Dataset-based Implementation

Objective

Using 'Price Prediction of USA High Quality Backpacks' dataset, objective is to

1. **Perform Exploratory Data Analysis (EDA)**
2. **Build Classification, Regression and Ensemble algorithm along with supporting activities.**

Dataset

1. Following url is provided as per the problem statement:
    - url=https://drive.google.com/file/d/18NNIgognzWnwPbclCkdak6oTABR92lUw/view

Libraries Used

Library	Description
pandas	For file operations
numpy	For mathematical operations
seaborn, matplotlib	For graphical plotting
wordcloud	For EDA
sklearn	For data processing, ML algorithm development
missingno	To look for missing values

Load and Pre-process '*Backpacks*' file

The data was loaded into the Jupyter notebook using the URL provided in the problem statement

- Load and explore the data using the given URL: 'USA High Quality Made - Backpacks.csv'
  1. Found 107 rows and 12 columns in total

```
In [5]: #inspect dataframe
        backpacks.describe()
```

Out[5]:

	Product	Brand	Country of Origin	Type	Image URL	Laptop Compartment	Capacity (L)	Material	Average Weight (Kg)	Waterproof	Wa repell
count	107	107	107	107	107	107	107	107	107	107	
unique	107	9	1	18	107	5	47	77	50	1	
top	Surge Backpack	The North Face	USA	Daypack/Outdoor	https://images.thenorthface.com/is/image/TheNo...	Yes	25	210D recycled nylon ripstop with non-PFC durab...	Not Specified	No	
freq	1	26	107	57	1	75	9	9	25	107	

  - Following are the observations from the dataset:
    - There are 107 unique backpacks and image urls.
    - There is a need to check null values across columns.
    - There are 18 'types'. Unique values are to be explored.
    - There are 77 materials. Extraction of the primary material for each backpack is planned to be carried out
    - Every entry on the dataset is not waterproof but is water-repellent. Similar is observed for 'Country of Origin'. These columns can be dropped

  2. Identified the following columns
     - Product | Brand | Country of Origin | Type | Image URL | Laptop Compartment | Capacity (L) | Material | Average Weight (Kg) | Waterproof | Water-repellent | Price (USD)

Data Cleaning

Effective data cleaning is a vital part of the data analytics process to improve the quality of data and support the data-mining program. Data cleaning is important because the clean data eases data mining and helps in making a successful strategic decision. Data cleaning involves tackling the missing data and smoothing noisy data.

- Followed the sequence of steps to arrive at clean data:
  1. Removing duplicate and irrelevant observations
     - Since no duplicate was observed in the dataset, so no row has been dropped

```
print('Number of duplicate rows found in the dataset: ',len(backpacks) - len(backpacks.drop_duplicates()))
backpacks = backpacks.drop_duplicates(keep='first')
```

```
Number of duplicate rows found in the dataset:  0
```

  2. Fix structural errors
     - Structural errors were observed in '*Capacity(L)*' and '*Average Weight (Kg)*' columns

- Most of the rows have numeric data. However, some rows have a range defined. This is transformed using an average function to allow machine learning algorithms to function properly

```
#To look at rows where Capacity(L) is either a range of numbers or a set of numbers.
backpacks[backpacks['Capacity (L)'].str.contains('/|-')]
```

	Type	Image URL	Laptop Compartment	Capacity (L)	Material	Average Weight (Kg)	Waterproof	Water-repellent	Price (USD)
	Daypack/Outdoor/Tactical	https://www.mysteryranch.com/Products/Mission%...	NaN	40/55/90	TPU, Shadow 1000D, CORDURA®, Wood...	1.7 - 2.9	No	Yes	250
	Daypack/Traveling/Outdoor/Tactical	https://cdn.shopify.com/s/files/1/0275/4985/99...	Yes	21 - 26	Primary Material (1000D Cordura)	1.3 - 1.5	No	Yes	325
	Daypack/Traveling/Outdoor/Tactical	https://cdn.shopify.com/s/files/1/0275/4985/99...	Yes	26 - 40	Primary Material (1000D Cordura)	1.5 - 1.9	No	Yes	395
	Daypack/Traveling/Outdoor/Tactical	https://cdn.shopify.com/s/files/1/0275/4985/99...	Yes	20 - 25	Primary Material (Cordura)	1.2 - 1.5	No	Yes	225

Row Dropped

Structural Errors → Average

```
#function that converts the list of strings to a list of float and take its average, then return the average.
def take_average(nums_list):
    #When the values are available
    try:
        nums_list = nums_list.split('-')
        nums_list = [float(x) for x in nums_list]
        nums_list = np.array(nums_list)
        ave = np.mean(nums_list)
    #to handle null values
    except:
        ave = np.nan
    return ave
```

- Only one row was observed to have erroneous 'Capacity(L)' value for which average may or may not be the right solution. For this reason that row is dropped.

```
# delete row number 42 invalid data
#backpacks.drop(42, axis = 0, inplace = True)
backpacks.drop(backpacks[backpacks['Capacity (L)'].str.contains('/')].index, axis = 0, inplace = True)
```

```
backpacks[backpacks['Capacity (L)'].str.contains('/')].count()
```

```
Product               0
Brand                 0
Country of Origin     0
Type                  0
Image URL             0
Laptop Compartment    0
Capacity (L)          0
Material              0
Average Weight (Kg)   0
Waterproof            0
Water-repellent       0
Price (USD)           0
dtype: int64
```

- Columns – **'Waterproof', 'Water-repellent' and 'Country of Origin'** are also removed since all data rows have exactly same value. This would not contribute to any learning of Machine Learning algorithms

```
#drop 'Waterproof', 'Water-repellent' and 'Country of Origin' columns due to all rows have same value
backpacks = backpacks.drop(['Waterproof', 'Water-repellent','Country of Origin'], axis = 1)
backpacks.info()
```

- Columns – '**Product** and **'Image URL'** are also removed since they contain unique value in all the rows (Like ID). This would also not contribute to any learning of Machine Learning algorithms
- Column – '**Laptop Compartment**' and '**Average Weight (Kg)'** are dropped due to large number of missing values in that column

```
backpacks = backpacks.drop(['Product','Image URL',],axis=1)   # For columns with unique values (like ID)
```

```
backpacks.isna().sum()
```

```
Brand                  0
Type                   0
Laptop Compartment     26
Capacity (L)           0
Material               0
Average Weight (Kg)    25
Price (USD)            0
dtype: int64
```

On Average Weight

- We could impute the missing values with the average for each type then delete the skiing and travelling types. We could also make an assumption that the material and its capacity will help us determine the weights. But this could lead to misleading data. Thus we will delete the column since we would not gain insight from it.

```
backpacks = backpacks.drop(['Laptop Compartment'],axis=1)     # For columns with large number of missing values
backpacks = backpacks.drop(['Average Weight (Kg)'],axis=1)     # For columns with large number of missing values
```

