Key observations from Step 1 about Data Description


*   This file contains 2452 product details from dataset.
*   There are 8 attributes and they are outlined below.
*   Product - Unique ID for each product
*   Product_Brand
*   Item_Category
*   Subcategory_1
*   Subcategory_2
*   Item_Rating - customer ratings of products
*   Date - The date the item was sold
*   Selling_Price



#%% md
# Step 2 : Problem Statement Definition
* Creating  a prediction model to wpredict the price of a product.
* Target Variable: SellingPrice
* Predictors/Features: Product/Product_Brand/Item_Category/Subcategory_1/Subcategory_2/Item_Rating/Date
#%% md
## Step 3: Choosing the appropriate ML/AI Algorithm for Data Analysis.
* Based on the problem statement to create a supervised ML Regression model, as the target variable is **Continuous**.

#%% md
```

```
58 # Step 4: Looking at the class distribution (Target
   variable distribution to check if the data is
   balanced or skewed.
59 #%%
60 %matplotlib inline
61 # Creating histogram as the Target variable is
   Continuous
62
63 df['Selling_Price'].hist(bins=20)
64
65 #%% md
66 ## Observations from Step 4
67 * The data distribution of the target variable is too
    skewed.
68 * After removing the data with large deviation there
   still have sufficient number of rows for each type of
    values to learn from.
69 #%% md
70 ## Step 5: Basic Exploratory Data Analysis
71
72 #%%
73 # Looking at sample rows in the data
74 df.head()
75 #%%
76 # Looking at sample rows in the data
77 df.tail()
78 #%%
79 # Summarise information of data
80 df.info()
81 #%%
82 # Looking at the descriptive statistics of the data
83 df.describe(include='all')
84 #%%
85 #Convert Date into categorical variable
86 df['Date'] = pd.to_datetime(df['Date'],format='%d/%m
   /%Y')
87
88 # import the holidays list
89 import holidays
90
91 df['Date'] = pd.to_datetime(df['Date'])
```

```
 92
 93 # Create a holidays object for Australia
 94 aus_holidays = holidays.Australia(years=[df['Date'].
    dt.year.min(), df['Date'].dt.year.max()])
 95
 96 # Check each date in the Date column to see if it is
     a holiday
 97 df['Is_Holiday'] = df['Date'].apply(lambda x: 1 if x
     in aus_holidays else 0)
 98 print(df.head())
 99
100 #%%
101 # Finding unique values for each column
102 # Typically if the numer of unique values are < 20
    then the variable is likely to be a category
    otherwise continuous
103 df.nunique()
104 #%% md
105 ## Observations from Step 5 - Basic Exploratory Data
     Analysis
106 * Product           - Just products' unique ID,
    waiting for delete.
107 * Product_Brand     - Categorical. Selected.
108 * Item_Category     - Categorical. Selected.
109 * Subcategory_1     - Categorical. Selected.
110 * Subcategory_2     - Categorical. Selected.
111 * Item_Rating       - Continuous. Selected.
112 * Date              - Transfer to Is_Holiday
113 * Selling_Price     - Continuous. Selected. This is
    the Target variable.
114 * Is_Holiday        - Categorial. Selected.
115 #%% md
116 ## Step 7: Removing Unwanted columns
117 * If each product has a unique identifier, this
    column will not provide any generalizable insight
    into the selling price across different products, as
     it does not encode any repeatable pattern that can
    be learned by the model.
118 * Remove the Product column as it is likely to lead
    to overfitting, where the model learns noise
    specific to individual data points rather than
```

```
118 general trends.
119 * Converting dates to categorical variables for
    research and machine learning. Considering that
    sales discounts are offered on holidays, there is an
     impact on the sales price.
120 * Replace Date column with Is_Holidays.
121 #%%
122 #Convert Date into categorical variable
123 df['Date'] = pd.to_datetime(df['Date'],format='%d/%m
    /%Y')
124
125 # import the holidays list
126 import holidays
127
128 df['Date'] = pd.to_datetime(df['Date'])
129
130 # Create a holidays object for Australia
131 aus_holidays = holidays.Australia(years=[df['Date'].
    dt.year.min(), df['Date'].dt.year.max()])
132
133 # Check each date in the Date column to see if it is
     a holiday
134 df['Is_Holiday'] = df['Date'].apply(lambda x: 1 if x
     in aus_holidays else 0)
135 print(df.head())
136
137 # Remove 'Product' and 'Date'.
138 df = df.drop('Product', axis=1)
139 df = df.drop('Date', axis=1)
140
141 #%% md
142 ## Step 8: Visual Exploratory Data Analysis
143 * Based on the Basic Exploration Data Analysis in
    the previous step,  we could spotted two categorical
     predictors in the data
144 * Categorical Predictors:
145
146 * Product_Brand
147 * Item_Category
148 * Subcategory_1
149 * Subcategory_2
```

