Step 1

```
# Reading the dataset
import pandas as pd
import numpy as np
clothesprice=pd.read_csv('/content/drive/MyDrive/ST1/clothes_price_prediction_data.csv', encoding='latin')
print('Shape before deleting duplicate values:', clothesprice.shape)

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

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

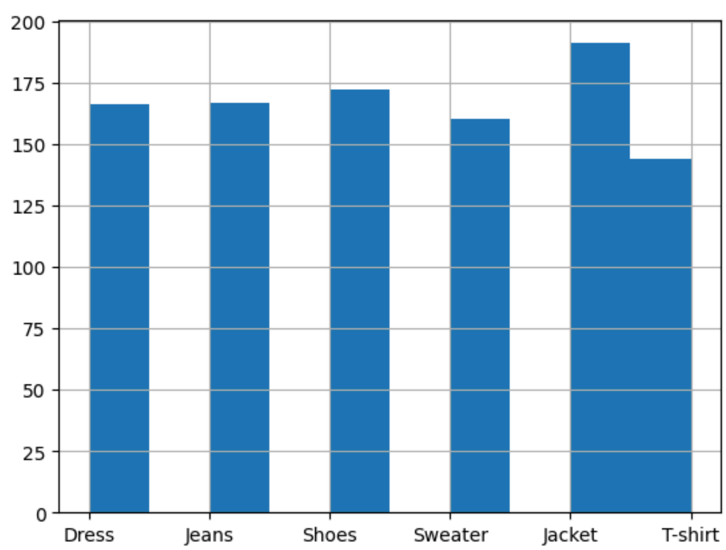
```
Shape before deleting duplicate values: (1000, 6)
Shape After deleting duplicate values: (1000, 6)
```

	Brand	Category	Color	Size	Material	Price
0	New Balance	Dress	White	XS	Nylon	182
1	New Balance	Jeans	Black	XS	Silk	57
2	Under Armour	Dress	Red	M	Wool	127
3	Nike	Shoes	Green	M	Cotton	77
4	Adidas	Sweater	White	M	Nylon	113
5	Reebok	Jacket	Red	XL	Nylon	19
6	Puma	Jacket	Red	XXL	Polyester	31
7	Adidas	Dress	Red	XS	Denim	46
8	Reebok	Dress	Black	S	Wool	97
9	Adidas	Jeans	Yellow	L	Wool	80

step 4

```
#stage4
%matplotlib inline
# Creating histogram as the Target variable is Continuous
# This will help us to understand the distribution of the MEDV values
clothesprice['Category'].hist()
```

```
<Axes: >
```



Step 5

```
# Looking at sample rows in the data
clothesprice.head()
```

	Brand	Category	Color	Size	Material	Price
0	New Balance	Dress	White	XS	Nylon	182
1	New Balance	Jeans	Black	XS	Silk	57
2	Under Armour	Dress	Red	M	Wool	127
3	Nike	Shoes	Green	M	Cotton	77
4	Adidas	Sweater	White	M	Nylon	113

```
#stage5
clothesprice.tail()
```

	Brand	Category	Color	Size	Material	Price
995	Puma	Jeans	Black	L	Polyester	176
996	Puma	Jacket	Red	XXL	Silk	110
997	Reebok	Sweater	Blue	XS	Denim	127
998	Under Armour	Sweater	Black	XXL	Denim	69
999	New Balance	Jacket	Yellow	XS	Wool	174

```
clothesprice.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1000 entries, 0 to 999
Data columns (total 6 columns):
 #   Column    Non-Null Count  Dtype 
---  ------    --------------  ----- 
 0   Brand     1000 non-null   object
 1   Category  1000 non-null   object
 2   Color     1000 non-null   object
 3   Size      1000 non-null   object
 4   Material  1000 non-null   object
 5   Price     1000 non-null   int64 
dtypes: int64(1), object(5)
memory usage: 47.0+ KB
```

```
# Looking at the descriptive statistics of the data
clothesprice.describe(include='all')
```

	Brand	Category	Color	Size	Material	Price
count	1000	1000	1000	1000	1000	1000.000000
unique	6	6	6	6	6	NaN
top	Under Armour	Jacket	Yellow	XS	Polyester	NaN
freq	179	191	173	196	175	NaN
mean	NaN	NaN	NaN	NaN	NaN	106.289000
std	NaN	NaN	NaN	NaN	NaN	53.695444
min	NaN	NaN	NaN	NaN	NaN	10.000000
25%	NaN	NaN	NaN	NaN	NaN	59.750000
50%	NaN	NaN	NaN	NaN	NaN	108.000000
75%	NaN	NaN	NaN	NaN	NaN	150.000000
max	NaN	NaN	NaN	NaN	NaN	199.000000