3. Format Data Type:
   - Column '**Price (USD)**' is converted to float type for data analyses

```
#change Price column into float
backpacks['Price (USD)'] = backpacks['Price (USD)'].astype('float')
#check if data cleansing is finished
backpacks.isna().sum(), backpacks.dtypes
```

```
(Brand           0
 Type            0
 Capacity (L)    0
 Material        0
 Price (USD)     0
 dtype: int64,
 Brand            object
 Type             object
 Capacity (L)    float64
 Material         object
 Price (USD)     float64
 dtype: object)
```

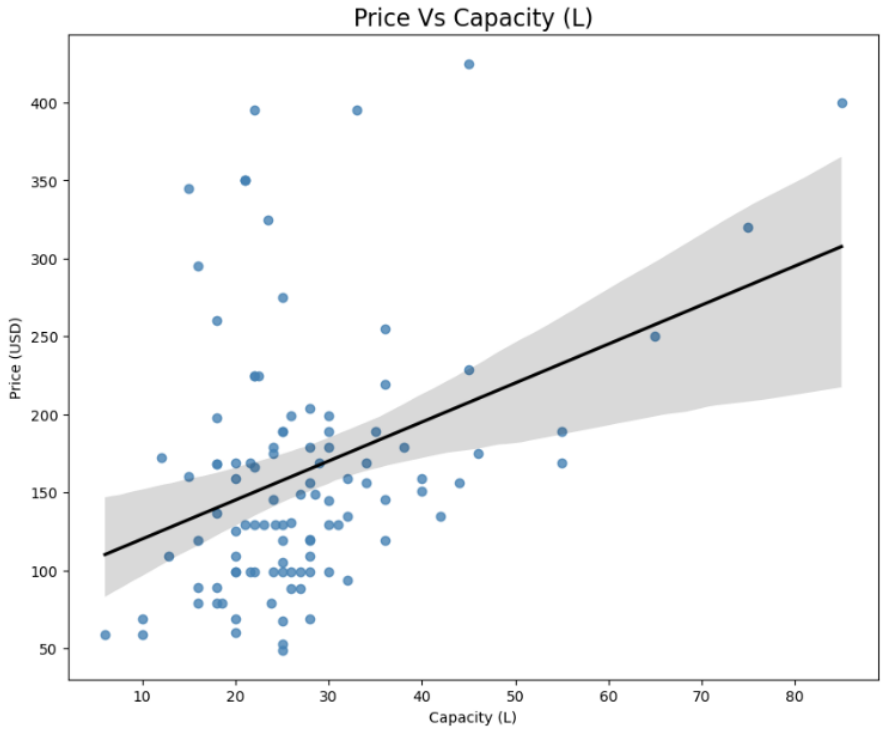
Exploratory Data Analysis

Exploratory Data Analysis (EDA) is an approach to analyze the data using visual techniques. It is used to discover trends, patterns, or to check assumptions with the help of statistical summary and graphical representations.

EDA is primarily used to see what data can reveal beyond the formal modelling or hypothesis testing task and provides a better understanding of data set variables and the relationships between them. It can also help determine if the statistical techniques being considered for data analysis are appropriate.

- **To judge if Capacity and Price of backpack are correlated:**

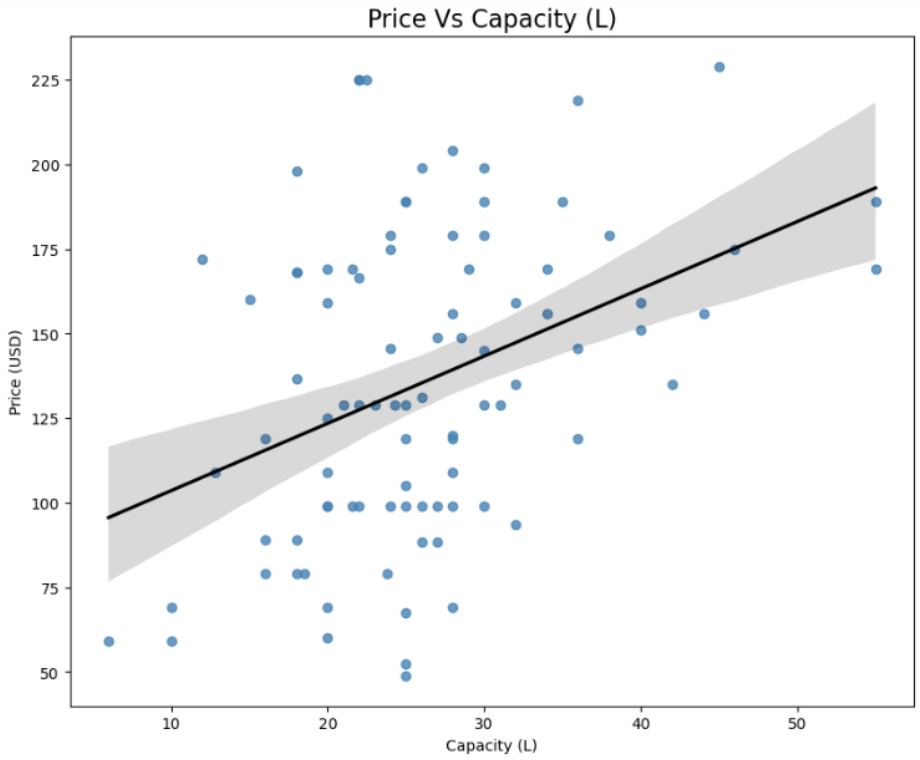
```
# plt price vs capcity
fig = plt.subplots(figsize = (10,8))
ax = sns.regplot(data = backpacks, x = 'Capacity (L)', y = 'Price (USD)', color = 'steelblue', line_kws = {'color' : 'black'})
ax.set_title('Price Vs Capacity (L)', fontsize = 16)

plt.show()
print('Pearson r = ', backpacks[['Price (USD)', 'Capacity (L)']].corr().iloc[0,1].round(2))
```




```
Pearson r =  0.36
```

A Pearson r score of 0.36 (less than 0.5) suggests that there is a weak positive correlation between price and capacity of the backpacks. Further exploration can be carried out into this relationship by limiting the price under 250$, since there are several data points in the 20 to 40 Litres range with high price.

```
#create a dataframe with only backpacks under 250$ only for study purposes
backpacks_under_250 = backpacks.query('`Price (USD)` < 250')

fig = plt.subplots(figsize = (10,8))
ax = sns.regplot(data = backpacks_under_250, x = 'Capacity (L)', y = 'Price (USD)', color = 'steelblue',
                 line_kws = {'color' : 'black'})
ax.set_title('Price Vs Capacity (L)', fontsize = 16)

plt.show()
print('Pearson r = ', backpacks_under_250[['Price (USD)', 'Capacity (L)']].corr().iloc[0,1].round(2))
```




```
Pearson r =  0.38
```

Since there is only a slight increase (0.36 to 0.38) in the Pearson r score, it can be concluded that there is only a weak correlation between price and capacity of backpacks.