```
* Is_Holiday

* Use bar charts to see how the data is distributed for these categorical columns.

#%%
# Plotting multiple bar charts at once for categorical variables
# Since there is no default function which can plot bar charts for multiple columns at once
# we are defining our own function for the same

import matplotlib.pyplot as plt
import seaborn as sns

#  Item_Category
plt.figure(figsize=(12, 6))
sns.countplot(data=df, x='Item_Category', order=df['Item_Category'].value_counts().index[:50])
plt.xticks(rotation=90)
plt.title('Item Category Distribution')
plt.show()

# Subcategory_1
plt.figure(figsize=(12, 6))
sns.countplot(data=df, x='Subcategory_1', order=df['Subcategory_1'].value_counts().index[:20])
plt.xticks(rotation=90)
plt.title('Top 20 Subcategory 1 Distribution')
plt.show()

# Subcategory_2
plt.figure(figsize=(12, 6))
sns.countplot(data=df, x='Subcategory_2', order=df['Subcategory_2'].value_counts().index[:20])
plt.xticks(rotation=90)
plt.title('Top 20 Subcategory 2 Distribution')
plt.show()

#Is_Holiday
plt.figure(figsize=(12, 6))
```

```
185 sns.countplot(data=df, x='Is_Holiday', order=df['
    Is_Holiday'].value_counts().index[:20])
186 plt.xticks(rotation=90)
187 plt.title('Is_Holiday Distribution')
188 plt.show()
189
190 plt.figure(figsize=(12, 6))
191 sns.countplot(data=df, x='Product_Brand', order=df['
    Product_Brand'].value_counts().index[:50])
192 plt.xticks(rotation=90)
193 plt.title('ProductBrand Distribution')
194 plt.show()
195 #%% md
196 ## Observations from Step 8 - Visual Exploratory
    Data Analysis
197 * Although the distribution of Product_Brand and
    Item_Category bars charts is skewed, but they were
    retained given the need for the predictive model to
    include a variety of different categories of goods.
198
199 * Selected Categorical Variables (**Product_Brand,
    Item_Category,Subcategory_1,Subcategory_2,Is_Holiday
    **): Both the categorical variables are selected for
     further analysis.
200 #%% md
201 ## Step 9: Now Visualize distribution of all the
    Continuous Predictor variables in the data using
    histograms
202 * Based on the Basic Exploratory Data Analysis,
    there are eleven continuous predictor variables '
    CRIM', 'ZN', 'INDUS', 'NOX', 'RM', 'AGE', 'DIS', '
    TAX', 'PTRATIO', 'B', and 'LSTAT'.
203 #%%
204 # Plotting histograms of multiple columns together
205 df.hist(['Item_Rating', 'Selling_Price'], figsize=(
    18,10))
206 #%% md
207 ## Observations from Step 9
208 * The distribution of Selling_Price is too skewed,
    with one piece of data dominating and under 6000
    needing to be dealt with.
```

```
209
210 * Selected Continuous Variables:
211
212 * Selling_Price : Selected. Outliers seen beyond
    6000, need to treat them.
213 * Item_Rating : Selected. The distribution is good.
214 #%% md
215 ## Step 10: Outlier Analysis
216 * Data with too high a selling price but very low
    frequency can make the model very inaccurate, so
    these outliers are removed
217
218 #%%
219 #Removing outliers for 'Selling_Price'
220
221 df = df[df['Selling_Price'] <= 6000]
222
223 #%% md
224 Observation: Above result hence, deleting any value
    above 6000 with it.
225 #%% md
226 ## Step 11:Visualising Data Distribution after
    outlier removal
227
228 #%%
229 df.hist(['Selling_Price'], figsize=(18,5))
230 #%% md
231 ## Observation from Step 11
232 * The distribution has improved after the outlier
    treatment.
233 * There is still a tail but it is thick, that means
    there are many values in that range, hence, it is
    acceptable.
234 #%% md
235 ## Step 12: Missing Values Analysis
236 #%%
237 # Finding how many missing values are there for each
     column
238 df.isnull().sum()
239 #%% md
240 ##Observations from Step 12: Missing Value Analysis
```