```
# Finging unique values for each column
# TO understand which column is categorical and which one is Continuous
```

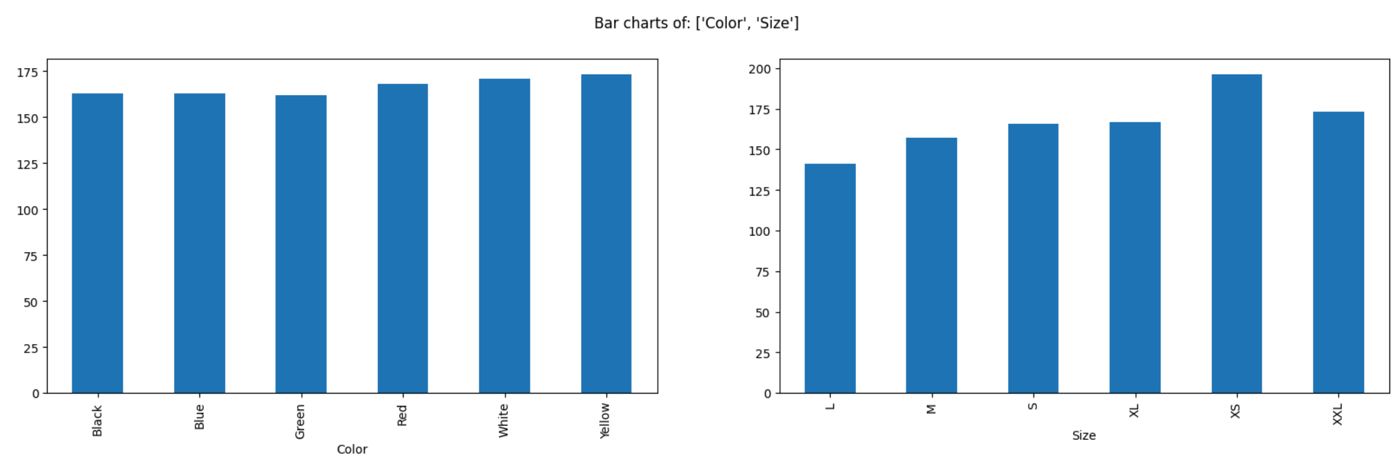
```
# Typically if the numer of unique values are < 20 then the variable is likely to be a category otherwise continuous
clothesprice.nunique()
```

	0
Brand	6
Category	6
Color	6
Size	6
Material	6
Price	190

dtype: int64

Step 8

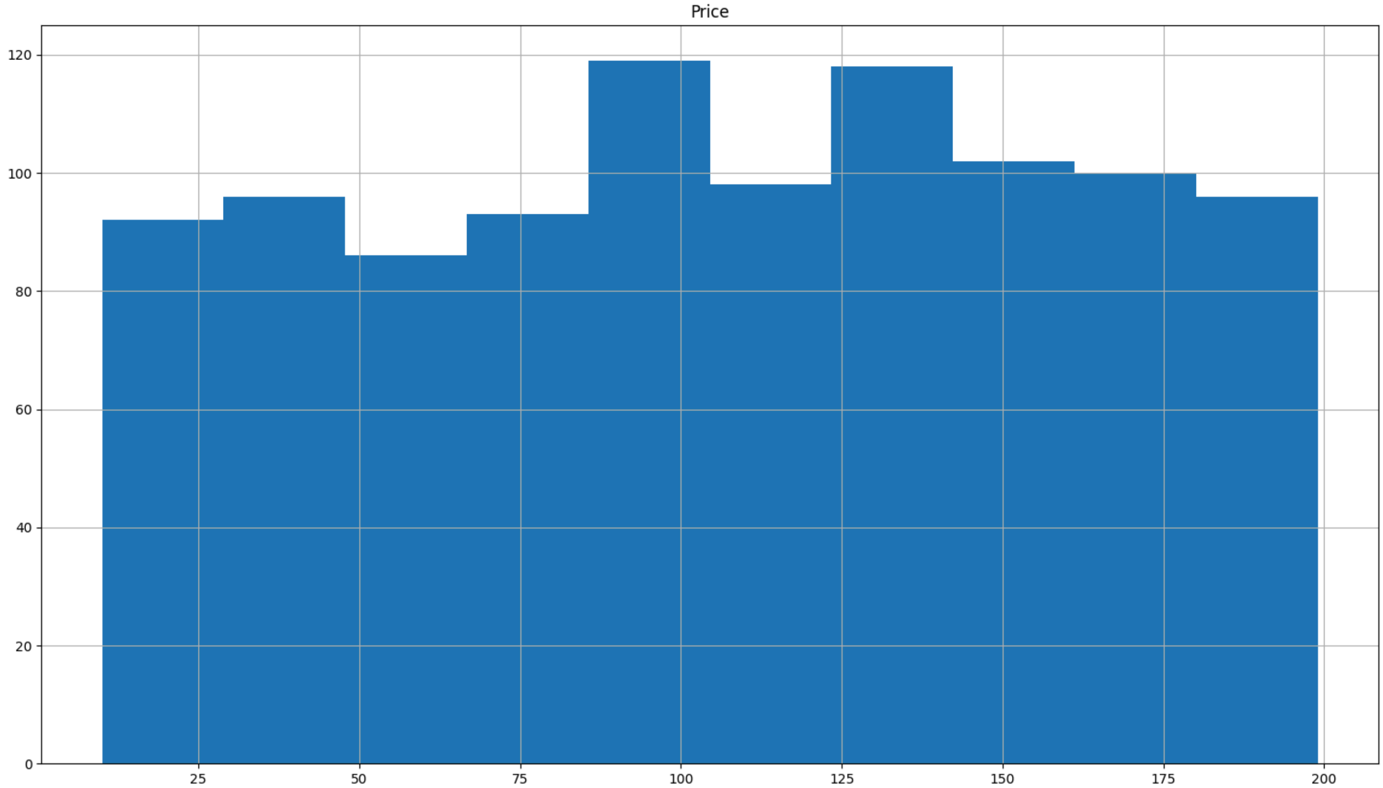
```
#stage8
# Plotting multiple bar charts at once for categorical variables
# Since there is no default function which can plot bar charts for multiple columns at once
# we are defining our own function for the same