- **To estimate which materials are most prevalent in the high-quality backpacks and if they determine their price:**
  Wordcloud library is used to find and visualize the most prevalent materials used in the manufacturing of backpacks. This is quite an appealing way of graphical representations of word frequency that give greater prominence to words that appear more frequently in a source text.

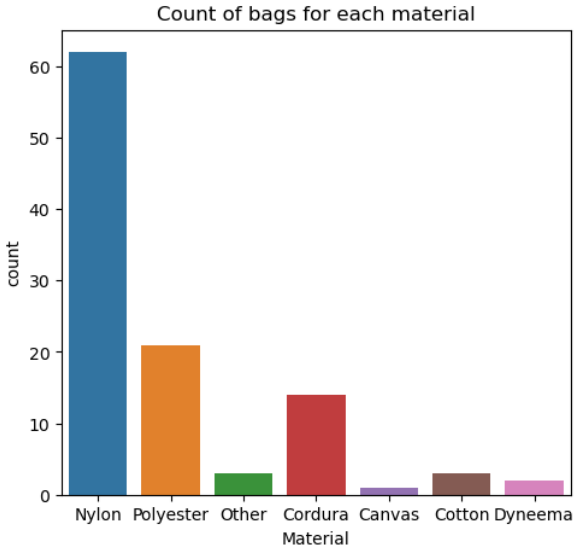
From the WordCloud, ‘Recycled Nylon’ and ‘Non-PFC’ are the most widely used materials for backpacks. The materials used becomes clear, however further study on the use of primary materials should be carried out for better understanding like 'cotton', 'cordura', 'dyneema', 'canvas', 'denier', 'nylon' and 'polyester'.

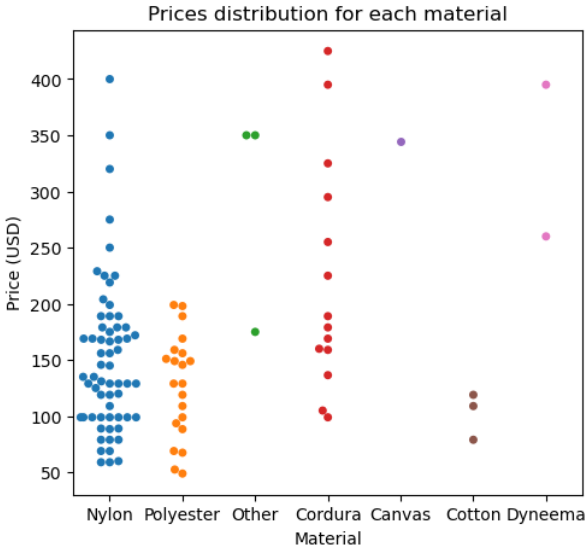
The following python snippet describes the function used to find primary material of the backpack by splitting the data in ‘Material’ feature vector.

```
#iterate the primary materials found
materials = ['cotton', 'cordura', 'dyneema', 'canvas', 'denier', 'nylon', 'polyester']

#create function to find primary material, we will assume that the first material in the description is the primary material
def find_primary_materials(entry):
    #initialize empty list
    materials_used = []
    #change string to lower case, then turn into list
    entry_as_list = entry.lower().replace('/',' ').split()
    #we will iterate through the list to find the matches, list the first match
    for word in entry_as_list:
        #initialize alphanumeric
        pattern = r'[^A-Za-z0-9]+'
        #remove non-alphanumeric characters in string
        word = re.sub(pattern, '', word)
        if word in materials:
            materials_used.append(word)
    #in case that the primary material is not included in our materials list
    if len(materials_used) == 0:
        materials_used = ['Other']
    #return our presumed primary material and capitalize for aesthetics
    return materials_used[0].capitalize()
```

```
#extract primary material of each backpack
backpacks['Material'] = backpacks['Material'].apply(lambda x: find_primary_materials(x))
```

After extracting the information of primary materials, count of bags as well as Price Distribution for each material is plotted.





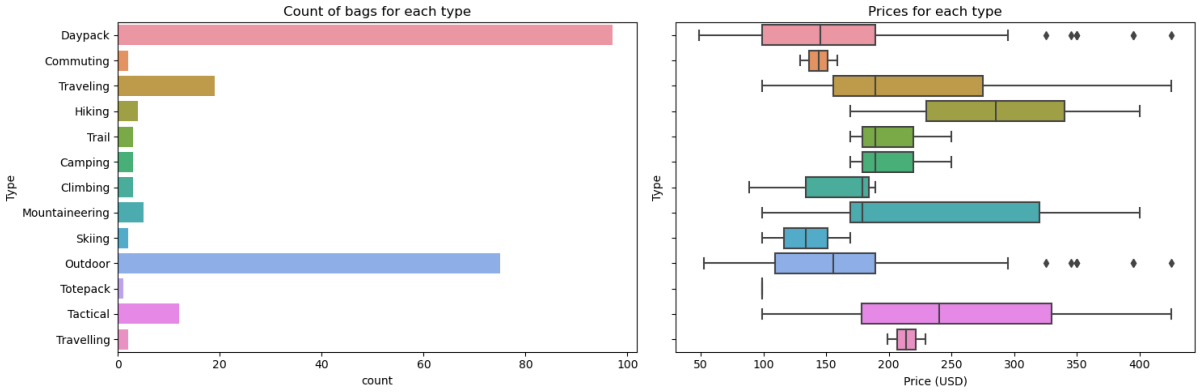
Observations for these plots is that the most common primary-material is Nylon as it can be seen that the maximum number of backpacks are made up of Nylon.
The prices for backpacks which uses nylon and polyester as primary materials seem to have the similar distribution (excluding outliers in Nylon). Cordura backpacks seem to have varying prices ranging from approx. 100$ to 450$. Most backpacks tend to have price in the range of 50$ to 200$. Thus it can be concluded that the materials used could determine the price of the backpacks.

But before that, it should be seen whether any combination type, brand and materials could give better insights.