```
241 * No missing values in this data!
242 * So no removal of any data samples(rows) is needed.
243
244 #%% md
245 ## Step 13: Feature Selection (Attribute Selection)
246
247 * **For this dataset, the Target variable is
    Continuous, hence following two scenarios will need
    attention**
248
249 * Continuous Target Variable Vs Continuous Predictor
250 * Continuous Target Variable Vs Categorical
    Predictor
251
252 #%% md
253 ## Relationship exploration: Continuous Vs
    Continuous -- Scatter Charts
254 * Using scatter plot and measure the strength of
    relation using a metric called pearson's correlation
     value.
255 #%%
256 ContinuousCols=['Item_Rating']
257
258 # Plotting scatter chart for each predictor vs the
    target variable
259 for predictor in ContinuousCols:
260     df.plot.scatter(x=predictor, y='Selling_Price',
    figsize=(10,5), title=predictor+" VS "+ '
    Selling_Price')
261 #%% md
262 ## Scatter charts interpretation
263 * **No Trend**: I cannot see any clear increasing or
     decreasing trend. This means there is no
    correlation between the variables. Hence that
    predictor/feature may not be the best one for ML
    model building.
264
265 * I'll confirm this by looking at the correlation
    value in the next step.
266 #%% md
267 ## Step 14: Statistical Feature Selection (
```

```
Continuous Vs Continuous) using Correlation value
#%%
# Calculating correlation matrix
ContinuousCols=['Selling_Price', 'Item_Rating']

# Creating the correlation matrix
CorrelationData=df[ContinuousCols].corr()
CorrelationData
#%%
# Filtering only those columns where absolute
    correlation > 0.5 with Target Variable
# reduce the 0.5 threshold if no variable is
    selected
CorrelationData['Selling_Price'][abs(CorrelationData
    ['Selling_Price']) > 0.5 ]
#%% md
## Observations from Step 14
* From the results we can see that there is no
    correlation between Item_Rating and the target
    variable. No Continuous columns selected.
#%% md
# Step 15:  Relationship exploration: Categorical Vs
     Continuous -- Box Plots
* Measure the strength of relation using Anova test.
#%%
# Box plots for continuous Target Variable "
    Selling_Price" and Categorical predictors
CategoricalColsList=['Product_Brand','Item_Category'
    , 'Subcategory_1', 'Subcategory_2', 'Is_Holiday']

import matplotlib.pyplot as plt
fig, PlotCanvas=plt.subplots(nrows=1, ncols=len(
    CategoricalColsList), figsize=(18,5))

# Creating box plots for each continuous predictor
    against the Target Variable "MEDV"
for PredictorCol , i in zip(CategoricalColsList,
    range(len(CategoricalColsList))):
    df.boxplot(column='Selling_Price', by=
    PredictorCol, figsize=(5,5), vert=True, ax=
    PlotCanvas[i])
```

```
#%% md
##Observations from Step 15: Box-Plots interpretation
* The distribution is different for each category( the boxes are not in same line). It hints that these variables might be correlated with Selling_Price.

* For this data, all the categorical predictors looks correlated with the Target variable.