def PlotBarCharts(inpData, colsToPlot):
    %matplotlib inline

    import matplotlib.pyplot as plt

    # Generating multiple subplots
    fig, subPlot=plt.subplots(nrows=1, ncols=len(colsToPlot), figsize=(20,5))
    fig.suptitle('Bar charts of: '+ str(colsToPlot))

    for colName, plotNumber in zip(colsToPlot, range(len(colsToPlot))):
        inpData.groupby(colName).size().plot(kind='bar',ax=subPlot[plotNumber])

#####################################################################
# Calling the function PlotBarCharts() we have created
PlotBarCharts(inpData=clothesprice, colsToPlot=['Color','Size'])
```



step 9

```
#stage9
# Plotting histograms of multiple columns together
clothesprice.hist(['Brand', 'Category', 'Color', 'Size', 'Material', 'Price'], figsize=(18,10))
```

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




```
# Finding nearest values to 100 mark
clothesprice['Price'][clothesprice['Price']>100].sort_values(ascending=False)
```

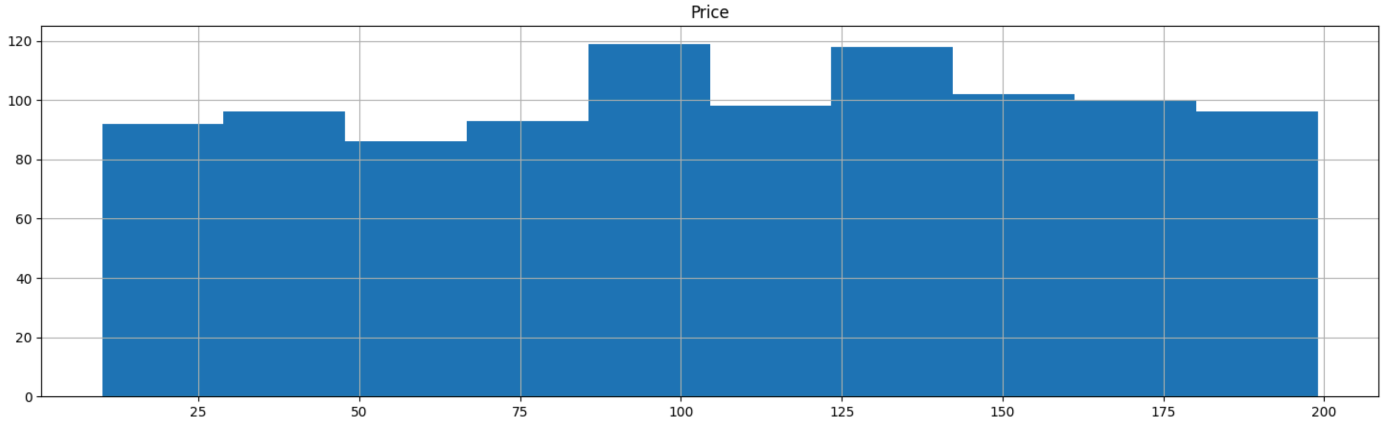
	Price
194	199
277	199
223	199
619	198
575	198
...	...
466	101
473	101
897	101
551	101
521	101

545 rows × 1 columns

Step 11

```
clothesprice.hist(['Price'], figsize=(18,5))
```

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



step 12

```
# Finding how many missing values are there for each column
clothesprice.isnull().sum()
```

	0
Brand	0
Category	0
Color	0
Size	0
Material	0
Price	0

step 14

```
# Calculating correlation matrix
ContinuousCols=['Price']  # Only include numerical columns

# Creating the correlation matrix
CorrelationData=clothesprice[ContinuousCols].corr()
CorrelationData
```

	Price
Price	1.0

```
# Filtering only those columns where absolute correlation > 0.5 with Target Variable
# reduce the 0.5 threshold if no variable is selected
CorrelationData['Price'][abs(CorrelationData['Price']) > 0.5 ]
```

	Price
Price	1.0

```
ContinuousCols=['Price']

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

step 16

```
#step 16
# 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])
    return(SelectedPredictors)

#Calling the function to check which categorical variables are correlated with target
CategoricalPredictorList=['Price', 'Size']
FunctionAnova(inpData=clothesprice,
              TargetVariable='Price',
              CategoricalPredictorList=CategoricalPredictorList)
```

```
##### ANOVA Results #####

Price is correlated with Price | P-Value: 0.0
Size is NOT correlated with Price | P-Value: 0.7098835429591747
/usr/local/lib/python3.10/dist-packages/scipy/stats/_axis_nan_policy.py:531: ConstantInputWarning: Each of the input arrays is constant;
  res = hypotest_fun_out(*samples, **kwds)
['Price']
```

```
SelectedColumns=['Brand', 'Category','Color', 'Size', 'Price']

# Selecting final columns
DataForML=clothesprice[SelectedColumns]
DataForML.head()
```