Data in the 'Type' feature vector is split as per the following python snippet.

```
#reload exploded dataset to copy new Material values
backpacks_type_exploded = backpacks.copy()
backpacks_type_exploded['Type'] = backpacks_type_exploded.Type.str.split('/')
backpacks_type_exploded = backpacks_type_exploded.explode('Type')

#check dataframe
backpacks_type_exploded.head()
```

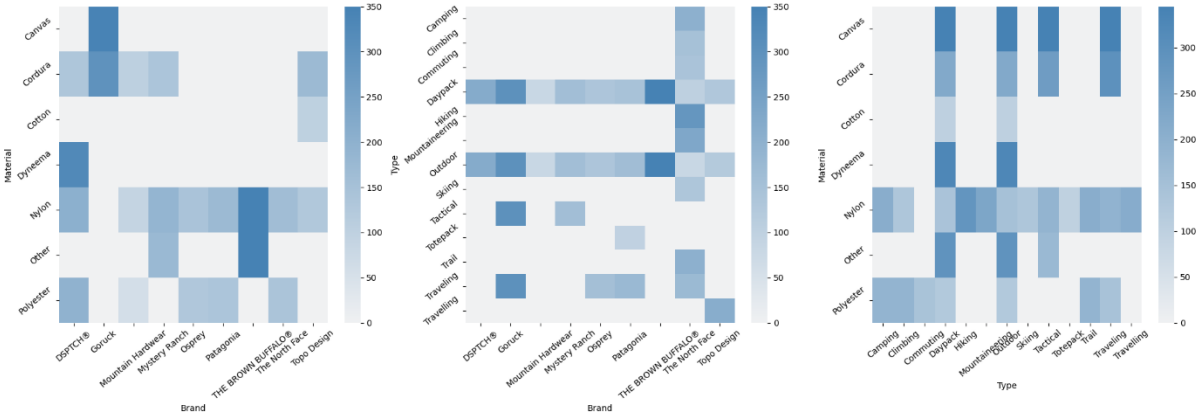
Countplot for the count of bags for each type is plotted along with a box plot of Price vs Type of bag.



It is observed that Daypacks and Outdoor bags are the most prevalent, followed by Travelling bags. The prices are mostly around the 100$ to 200$ range. The travelling, climbing, camping and mountaineering types are worth exploring since the price distribution is skewed. This could mean that there are a few bags of those types which costs more.
Also, the outdoor and daypacks produced some outliers. Those outliers could be the bags that are both outdoor and daypacks. A little research on those bags can be done to see why they are of that price.

Correlation heatmaps are plotted for 1) Brand vs Material, 2) Brand vs Type and 3) Type vs Material with respect to Price of the backpacks. Following python code is used for the same.

```
#set a light pallete for our heatmap
pallete = sns.light_palette('steelblue', as_cmap = True)

#define a function to create heatmaps from any columns on our data.
def create_heatmap(col1, col2, i):
    #create pivot table
    backpacks_pivot = pd.pivot_table(data = backpacks_type_exploded,
                                     columns = col1,
                                     index = col2,
                                     values = 'Price (USD)',
                                     #fill with 0 so it shows up as a white space on the map
                                     fill_value = 0,
                                    aggfunc = 'mean')
    #create  the map from the table
    ax[i] = sns.heatmap(backpacks_pivot, cmap = pallete, ax = ax[i])
    #rotate the axis for easier reading
    ax[i].set_yticklabels(ax[i].get_yticklabels(), rotation = 40)
    ax[i].set_xticklabels(ax[i].get_xticklabels(), rotation = 40)
    #return axis for plotting on the figure
    return ax

#create figure
fig, ax = plt.subplots(1,3, figsize = (20,7), tight_layout = True)

#fill in axes
create_heatmap('Brand', 'Material', 0)
create_heatmap('Brand', 'Type', 1)
create_heatmap('Type', 'Material', 2)

plt.show()
```



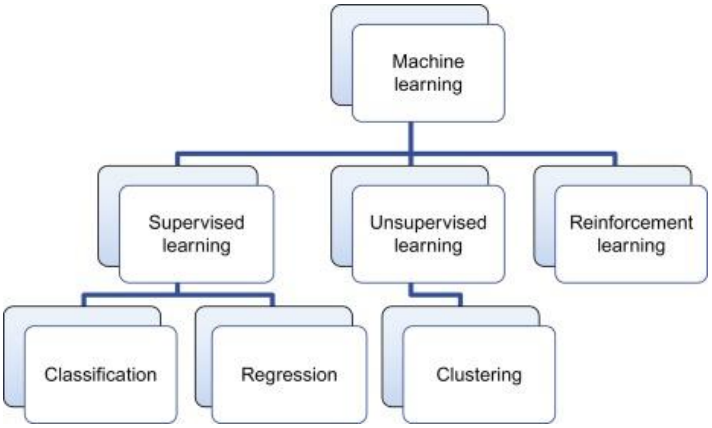
The variance in colors of the heatmap is used going across either vertically or horizontally to draw conclusions, since both axis are made of categorical variables.

Notice that the heatmaps are filled with white color. This means that those bags do not exist on the dataset provided.
When comparing using the material and brand, we can see that most brands tend to have the same price across different materials. Thus it can be said that the brand helps determine the price more.
While comparing the type and brand, same conclusion can be obtained that brand determines the prices more.
While comparing the prices through type and material, the prices tend to stay consistent through the materials, but have some variance with respect to the type of backpacks. Thus it can be concluded that the Type helps determine the price more.

Final Observation:
The bags capacity does have a weak correlation with prices, so bigger bags should not always be more expensive.
Categorically, it can be determine that the price is based on this order: brand, type, material. But this conclusion cannot be sure as the pivot table and heatmap is sparse.

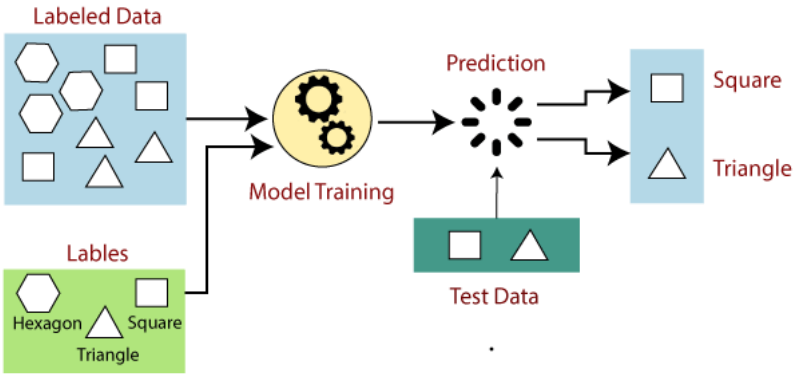
Machine Learning Activity:

Machine learning is a field of computer science and artificial intelligence (AI) that involves developing algorithms that can learn from data and make predictions or decisions based on the patterns they observe in the data.