We confirm this by looking at the results of ANOVA test below
#%% md
## Step 16: Statistical Feature Selection ( Categorical Vs Continuous) using ANOVA test
#%%
# Defining a function to find the statistical relationship with all the categorical variables
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:
        CategoryGroupLists=inpData.groupby(predictor)[TargetVariable].apply(list)
        AnovaResults = f_oneway(*CategoryGroupLists)

        # If the ANOVA P-Value is <0.05, that means we reject H0
        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])
```

```
323
324     return(SelectedPredictors)
325 #%%
326 #Calling the function to check which categorical
    variables are correlated with target
327 CategoricalPredictorList=['Product_Brand','
    Subcategory_1', 'Subcategory_2', 'Item_Category','
    Is_Holiday']
328 FunctionAnova(inpData=df,
329               TargetVariable='Selling_Price',
330               CategoricalPredictorList=
    CategoricalPredictorList)
331 #%% md
332 ##Observations from Step 16
333 * Final selected Categorical columns:
334
335 'Product_Brand','Subcategory_1', 'Subcategory_2', '
    Item_Category','Is_Holiday'
336
337 #%% md
338 ## Selecting final Predictors/Features for building
    Machine Learning/AI model.
339 * Select the final features/predictors/columns for
    machine learning model building as:
340 * **'Product_Brand','Subcategory_1', 'Subcategory_2
    ', 'Item_Category','Is_Holiday**
341
342 #%%
343 SelectedColumns=['Product_Brand','Subcategory_1', '
    Subcategory_2', 'Item_Category','Is_Holiday']
344
345 # Selecting final columns
346 DataForML=df[SelectedColumns]
347 DataForML.head()
348
349 #%%
350 # Saving this final data subset for reference during
     deployment
351 DataForML.to_pickle('DataForML.pkl')
352 #%% md
353 ## Step 17: Data Pre-processing for Machine Learning
```

```
 Model Building or Model Development
Considering the large number of categorical variables in the predicted values, it is necessary to encode the data using one-hot coding for easy use in machine learning.

#%% md
## Converting the nominal variable to numeric using get_dummies()
#%%
# Treating all the nominal variables at once using dummy variables
DataForML_Numeric=pd.get_dummies(DataForML)

# Adding Target Variable to the data
DataForML_Numeric['Selling_Price']=df['Selling_Price']

# Printing sample rows
DataForML_Numeric.head()
#%% md
## Step 18: Machine Learning Model Development:
#%%
# Printing all the column names for reference
DataForML_Numeric.columns
#%%
#Separate Target Variable and Predictor Variables
import numpy as np
import pandas as pd
from sklearn.preprocessing import MinMaxScaler
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression

TargetVariable = 'Selling_Price'
Predictors = [column for column in DataForML_Numeric.columns if column != TargetVariable]


# Extraction of features and target variables
X=DataForML_Numeric[Predictors].values
y=DataForML_Numeric[TargetVariable].values
```

```
387
388 # Split the data into training(70%) and testing set(
    30%).
389 from sklearn.model_selection import train_test_split
390 X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.3, random_state=428)
391
392 #%% md
393 ## Step 19: Standardization/Normalization of data
394 * I'm using KNN, so this step is necessary.
395 #%%
396 ### Standardization of data ###
397 from sklearn.preprocessing import StandardScaler,
    MinMaxScaler
398
399 # Using MinMAx normalization is better
400 PredictorScaler=MinMaxScaler()
401
402 # Storing the fit object for later reference
403 PredictorScalerFit=PredictorScaler.fit(X)
404
405 # Generating the standardized values of X
406 X=PredictorScalerFit.transform(X)
407
408 # Split the data into training and testing set
409 from sklearn.model_selection import train_test_split
410 X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.3, random_state=42)
411 #%%
412 # Sanity check for the sampled data
413 print(X_train.shape)
414 print(y_train.shape)
415 print(X_test.shape)
416 print(y_test.shape)
417 #%% md
418 ## Step 20: Multiple Linear Regression Algorithm For
     ML/AI model building
419 #%% md
420 # Decision Tree Regressor
421 #%%
422 # Decision Trees (Multiple if-else statements!)
```