	Brand	Category	Color	Size	Price
0	New Balance	Dress	White	XS	182
1	New Balance	Jeans	Black	XS	57
2	Under Armour	Dress	Red	M	127
3	Nike	Shoes	Green	M	77
4	Adidas	Sweater	White	M	113

```
# Saving this final data subset for reference during deployment
DataForML.to_pickle('DataForML.pkl')
```

Step 17

```
# Treating all the nominal variables at once using dummy variables
DataForML_Numeric = pd.get_dummies(DataForML)
# Use the DataFrame 'DataForML' which is already defined

# Adding Target Variable to the data
DataForML_Numeric['Price'] = clothesprice['Price']

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

	Price	Brand_Adidas	Brand_New Balance	Brand_Nike	Brand_Puma	Brand_Reebok	Brand_Under Armour	Category_Dress	Category_Jacket	Category_Jeans	..
0	182	False	True	False	False	False	False	True	False	False	
1	57	False	True	False	False	False	False	False	False	True	
2	127	False	False	False	False	False	True	True	False	False	
3	77	False	False	True	False	False	False	False	False	False	
4	113	True	False	False	False	False	False	False	False	False	

5 rows × 25 columns

step 18

```
# Printing all the column names for our reference
DataForML_Numeric.columns
```

```
Index(['Price', 'Brand_Adidas', 'Brand_New Balance', 'Brand_Nike',
       'Brand_Puma', 'Brand_Reebok', 'Brand_Under Armour', 'Category_Dress',
       'Category_Jacket', 'Category_Jeans', 'Category_Shoes',
       'Category_Sweater', 'Category_T-shirt', 'Color_Black', 'Color_Blue',
       'Color_Green', 'Color_Red', 'Color_White', 'Color_Yellow', 'Size_L',
       'Size_M', 'Size_S', 'Size_XL', 'Size_XS', 'Size_XXL'],
      dtype='object')
```

```
SelectedColumns=['Brand', 'Category','Color', 'Size', 'Material']

# Selecting final columns
DataForML=clothesprice[SelectedColumns]
DataForML.head()

# Saving this final data subset for reference during deployment
DataForML.to_pickle('DataForML.pkl')


#Separate Target Variable and Predictor Variables
TargetVariable='Price'
Predictors=list(DataForML_Numeric.columns) # Get all column names from DataFrame
Predictors.remove(TargetVariable) # Remove the target variable from the predictor list

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

```
#STEP 19
# Treating all the nominal variables at once using dummy variables
DataForML_Numeric=pd.get_dummies(DataForML)

# Adding Target Variable to the data
DataForML_Numeric['Price']=clothesprice['Price']

### Standardization 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()

#Separate Target Variable and Predictor Variables
TargetVariable='Price'
Predictors=list(DataForML_Numeric.columns) # Get all column names from DataFrame
Predictors.remove(TargetVariable) # Remove the target variable from the predictor list

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

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

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

# 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=42)
# Sanity check for the sampled data
print(X_train.shape)
print(y_train.shape)
print(X_test.shape)
print(y_test.shape)
```

```
(700, 30)
(700,)
(300, 30)
(300,)
```

step 20

```
#Step 20
#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)
# Printing sample prediction values
print(TestingDataResults.head())

# Calculating the error for each row
TestingDataResults['Error'] = 100 * ((abs(
  TestingDataResults['Price']-TestingDataResults['PredictedPrice']))/TestingDataResults['Price'])

# Use the newly created 'Error' column to calculate MAPE
MAPE=np.mean(TestingDataResults['Error'])
MedianMAPE=np.median(TestingDataResults['Error'])

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: 0.014863520562398258

##### Model Validation and Accuracy Calculations ##########
   Brand_Adidas  Brand_New Balance  Brand_Nike  Brand_Puma  Brand_Reebok  \
0           0.0                0.0         0.0         0.0           0.0
1           0.0                1.0         0.0         0.0           0.0
2           0.0                0.0         0.0         0.0           1.0
3           0.0                0.0         1.0         0.0           0.0
4           0.0                1.0         0.0         0.0           0.0
```

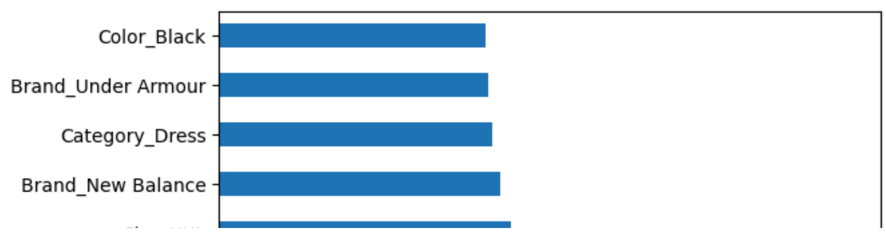
```
   Brand_Under Armour  Category_Dress  Category_Jacket  Category_Jeans  \
0                 1.0             1.0              0.0             0.0
1                 0.0             1.0              0.0             0.0
2                 0.0             0.0              1.0             0.0
3                 0.0             0.0              0.0             0.0
4                 0.0             0.0              0.0             0.0

   Category_Shoes  ...  Size_XS  Size_XXL  Material_Cotton  Material_Denim  \
0             0.0  ...      0.0       0.0              0.0             1.0
1             0.0  ...      0.0       1.0              0.0             0.0
2             0.0  ...      0.0       0.0              0.0             0.0
3             0.0  ...      0.0       0.0              0.0             0.0
4             0.0  ...      0.0       0.0              1.0             0.0

   Material_Nylon  Material_Polyester  Material_Silk  Material_Wool  Price  \
0             0.0                 0.0            0.0            0.0    101
1             1.0                 0.0            0.0            0.0    164
2             0.0                 0.0            0.0            1.0     58
3             0.0                 0.0            0.0            1.0     82
4             0.0                 0.0            0.0            0.0    177

   PredictedPrice
0           110.0
1           140.0
2           119.0
3            95.0
4           108.0

[5 rows x 32 columns]
Mean Accuracy on test data: -1.3676545919518759
Median Accuracy on test data: 64.140486876105

Accuracy values for 10-fold Cross Validation:
 [ 15.02254318 -14.61235203 -15.28587943  15.23577135   2.72574755
  23.36559313  41.66350595   0.36452287  -7.79736327   2.43011178]

Final Average Accuracy of the model: 6.31
```