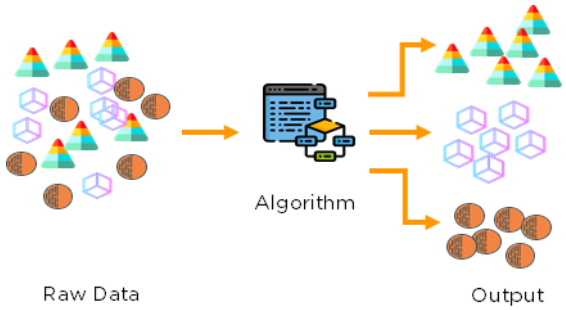


Machine learning is divided into three broad categories: supervised learning, unsupervised learning, and reinforcement learning.

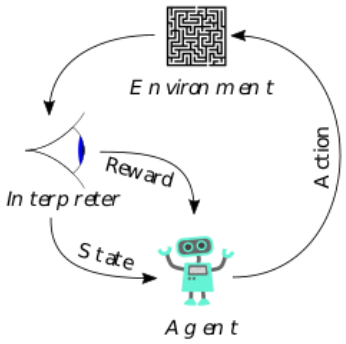
Supervised learning involves training an algorithm on a labelled dataset, where the input data points are associated with known output values. The goal of the algorithm is to learn a mapping between the input and output variables, so that it can predict the output values for new, unseen input data points. This shall be utilized for the problem statement provided.



Unsupervised learning, on the other hand, involves training an algorithm on an unlabelled dataset, where the input data points are not associated with any known output values. The goal of the algorithm is to discover patterns or relationships in the data that can be used for clustering or dimensionality reduction.

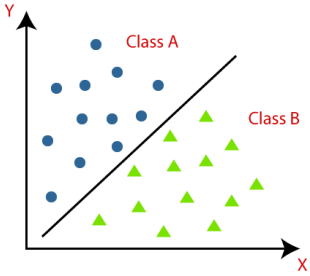


Reinforcement learning is a type of machine learning that involves training an agent to make a sequence of decisions in an environment, in order to maximize a cumulative reward. The agent interacts with the environment, receiving feedback in the form of rewards or penalties based on its actions.

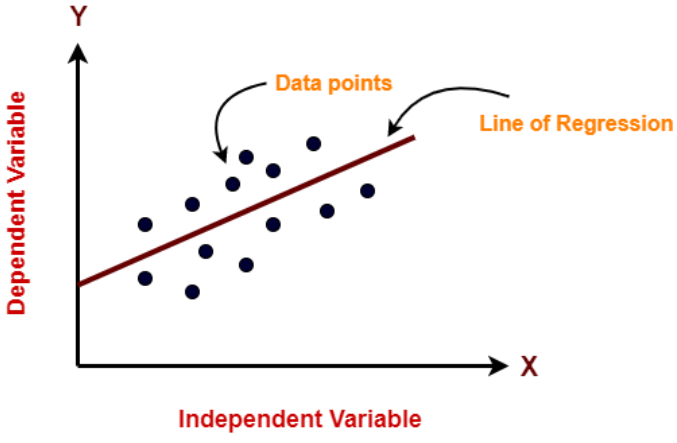


Supervised Learning:

1. **Classification:**
   Classification is the process of predicting the class of given data points. Classes are sometimes called targets, labels or categories. Classification predictive modelling is the task of approximating a mapping function (f) from input variables (X) to discrete output variables (y.). Some of the famous classification algorithms are Decision Tree, Support Vector Machine, Naïve Bayes, K-Nearest Neighbours, Artificial Neural Network.



2. **Regression:**
   Regression is a type of supervised learning in machine learning, where the algorithm is trained to predict a continuous numerical value, also known as a target variable. The input data points consist of one or more features, which are attributes or characteristics that describe the data point. Some of the famous regression algorithms are Linear Regression, Polynomial regression, KNN, Stochastic, Gradient Descent.



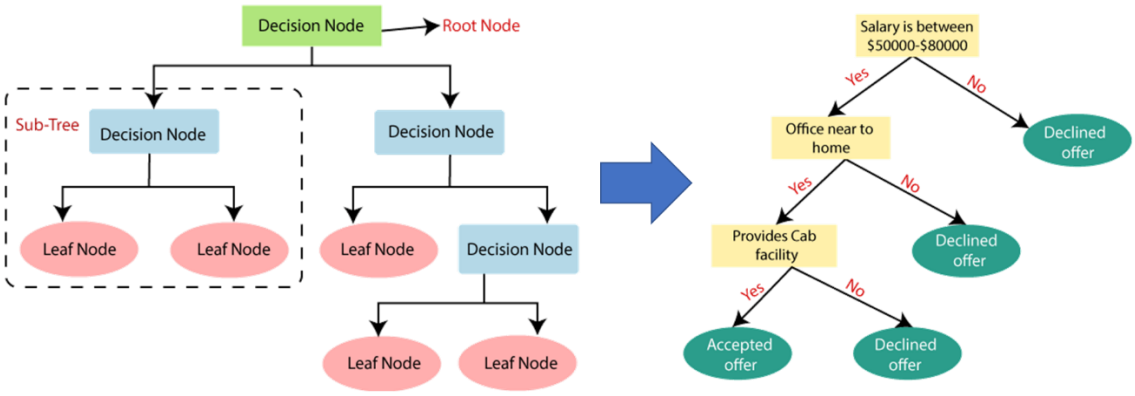
Classification:

For the given problem dataset, Decision Tree is used to classify the price category of backpacks.

Decision Tree:

Decision tree is a popular machine learning algorithm used for classification tasks. It is a tree-like model where each internal node represents a feature or attribute, each branch represents a decision rule, and each leaf node represents a class label or a predicted value.

The algorithm works by recursively splitting the data based on the most informative feature or attribute, which maximizes the separation between the classes or minimizes the variance in the target variable. The split is done based on a criterion, such as entropy or Gini impurity, which measures the homogeneity of the classes or the variance of the target variable.



Decision Tree Structure **Decision Tree Example**

Once the tree is constructed, the decision rules can be used to classify new data points or predict new target values by traversing the tree from the root node to the appropriate leaf node.

Feature Engineering:

Feature engineering is the process of transforming raw data into meaningful and informative features that can be used by machine learning algorithms to improve their performance. It involves selecting, creating, and transforming features from the original data in order to improve the quality and relevance of the input data.