```
423 from sklearn.tree import DecisionTreeRegressor
424 RegModel = DecisionTreeRegressor(max_depth=5,
    criterion='friedman_mse')
425 # Good Range of Max_depth = 2 to 20
426
427 # Printing all the parameters of Decision Tree
428 print(RegModel)
429
430 # Creating the model on Training Data
431 DT=RegModel.fit(X_train,y_train)
432 prediction=DT.predict(X_test)
433
434 from sklearn import metrics
435 # Measuring Goodness of fit in Training data
436 print('R2 Value:',metrics.r2_score(y_train, DT.
    predict(X_train)))
437
438 # Plotting the feature importance for Top 10 most
    important columns
439 %matplotlib inline
440 feature_importances = pd.Series(DT.
    feature_importances_, index=Predictors)
441 feature_importances.nlargest(10).plot(kind='barh')
442
443 ##################################################
    ########################
444 print('\n##### Model Validation and Accuracy
    Calculations ##########')
445
446 # Printing some sample values of prediction
447 TestingDataResults=pd.DataFrame(data=X_test, columns
    =Predictors)
448 TestingDataResults[TargetVariable]=y_test
449 TestingDataResults[('Predicted'+TargetVariable)]=np.
    round(prediction)
450
451 # Printing sample prediction values
452 print(TestingDataResults.head())
453
454 # Calculating the error for each row
455 TestingDataResults['APE']=100 * ((abs(
```

```
456   TestingDataResults['Selling_Price']-
    TestingDataResults['PredictedSelling_Price']))/
    TestingDataResults['Selling_Price'])
457
458 MAPE=np.mean(TestingDataResults['APE'])
459 MedianMAPE=np.median(TestingDataResults['APE'])
460
461 Accuracy =100 - MAPE
462 MedianAccuracy=100- MedianMAPE
463 print('Mean Accuracy on test data:', Accuracy) # Can
     be negative sometimes due to outlier
464 print('Median Accuracy on test data:',
    MedianAccuracy)
465
466 # Defining a custom function to calculate accuracy
467 # Make sure there are no zeros in the Target
    variable
468 def Accuracy_Score(orig,pred):
469     MAPE = np.mean(100 * (np.abs(orig-pred)/orig))
470     #print('#'*70,'Accuracy:', 100-MAPE)
471     return(100-MAPE)
472
473 # Custom Scoring MAPE calculation
474 from sklearn.metrics import make_scorer
475 custom_Scoring=make_scorer(Accuracy_Score,
    greater_is_better=True)
476
477 # Importing cross validation function from sklearn
478 from sklearn.model_selection import cross_val_score
479
480 # Running 10-Fold Cross validation on a given
    algorithm
481 # Passing full data X and y because the K-fold will
    split the data and automatically choose train/test
482 Accuracy_Values=cross_val_score(RegModel, X , y, cv=
    10, scoring=custom_Scoring)
483 print('\nAccuracy values for 10-fold Cross
    Validation:\n',Accuracy_Values)
484 print('\nFinal Average Accuracy of the model:',
    round(Accuracy_Values.mean(),2))
485 #%% md
```

```
# Plotting/Visualising the Decision Tree
#%%

# Load libraries
from IPython.display import Image
from sklearn import tree
import pydotplus

# Create DOT data
dot_data = tree.export_graphviz(RegModel, out_file=
    None,
                                    feature_names=
    Predictors, class_names=TargetVariable)

# printing the rules
print(dot_data)

# Draw graph
graph = pydotplus.graph_from_dot_data(dot_data)

# Show graph
Image(graph.create_png(), width=2000,height=2000)
# Double click on the graph to zoom in
#%% md
# Random Forest Regressor
#%%
# 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
TestingDataResults['APE']=100 * ((abs(
  TestingDataResults['Selling_Price']-
    TestingDataResults['PredictedSelling_Price']))/
    TestingDataResults['Selling_Price'])

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))
#%% md

## Step 21: AdaBoost Algorithm For ML/AI model building
#%%
# 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 ,
```

```
584 learning_rate=0.04)
585
586 # Printing all the parameters of Adaboost
587 print(RegModel)
588
589 # Creating the model on Training Data
590 AB=RegModel.fit(X_train,y_train)
591 prediction=AB.predict(X_test)
592
593 from sklearn import metrics
594 # Measuring Goodness of fit in Training data
595 print('R2 Value:',metrics.r2_score(y_train, AB.
    predict(X_train)))
596
597 # Plotting the feature importance for Top 10 most
    important columns
598 %matplotlib inline
599 feature_importances = pd.Series(AB.
    feature_importances_, index=Predictors)
600 feature_importances.nlargest(10).plot(kind='barh')
601
602 ##################################################
    ########################
603 print('\n##### Model Validation and Accuracy
    Calculations ##########')
604
605 # Printing some sample values of prediction
606 TestingDataResults=pd.DataFrame(data=X_test, columns
    =Predictors)
607 TestingDataResults[TargetVariable]=y_test
608 TestingDataResults[('Predicted'+TargetVariable)]=np.
    round(prediction)
609
610 # Printing sample prediction values
611 print(TestingDataResults.head())
612
613 # Calculating the error for each row
614 TestingDataResults['APE']=100 * ((abs(
615   TestingDataResults['Selling_Price']-
    TestingDataResults['PredictedSelling_Price']))/
    TestingDataResults['Selling_Price'])
```