Decision Trees

```
# Decision Trees (Multiple if-else statements!)
from sklearn.tree import DecisionTreeRegressor
RegModel = DecisionTreeRegressor(max_depth=5,criterion='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['Error'] = 100 * ((abs( # Changed 'Brand' to 'Error' for clarity
  TestingDataResults['Price']-TestingDataResults['PredictedPrice']))/TestingDataResults['Price'])

MAPE = np.mean(TestingDataResults['Error']) # Changed 'Brand' to 'MAPE'
MedianMAPE = np.median(TestingDataResults['Error']) # Changed 'MedianBrand' to 'MedianMAPE'

Accuracy = 100 - MAPE # Changed 'Price' to 'Accuracy'
MedianAccuracy = 100 - MedianMAPE # Changed 'MedianPricr' to 'MedianAccuracy'
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) # Changed 'Price_Score' to 'Accuracy_Score'

# 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) # Changed 'Price values' to 'Accuracy values'
print('\nFinal Average Accuracy of the model:', round(Accuracy_Values.mean(),2))
```

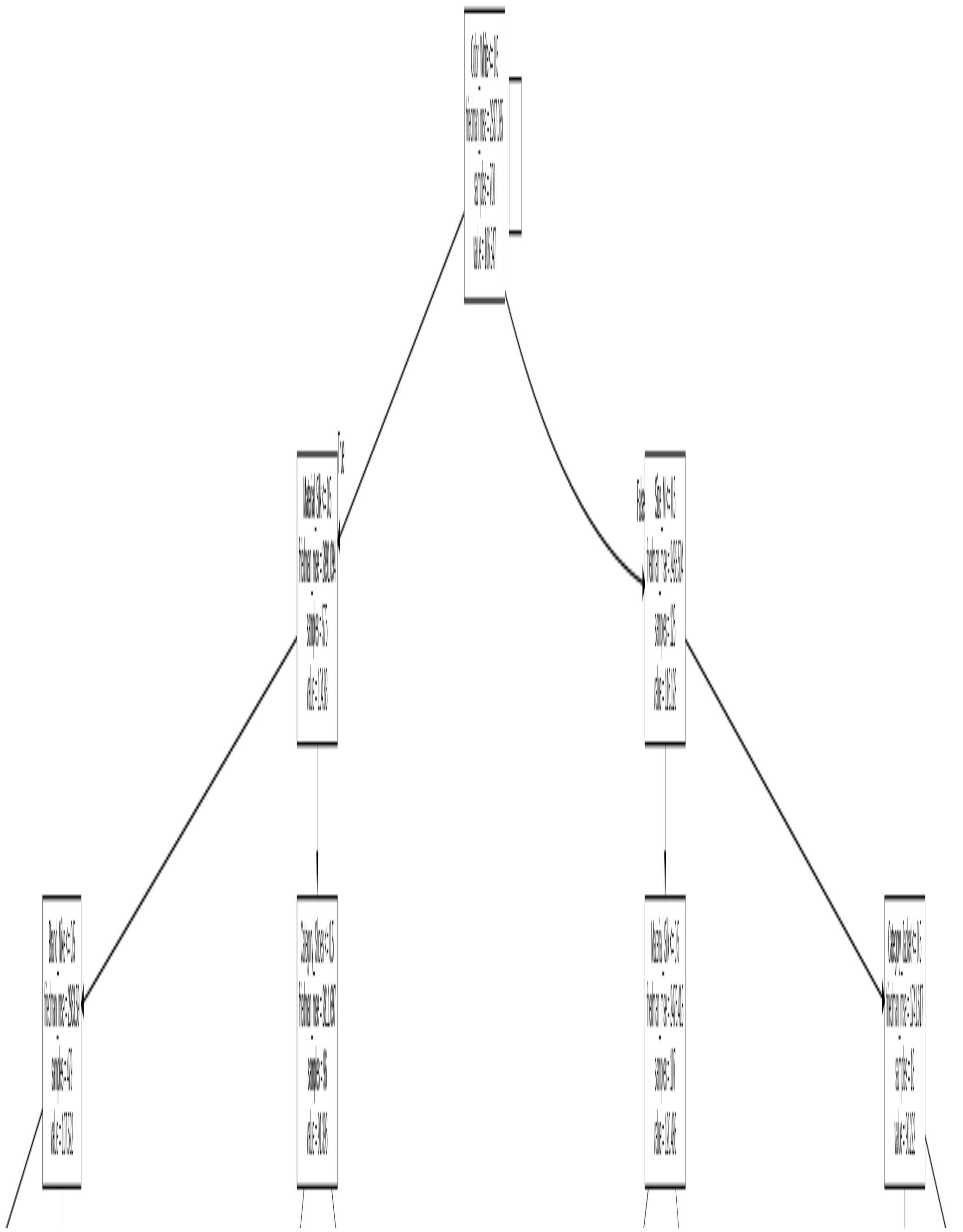
```
DecisionTreeRegressor(criterion='friedman_mse', max_depth=5)
R2 Value: 0.12400587905826077

##### Model Validation and Accuracy Calculations ##########
   Brand_Adidas  Brand_New Balance  Brand_Nike  Brand_Puma  Brand_Reebok  \
0           0.0                0.0         0.0         0.0           0.0
1           0.0                1.0         0.0         0.0           0.0
2           0.0                0.0         0.0         0.0           1.0
3           0.0                0.0         1.0         0.0           0.0
4           0.0                1.0         0.0         0.0           0.0

   Brand_Under Armour  Category_Dress  Category_Jacket  Category_Jeans  \
0                 1.0             1.0              0.0             0.0
1                 0.0             1.0              0.0             0.0
2                 0.0             0.0              1.0             0.0
3                 0.0             0.0              0.0             0.0
4                 0.0             0.0              0.0             0.0

   Category_Shoes  ...  Size_XS  Size_XXL  Material_Cotton  Material_Denim  \
0             0.0  ...      0.0       0.0              0.0             1.0
1             0.0  ...      0.0       1.0              0.0             0.0
2             0.0  ...      0.0       0.0              0.0             0.0
3             0.0  ...      0.0       0.0              0.0             0.0
4             0.0  ...      0.0       0.0              1.0             0.0

   Material_Nylon  Material_Polyester  Material_Silk  Material_Wool  Price  \
0             0.0                 0.0            0.0            0.0    101
1             1.0                 0.0            0.0            0.0    164
2             0.0                 0.0            0.0            1.0     58
3             0.0                 0.0            0.0            1.0     82
4             0.0                 0.0            0.0            0.0    177

   PredictedPrice
0           111.0
1           117.0
2           117.0
3            68.0
4           122.0

[5 rows x 32 columns]
Mean Accuracy on test data: -2.511594712211334
Median Accuracy on test data: 61.77429876060013

Accuracy values for 10-fold Cross Validation:
 [ 11.0156758  -17.45120437 -16.66219667   7.97239132   2.20750313
  21.34070157  37.86298324  -3.40259221  -8.34181693   2.79206896]

Final Average Accuracy of the model: 3.73
```