1. **Feature Extraction from 'Type' Column:**
   'Type' column holds 1 or more values in a single cell. For example: Climbing/Trail/Camping
   Such values are separated using split function and placed in newly created columns – Type, Type1, Type2 and Type3. This allows the algorithm to not lose any information.
   For values with less than 4 types, empty cells are filled with the first type value.
   This is needed to enable proper functioning of label encoding algorithm which is explained in the following subsections.

```
# Split 'Type' column data with respect to '/'
temp_ = pd.DataFrame(df['Type'].str.split('/', expand=True))
for i in range(3):
    temp_[i+1] = temp_[i+1].fillna(temp_[0])
    df['Type'+str(i+1)] = temp_[i+1]
df['Type'] = temp_[0]
```

```
df
```

	Brand	Type	Capacity (L)	Material	Price (USD)	Type1	Type2	Type3
0	The North Face	Daypack	31.0	Nylon	129.00	Daypack	Daypack	Daypack
1	The North Face	Daypack	28.0	Nylon	99.00	Daypack	Daypack	Daypack
2	The North Face	Daypack	40.0	Nylon	159.00	Daypack	Daypack	Daypack
3	The North Face	Commuting	23.0	Polyester	129.00	Daypack	Commuting	Commuting
4	The North Face	Daypack	28.0	Polyester	69.00	Daypack	Daypack	Daypack
...	...	...	...	...	...	...	...	...
102	Osprey	Daypack	44.0	Polyester	156.00	Outdoor	Traveling	Daypack
103	Osprey	Daypack	36.0	Polyester	145.68	Outdoor	Traveling	Daypack
104	Osprey	Daypack	46.0	Nylon	175.00	Outdoor	Traveling	Daypack
105	Osprey	Daypack	28.0	Nylon	156.00	Outdoor	Traveling	Daypack
106	Osprey	Daypack	40.0	Polyester	151.00	Outdoor	Traveling	Daypack

106 rows × 8 columns

2. One-Hot Encoding:

One-hot encoding is a technique used in machine learning to represent categorical data as numerical data. It involves creating a binary vector where each element represents a unique category in the dataset. The vector has a length equal to the number of unique categories, and each element is either a 0 or 1, indicating the presence or absence of the corresponding category. One-hot encoding ensures that no implicit ordering or ranking is imposed on the categories, which could lead to misleading results. However, it can create high-dimensional data, especially when there are many unique categories in the dataset. Features converted into columns with binary values are – Type, Type1, Type2, Type3, Brand and Material

```
# One Hot Encoding to converts categorical data into a numeric form,
# so as to convert them into the machine-readable form

one_hot_encoded_data = pd.get_dummies(df, columns=['Type','Type1','Type2','Type3','Brand','Material'])
df = one_hot_encoded_data.copy()
```

```
df.head()
```

	Capacity (L)	Price (USD)	Type_Climbing	Type_Commuting	Type_Daypack	Type_Hiking	Type_Skiing	Type_Traveling	Type1_Climbing	Type1_Daypack	...	Brand_THE BROWN BUFFALO
0	31.0	129.0	0	0	1	0	0	0	0	1	...	0
1	28.0	99.0	0	0	1	0	0	0	0	1	...	0
2	40.0	159.0	0	0	1	0	0	0	0	1	...	0
3	23.0	129.0	0	1	0	0	0	0	0	1	...	0
4	28.0	69.0	0	0	1	0	0	0	0	1	...	0

5 rows × 49 columns

3. Label Encoding:

Label Encoding refers to converting the labels into a numeric form so as to convert them into the machine-readable form. In label encoding, each unique value of a categorical feature is assigned a unique integer label.

'Price (USD)' is converted to bins using pd.cut function. Pandas cut() function is used to separate the array elements into different bins . The cut function is mainly used to perform statistical analysis on scalar data. This is then converted to categorical format using Label Encoder.

```
n_bins = i+2
y = label_encoder.fit_transform(pd.cut(df['Price (USD)'], n_bins, retbins=True)[0])
```

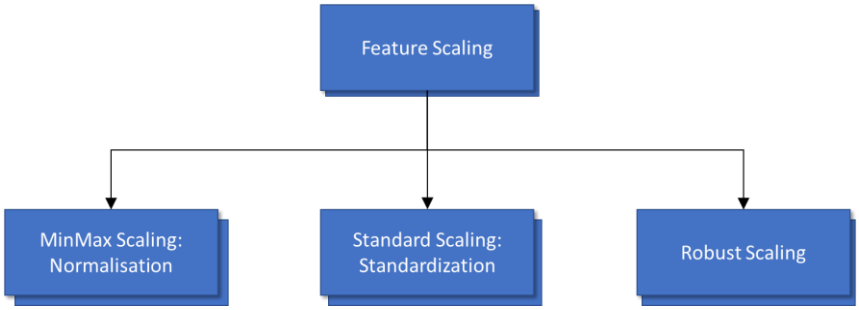
4. **Type Casting:**
   Type casting is when you assign a value of one primitive data type to another type. Price is converted to float data type explicitly for use in machine learning algorithm development.

```
#converting brand, type, material to integer and price to float
df[['Brand','Type','Type1','Type2','Type3','Material']] = df[['Brand','Type','Type1','Type2','Type3','Material']].astype(int)
df[['Price (USD)']] = df[['Price (USD)']].astype(float)
```

```
df.isna().sum()
```

```
Brand           0
Type            0
Capacity (L)    0
Material        0
Price (USD)     0
dtype: int64
```

5. **Feature Scaling:**
   Feature scaling is a method used to normalize/standardize the range of independent variables or features of data.



For the classification problem, three different scaling functions are tried and tested.

1. **MinMaxScaler**(): The MinMaxscaler is a type of scaler that scales the minimum and maximum values to be 0 and 1 respectively.
   scaled_value = (value - min) / (max - min)

2. **StandardScaler**(): Standard scaling normalization, also known as Z-score normalization, is a common data normalization technique used in machine learning. It involves transforming the data so that it has a mean of 0 and a stndard deviation of 1.
   scaled_value = (value - mean) / standard_deviation