```
616
617 MAPE=np.mean(TestingDataResults['APE'])
618 MedianMAPE=np.median(TestingDataResults['APE'])
619
620 Accuracy =100 - MAPE
621 MedianAccuracy=100- MedianMAPE
622 print('Mean Accuracy on test data:', Accuracy) # Can
     be negative sometimes due to outlier
623 print('Median Accuracy on test data:',
    MedianAccuracy)
624
625
626 # Defining a custom function to calculate accuracy
627 # Make sure there are no zeros in the Target
    variable if you are using MAPE
628 def Accuracy_Score(orig,pred):
629     MAPE = np.mean(100 * (np.abs(orig-pred)/orig))
630     #print('#'*70,'Accuracy:', 100-MAPE)
631     return(100-MAPE)
632
633 # Custom Scoring MAPE calculation
634 from sklearn.metrics import make_scorer
635 custom_Scoring=make_scorer(Accuracy_Score,
    greater_is_better=True)
636
637 # Importing cross validation function from sklearn
638 from sklearn.model_selection import cross_val_score
639
640 # Running 10-Fold Cross validation on a given
    algorithm
641 # Passing full data X and y because the K-fold will
    split the data and automatically choose train/test
642 Accuracy_Values=cross_val_score(RegModel, X , y, cv=
    10, scoring=custom_Scoring)
643 print('\nAccuracy values for 10-fold Cross
    Validation:\n',Accuracy_Values)
644 print('\nFinal Average Accuracy of the model:',
    round(Accuracy_Values.mean(),2))
645 #%% md
646 # XGBoost Regressor
647 #%%
```

```
# 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())

```

```
# Calculating the error for each row
TestingDataResults['APE']=100 * ((abs(
  TestingDataResults['Selling_Price']-
TestingDataResults['PredictedSelling_Price']))/
TestingDataResults['Selling_Price'])


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
```

```
713 Validation:\n',Accuracy_Values)
714 print('\nFinal Average Accuracy of the model:',
    round(Accuracy_Values.mean(),2))
715 #%% md
716 #Plotting a single Decision tree out of XGBoost
717 #%%
718 #Plotting a single Decision tree out of XGBoost
719 from xgboost import plot_tree
720 import matplotlib.pyplot as plt
721 fig, ax = plt.subplots(figsize=(20, 8))
722 plot_tree(XGB, num_trees=10, ax=ax)
723 #%% md
724 # K-Nearest Neighbor(KNN)
725 #%%
726 #kNN
727 # K-Nearest Neighbor(KNN)
728 from sklearn.neighbors import KNeighborsRegressor
729 RegModel = KNeighborsRegressor(n_neighbors=3)
730
731 # Printing all the parameters of KNN
732 print(RegModel)
733
734 # Creating the model on Training Data
735 KNN=RegModel.fit(X_train,y_train)
736 prediction=KNN.predict(X_test)
737
738 from sklearn import metrics
739 # Measuring Goodness of fit in Training data
740 print('R2 Value:',metrics.r2_score(y_train, KNN.
    predict(X_train)))
741
742 # Plotting the feature importance for Top 10 most
    important columns
743 # The variable importance chart is not available for
     KNN
744
745 ##################################################
    ########################
746 print('\n##### Model Validation and Accuracy
    Calculations ##########')
747
```

```
# 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['Selling_Price']-TestingDataResults['PredictedSelling_Price']))/TestingDataResults['Selling_Price'])