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=None) # Set class_names to None for regression

# 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
```

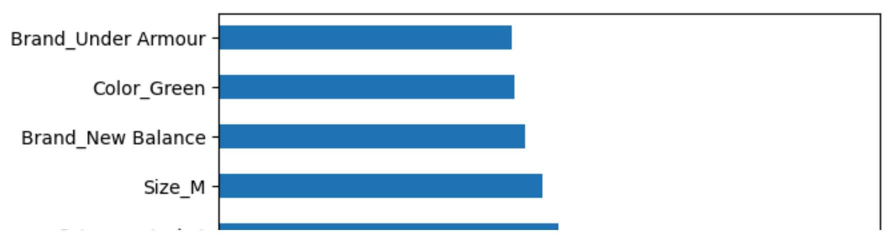


Random Forest

```
# 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 TargetVariable name here
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: 0.12352160921145428

##### Model Validation and Accuracy Calculations ##########
   Brand_Adidas  Brand_New Balance  Brand_Nike  Brand_Puma  Brand_Reebok  \
0           0.0                0.0         0.0         0.0           0.0
1           0.0                1.0         0.0         0.0           0.0
2           0.0                0.0         0.0         0.0           1.0
3           0.0                0.0         1.0         0.0           0.0
4           0.0                1.0         0.0         0.0           0.0

   Brand_Under Armour  Category_Dress  Category_Jacket  Category_Jeans  \
0                 1.0             1.0              0.0             0.0
1                 0.0             1.0              0.0             0.0
2                 0.0             0.0              1.0             0.0
3                 0.0             0.0              0.0             0.0
4                 0.0             0.0              0.0             0.0

   Category_Shoes  ...  Size_XS  Size_XXL  Material_Cotton  Material_Denim  \
0             0.0  ...      0.0       0.0              0.0             1.0
1             0.0  ...      0.0       1.0              0.0             0.0
2             0.0  ...      0.0       0.0              0.0             0.0
3             0.0  ...      0.0       0.0              0.0             0.0
4             0.0  ...      0.0       0.0              1.0             0.0

   Material_Nylon  Material_Polyester  Material_Silk  Material_Wool  Price  \
0             0.0                 0.0            0.0            0.0    101
1             1.0                 0.0            0.0            0.0    164
2             0.0                 0.0            0.0            1.0     58
3             0.0                 0.0            0.0            1.0     82
4             0.0                 0.0            0.0            0.0    177