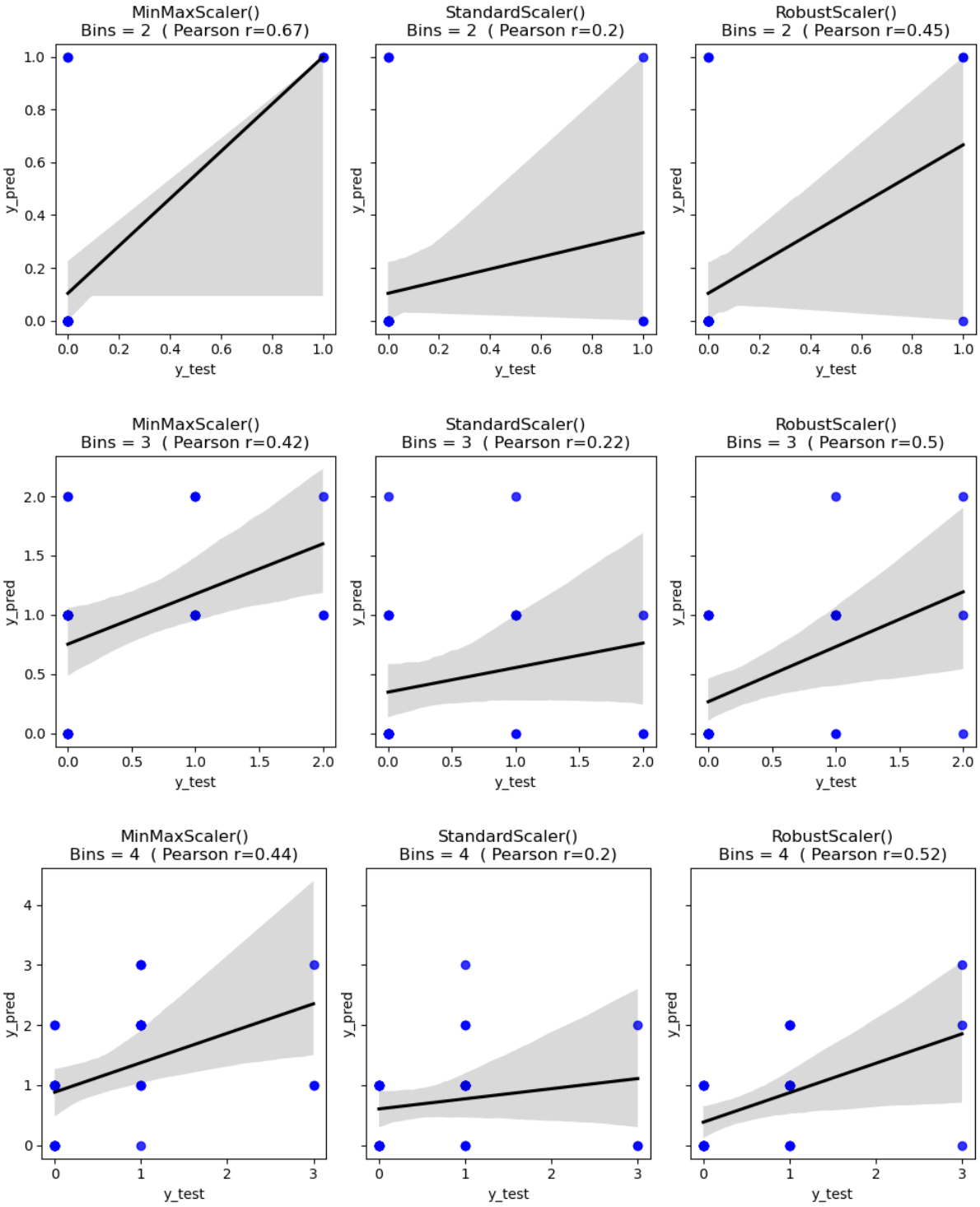
3. **RobustScaler**():Robust scaling normalization is a type of data normalization technique used in machine learning that is designed to be robust to outliers. It involves scaling the data so that it is centered around the median and has a similar interquartile range (IQR).
   scaled_value = (value - median) / IQR