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
```

```
781
782 # Running 10-Fold Cross validation on a given
    algorithm
783 # Passing full data X and y because the K-fold will
    split the data and automatically choose train/test
784 Accuracy_Values=cross_val_score(RegModel, X , y, cv=
    10, scoring=custom_Scoring)
785 print('\nAccuracy values for 10-fold Cross
    Validation:\n',Accuracy_Values)
786 print('\nFinal Average Accuracy of the model:',
    round(Accuracy_Values.mean(),2))
787 #%% md
788 # Step 21: Model Deployment
789 * I am choosing XGboost as the final model it has
    the highest accuracy(35.78).
790
791 * For this dataset, the most important predictor
    variables are 'Product_Brand','Subcategory_1', '
    Subcategory_2', 'Item_Category'.
792 #%%
793 # Separate Target Variable and Predictor Variables
794 TargetVariable='Selling_Price'
795
796 # Selecting the final set of predictors for the
    deployment
797 # Based on the variable importance charts of
    multiple algorithms above
798 Predictors=[column for column in DataForML_Numeric.
    columns if column != TargetVariable]
799
800 X=DataForML_Numeric[Predictors].values
801 y=DataForML_Numeric[TargetVariable].values
802
803 ### Sandardization of data ###
804 from sklearn.preprocessing import StandardScaler,
    MinMaxScaler
805 # Choose either standardization or Normalization
806 # On this data Min Max Normalization produced better
     results
807
808 # Choose between standardization and MinMAx
```

```
808 normalization
809 #PredictorScaler=StandardScaler()
810 PredictorScaler=MinMaxScaler()
811
812 # Storing the fit object for later reference
813 PredictorScalerFit=PredictorScaler.fit(X)
814
815 # Generating the standardized values of X
816 X=PredictorScalerFit.transform(X)
817
818 print(X.shape)
819 print(y.shape)
820 #%% md
821 # Cross validating the final model accuracy with
    less predictors
822 #%%
823 # Importing cross validation function from sklearn
824 from sklearn.model_selection import cross_val_score
825
826 # choose from different tunable hyper parameters
827 from xgboost import XGBRegressor
828 RegModel=XGBRegressor(max_depth=2,
829                       learning_rate=0.1,
830                       n_estimators=1000,
831                       objective='reg:linear',
832                       booster='gbtree')
833
834 # Running 10-Fold Cross validation on a given
    algorithm
835 # Passing full data X and y because the K-fold will
    split the data and automatically choose train/test
836 Accuracy_Values=cross_val_score(RegModel, X , y, cv=
    10, scoring=custom_Scoring)
837 print('\nAccuracy values for 10-fold Cross
    Validation:\n',Accuracy_Values)
838 print('\nFinal Average Accuracy of the model:',
    round(Accuracy_Values.mean(),2))
839 #%% md
840 # Step 22: Retraining the final model using 100%
    data
841 #%%
```

```
# Training the model on 100% Data available
Final_XGB_Model=RegModel.fit(X,y)
#%% md
# Step 23: Save the model as a serialized file which
 can be stored anywhere
#%%
import pickle
import os

# Saving the Python objects as serialized files can
be done using pickle library
# Save the Final model
with open('Final_XGB_Model.pkl', 'wb') as
fileWriteStream:
    pickle.dump(Final_XGB_Model, fileWriteStream)
    # Close the filestream!
    fileWriteStream.close()

print('pickle file of Predictive Model is saved at
Location:',os.getcwd())
#%% md
# Step 24: Create a python function
#%%
from re import IGNORECASE

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

    # 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)

    # Generating dummy variables for rest of the
nominal variables
    InputData=pd.get_dummies(InputData)

    # Maintaining the same order of columns as it
```

```
was during the model training
    Predictors=[column for column in DataForML_Numeric.columns if column != TargetVariable]

    # Generating the input values to the model
    X=InputData[Predictors].values[0:Num_Inputs]

    # Generating the standardized values of X since it was done while model training also
    X=PredictorScalerFit.transform(X)

    # Loading the Function from pickle file
    import pickle
    with open('Final_XGB_Model.pkl', 'rb') as fileReadStream:
        PredictionModel=pickle.load(fileReadStream)
        # Don't forget to close the filestream!
        fileReadStream.close()