   PredictedPrice
0           113.0
1           123.0
2           113.0
3            97.0
4           113.0

[5 rows x 32 columns]
Mean Accuracy on test data: -1.9880156600036685
Median Accuracy on test data: 64.12123734221112

Accuracy values for 10-fold Cross Validation:
 [ 15.22335125 -14.08562698 -19.45736589  13.8894993    3.36194617
  22.34754604  42.72433119  -0.25287309  -7.18690777   4.83669839]

Final Average Accuracy of the model: 6.14
```



K-Nearest Neighbor(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['Price']-TestingDataResults['PredictedPrice']))/TestingDataResults['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))
```

```
KNeighborsRegressor(n_neighbors=3)
R2 Value: 0.3435829034903999

##### Model Validation and Accuracy Calculations ##########
   Brand_Adidas  Brand_New Balance  Brand_Nike  Brand_Puma  Brand_Reebok  \
0           0.0                0.0         0.0         0.0           0.0
1           0.0                1.0         0.0         0.0           0.0
2           0.0                0.0         0.0         0.0           1.0
3           0.0                0.0         1.0         0.0           0.0
4           0.0                1.0         0.0         0.0           0.0

   Brand_Under Armour  Category_Dress  Category_Jacket  Category_Jeans  \
0                 1.0             1.0              0.0             0.0
1                 0.0             1.0              0.0             0.0
2                 0.0             0.0              1.0             0.0
3                 0.0             0.0              0.0             0.0
4                 0.0             0.0              0.0             0.0

   Category_Shoes  ...  Size_XS  Size_XXL  Material_Cotton  Material_Denim  \
0             0.0  ...      0.0       0.0              0.0             1.0
1             0.0  ...      0.0       1.0              0.0             0.0
2             0.0  ...      0.0       0.0              0.0             0.0
3             0.0  ...      0.0       0.0              0.0             0.0
4             0.0  ...      0.0       0.0              1.0             0.0

   Material_Nylon  Material_Polyester  Material_Silk  Material_Wool  Price  \
0             0.0                 0.0            0.0            0.0    101
1             1.0                 0.0            0.0            0.0    164
2             0.0                 0.0            0.0            1.0     58
3             0.0                 0.0            0.0            1.0     82
4             0.0                 0.0            0.0            0.0    177
```

```
   PredictedPrice
0           171.0
1           138.0
2           143.0
3            74.0
4           147.0

[5 rows x 32 columns]
Mean Accuracy on test data: -10.479903213317982
Median Accuracy on test data: 59.625943719972554

Accuracy values for 10-fold Cross Validation:
 [ 14.57551381 -25.10075877 -26.97418494  10.23515941  -3.39251075
  16.38812119  33.02316696  -0.30122851 -10.93325665   0.26162594]

Final Average Accuracy of the model: 0.78
```

Support Vector Machines(SVM)

```
# 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 # Commented out as it is not directly related to the error
#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
# Get the predictor names directly from X_test
TestingDataResults=pd.DataFrame(data=X_test, columns=[f"feature_{i}" for i in range(X_test.shape[1])])
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 TargetVariable name here and consistent column naming
TestingDataResults['APE']=100 * ((abs(
  TestingDataResults[TargetVariable]-TestingDataResults['Predicted'+TargetVariable]))/TestingDataResults[TargetVariable])

# Calculate MAPE using the 'APE' column
MAPE=np.mean(TestingDataResults['APE'])
MedianMAPE=np.median(TestingDataResults['APE']) # Use 'APE' for median as well

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: 0.026671413627403107

##### Model Validation and Accuracy Calculations ##########
   feature_0  feature_1  feature_2  feature_3  feature_4  feature_5  \
0        0.0        0.0        0.0        0.0        0.0        1.0
1        0.0        1.0        0.0        0.0        0.0        0.0
2        0.0        0.0        0.0        0.0        1.0        0.0
3        0.0        0.0        1.0        0.0        0.0        0.0
4        0.0        1.0        0.0        0.0        0.0        0.0

   feature_6  feature_7  feature_8  feature_9  ...  feature_22  feature_23  \
0        1.0        0.0        0.0        0.0  ...         0.0         0.0
1        1.0        0.0        0.0        0.0  ...         0.0         1.0
2        0.0        1.0        0.0        0.0  ...         0.0         0.0
3        0.0        0.0        0.0        0.0  ...         0.0         0.0
4        0.0        0.0        0.0        0.0  ...         0.0         0.0

   feature_24  feature_25  feature_26  feature_27  feature_28  feature_29  \
0         0.0         1.0         0.0         0.0         0.0         0.0
1         0.0         0.0         1.0         0.0         0.0         0.0
2         0.0         0.0         0.0         0.0         0.0         1.0
3         0.0         0.0         0.0         0.0         0.0         1.0
4         1.0         0.0         0.0         0.0         0.0         0.0

   Price  PredictedPrice
0    101           100.0
1    164           130.0
2     58           115.0
3     82           106.0
4    177           111.0

[5 rows x 32 columns]
Mean Accuracy on test data: -0.9438266220202394
Median Accuracy on test data: 63.720238095238095

Accuracy values for 10-fold Cross Validation:
 [ 13.92874747 -15.67631295 -18.8692552   13.12078928   1.4835095
  22.86308339  42.66284181  -0.84458924  -9.55705589   2.76292976]

Final Average Accuracy of the model: 5.19
```