    # Genrating Predictions
    Prediction=PredictionModel.predict(X)
    PredictionResult=pd.DataFrame(Prediction, columns=['Prediction'])
    return(PredictionResult)
#%% md
# Step 25: Calling the function for some new data
#%%
# Load the test dataframe to test the function
test_data = pd.read_csv('D:\PricePrediction\Test.csv')
NewSampleData=pd.DataFrame(data=test_data,columns=['Product_Brand','Item_Category','Subcategory_1','Subcategory_2'])
print(NewSampleData)
# Passing the loaded DataFrame
FunctionPredictResult(InputData=NewSampleData)

#%% md
# Conclusion
* The Function FunctionPredictResult() can be used
```

```
907 to produce the predictions for one or more new cases
     at a time.
908 * Hence, it can be scheduled using a batch job or
    cron job to run every night and generate predictions
     for all the products  in the E-commerce platform.
909 #%% md
910 # Desktop App deployment: Tkinter package
911
912 #%%
913 import tkinter as tk
914 from tkinter import messagebox
915 from tkinter import ttk
916 import pandas as pd
917 from sklearn.model_selection import train_test_split
918 from xgboost import XGBRegressor
919
920 class SellingPricePredictionApp:
921     def __init__(self, master):
922         self.master = master
923         self.master.title('Price Prediction')
924
925         # Read dataset for training and applying one
    -hot encoding
926         self.data = pd.read_csv('Train.csv')
927
928         # Applying one-hot encoding
929         self.data = pd.get_dummies(self.data)
930
931         # List to potentially store slider widgets
932         self.sliders = []
933
934         # Separating features (X) and target
    variable (y)
935         self.X = self.data.drop('Selling_Price',
    axis=1).values
936         self.y = self.data['Selling_Price'].values
937
938         # Splitting data into training and testing
    sets
939         self.X_train, self.X_test, self.y_train,
    self.y_test = train_test_split(self.X, self.y,
```

```
939 test_size=0.2, random_state=42)
940 
941         # Initialize and train the XGBoost
    regression model
942         self.model = XGBRegressor()
943         self.model.fit(self.X_train, self.y_train)
944 
945         # Create GUI widgets for user interaction
946         self.create_widgets()
947 
948     def create_widgets(self):
949         raw_data = pd.read_csv('Train.csv')
950         dropdowns = {}
951 
952         # To track rows for placing widgets
953         row_idx = 0
954 
955         # Loop through each categorical column to
    create dropdown menus
956         for column in ['Product_Brand', '
    Item_Category', 'Subcategory_1', 'Subcategory_2']:
957             label = tk.Label(self.master, text=
    column + ': ')
958             label.grid(row=row_idx, column=0)
959             options = list(raw_data[column].unique
    ())
960             # Sort options alphabetically
961             options = sorted(options)
962         # for storing dropdown selection
963             selected_value = tk.StringVar(self.
    master)
964         # Default to first option
965             selected_value.set(options[0])
966             dropdown = ttk.Combobox(self.master,
    textvariable=selected_value, values=options)
967             dropdown.grid(row=row_idx, column=1)
968             dropdowns[column] = selected_value
969             row_idx += 1
970 
971         # Button for price prediction
972         predict_button = tk.Button(self.master, text
```

```
 972 ='Predict Price', command=lambda: self.
     predict_price(dropdowns))
 973         predict_button.grid(row=row_idx, columnspan
     =2)
 974 
 975     def predict_price(self, dropdowns):
 976         # Convert dropdown selections to model
     input format
 977         input_data = {col: val.get() for col, val
     in dropdowns.items()}
 978 
 979         # Create a new DataFrame and convert to one
     -hot encoding format
 980         input_df = pd.DataFrame([input_data])
 981         input_df = pd.get_dummies(input_df)
 982         # Fill missing columns with 0
 983         for col in self.data.columns:
 984             if col not in input_df.columns:
 985                 input_df[col] = 0
 986 
 987         # Ensure the correct feature order
 988         input_df = input_df[self.data.drop('
     Selling_Price', axis=1).columns]
 989 
 990         # Predict and Print results
 991         price = self.model.predict(input_df)
 992         messagebox.showinfo('Predicted Price', f'
     The predicted selling price is ${price[0]:.2f}')
 993 
 994 
 995 if __name__ == '__main__':
 996     root = tk.Tk()
 997     app = SellingPricePredictionApp(root)
 998     root.mainloop()
 999 
1000 #%% md
1001 # END OF PROGRAMMING PROJECT
```