Step 21

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

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

# Verify that 'DataForML_Numeric' contains the specified columns
print(DataForML_Numeric.columns)  # Print columns to check for typos or missing columns

# Proceed with extracting X and y if columns are present
if all(col in DataForML_Numeric.columns for col in Predictors):
    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)
else:
    print("Error: One or more predictor columns not found in the DataFrame.")
```

```
Index(['Brand_Adidas', 'Brand_New Balance', 'Brand_Nike', 'Brand_Puma',
       'Brand_Reebok', 'Brand_Under Armour', 'Category_Dress',
       'Category_Jacket', 'Category_Jeans', 'Category_Shoes',
       'Category_Sweater', 'Category_T-shirt', 'Color_Black', 'Color_Blue',
       'Color_Green', 'Color_Red', 'Color_White', 'Color_Yellow', 'Size_L',
       'Size_M', 'Size_S', 'Size_XL', 'Size_XS', 'Size_XXL', 'Material_Cotton',
       'Material_Denim', 'Material_Nylon', 'Material_Polyester',
       'Material_Silk', 'Material_Wool', 'Price'],
      dtype='object')
Error: One or more predictor columns not found in the DataFrame.
```

XGBoost 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
# Get the feature names from the training data
feature_names = [f"feature_{i}" for i in range(X_train.shape[1])] # Changed Predictors to feature names from training data
feature_importances = pd.Series(XGB.feature_importances_, index=feature_names)
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=feature_names) # Changed Predictors to feature names
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']) # Changed 'Size' to 'APE'
MedianMAPE=np.median(TestingDataResults['APE']) # Changed 'Size' to '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: 0.3162134605182447
/usr/local/lib/python3.10/dist-packages/xgboost/core.py:158: UserWarning: [02:51:33] WARNING: /workspace/src/objective/regression_obj
  warnings.warn(smsg, UserWarning)

##### Model Validation and Accuracy Calculations ##########
   feature_0  feature_1  feature_2  feature_3  feature_4  feature_5  \
0        0.0        0.0        0.0        0.0        0.0        1.0
1        0.0        1.0        0.0        0.0        0.0        0.0
2        0.0        0.0        0.0        0.0        1.0        0.0
3        0.0        0.0        1.0        0.0        0.0        0.0
4        0.0        1.0        0.0        0.0        0.0        0.0

   feature_6  feature_7  feature_8  feature_9  ...  feature_22  feature_23  \
0        1.0        0.0        0.0        0.0  ...         0.0         0.0
1        1.0        0.0        0.0        0.0  ...         0.0         1.0
2        0.0        1.0        0.0        0.0  ...         0.0         0.0
3        0.0        0.0        0.0        0.0  ...         0.0         0.0
4        0.0        0.0        0.0        0.0  ...         0.0         0.0

   feature_24  feature_25  feature_26  feature_27  feature_28  feature_29  \
0         0.0         1.0         0.0         0.0         0.0         0.0
1         0.0         0.0         1.0         0.0         0.0         0.0
2         0.0         0.0         0.0         0.0         0.0         1.0
3         0.0         0.0         0.0         0.0         0.0         1.0
4         1.0         0.0         0.0         0.0         0.0         0.0

   Price  PredictedPrice
0    101           158.0
1    164           124.0
2     58           145.0
3     82            69.0
4    177           145.0

[5 rows x 32 columns]
Mean Accuracy on test data: -6.8816606996500695
Median Accuracy on test data: 61.69192700278379
/usr/local/lib/python3.10/dist-packages/xgboost/core.py:158: UserWarning: [02:51:34] WARNING: /workspace/src/objective/regression_obj
  warnings.warn(smsg, UserWarning)
/usr/local/lib/python3.10/dist-packages/xgboost/core.py:158: UserWarning: [02:51:35] WARNING: /workspace/src/objective/regression_obj
  warnings.warn(smsg, UserWarning)

Accuracy values for 10-fold Cross Validation:
 [ 18.69964401 -15.81527848 -22.3655382   11.79850295   0.84854619
  18.66850336  38.96042632   2.83818335 -10.58770325   5.52290842]

Final Average Accuracy of the model: 4.86
```

Cross validating the final model accuracy with less predictors

```
# 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))
```

```
/usr/local/lib/python3.10/dist-packages/xgboost/core.py:158: UserWarning: [02:46:41] WARNING: /workspace/src/objective/regression_obj.cu
  warnings.warn(smsg, UserWarning)
/usr/local/lib/python3.10/dist-packages/xgboost/core.py:158: UserWarning: [02:46:43] WARNING: /workspace/src/objective/regression_obj.cu
  warnings.warn(smsg, UserWarning)
/usr/local/lib/python3.10/dist-packages/xgboost/core.py:158: UserWarning: [02:46:45] WARNING: /workspace/src/objective/regression_obj.cu
  warnings.warn(smsg, UserWarning)
/usr/local/lib/python3.10/dist-packages/xgboost/core.py:158: UserWarning: [02:46:46] WARNING: /workspace/src/objective/regression_obj.cu
  warnings.warn(smsg, UserWarning)

Accuracy values for 10-fold Cross Validation:
```