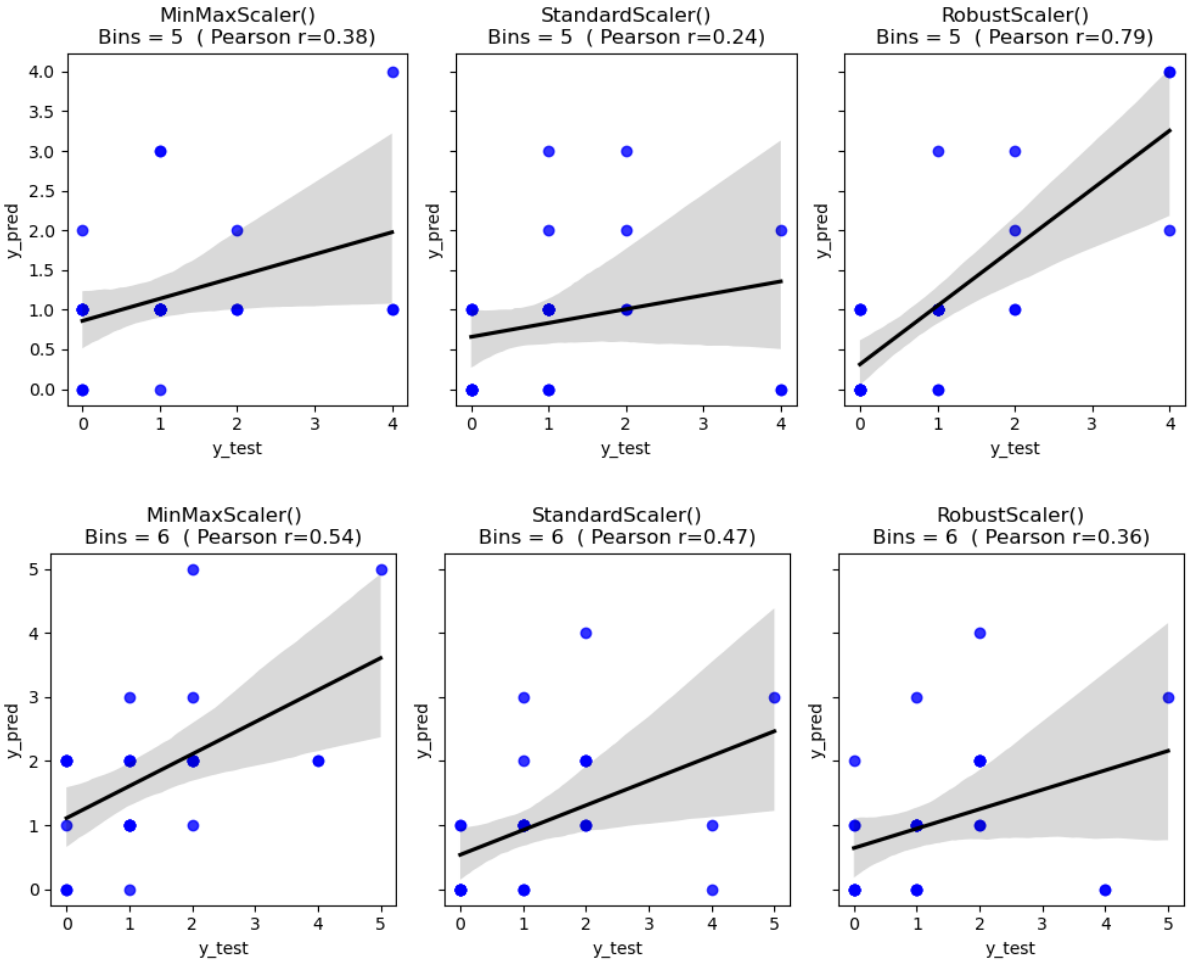
   Out of the 3 scalers, robustscaler showed a higher accuracy than the other 2 scalers since it is robust to the presence of outliers. Observations are shown in the following context.

```
print('\n------------------------ Classification ------------------------')
print('Breaking down continuous values of PRICE to following bins and checking classification accuracy')

# For loop to compare performance of ML Model with Bins=2 to Bins=6
for i in range(5):

    # Label encoding and Binning of price data for conersion of continuos to categorical data
    label_encoder = LabelEncoder()
    n_bins = i+2
    y = label_encoder.fit_transform(pd.cut(df['Price (USD)'], n_bins, retbins=True)[0])
    df_bins = pd.DataFrame()

    df_bins['bins'] = (pd.cut(df['Price (USD)'], n_bins,retbins=False))
    df_bins['bins'] = df_bins['bins'].astype(str)
    bins_ = df_bins['bins'].unique()[0:n_bins]
    print('\n\nWith Bins =',n_bins,'Bin Ranges:', bins_)

    X = df[cols_].values

    # 70% training dataset and 30% test dataset
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=1)

    # Use of 3 scalers for performance comparison
```

```
    # Use of 3 scalers for performance comparison
    sc_s = [MinMaxScaler(), StandardScaler(), RobustScaler()]
    fig, axes = plt.subplots(1, 3, sharey=True, figsize=(10, 4))
    i=0
    print('\t MinMaxScaler \t      StandardScaler \t RobustScaler')
    print('Pearson r = ',end="")
    for sc in sc_s:
        X_train_sc = sc.fit_transform(X_train)
        X_test_sc = sc.fit_transform(X_test)

        # Decision Tree Classifier is used with maximum depth = 10 (To avoid overfitting low number is used)
        # For node splitting - Gini Index is used as the metric
        DTC = DecisionTreeClassifier(max_depth=10, criterion='gini')
        clf = DTC.fit(X_train_sc,y_train)
        y_pred = clf.predict(X_test_sc)
        df_temp = pd.DataFrame()
        df_temp['y_test'] = y_test
        df_temp['y_pred'] = y_pred
        ax = axes[i]
        sns.regplot(ax=ax, data=df_temp, x='y_test', y='y_pred', color='blue', line_kws={'color': 'black'})
        ax.set_title(str(sc)+'\nBins = '+str(n_bins) +
                     '  ( Pearson r=' + str(df_temp[['y_test', 'y_pred']].corr().iloc[0,1].round(2)) + ')')

        # Use of Pearson r score metric to judge the performance of algorithm
        if(df_temp[['y_test', 'y_pred']].corr().iloc[0,1].astype(float).round(2) == np.nan):
            corr_ = 0.00
        else:
            corr_ = df_temp[['y_test', 'y_pred']].corr().iloc[0,1].astype(float)
        print('%.2f'%round(corr_,2),end="\t\t")
        plt.tight_layout()
        plt.draw()
        i += 1
```

```
------------------------ Classification ------------------------
Breaking down continuous values of PRICE to following bins and checking classification accuracy


With Bins = 2 Bin Ranges: ['(48.554, 236.965]' '(236.965, 425.0]']
         MinMaxScaler         StandardScaler     RobustScaler
Pearson r = 0.67                0.20            0.45

With Bins = 3 Bin Ranges: ['(48.554, 174.287]' '(174.287, 299.643]' '(299.643, 425.0]']
         MinMaxScaler         StandardScaler     RobustScaler
Pearson r = 0.42                0.22            0.50

With Bins = 4 Bin Ranges: ['(48.554, 142.948]' '(142.948, 236.965]' '(236.965, 330.982]'
 '(330.982, 425.0]']
         MinMaxScaler         StandardScaler     RobustScaler
Pearson r = 0.44                0.20            0.52

With Bins = 5 Bin Ranges: ['(124.144, 199.358]' '(48.554, 124.144]' '(199.358, 274.572]'
 '(274.572, 349.786]' '(349.786, 425.0]']
         MinMaxScaler         StandardScaler     RobustScaler
Pearson r = 0.38                0.24            0.79

With Bins = 6 Bin Ranges: ['(111.608, 174.287]' '(48.554, 111.608]' '(174.287, 236.965]'
 '(236.965, 299.643]' '(299.643, 362.322]' '(362.322, 425.0]']
         MinMaxScaler         StandardScaler     RobustScaler
Pearson r = 0.54                0.47            0.36
```


MinMaxScaler()
Bins = 2 (Pearson r=0.67)
StandardScaler()
Bins = 2 (Pearson r=0.2)
RobustScaler()
Bins = 2 (Pearson r=0.45)
MinMaxScaler()
Bins = 3 (Pearson r=0.42)
StandardScaler()
Bins = 3 (Pearson r=0.22)
RobustScaler()
Bins = 3 (Pearson r=0.5)
MinMaxScaler()
Bins = 4 (Pearson r=0.44)
StandardScaler()
Bins = 4 (Pearson r=0.2)
RobustScaler()
Bins = 4 (Pearson r=0.52)
y_pred
y_test

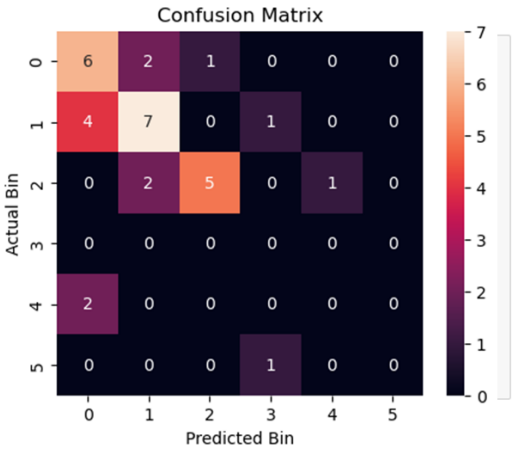


It is observed that with bins = 5, RobustScaler shows the best performance with Pearson r score = 0.79 which shows a significant and positive relationship exists between the input feature vectors and output – Price of Backpacks.

Generally, for classification algorithms, Confusion Matrix is used to judge the accuracy/performance of the algorithm so developed. However, in this case Pearson's r score and regression plots are a better metric even if prices are binned for categorical distribution.

This is because if the ML algorithm predicts that price in the range defined by *Bin-2* when the actual price was in the range of *Bin-3,* which is the adjacent bin*,* then Pearson's r score gives it better value than simply saying wrongly predicted as per confusion matrix.

```
# Building a Confusion Matrix for Classification model on test dataset

from sklearn.metrics import confusion_matrix

cm = confusion_matrix(y_test, y_pred)

cm_df = pd.DataFrame(cm,
                     index = [i for i in range(6)],
                     columns = [i for i in range(6)])

#Plotting the confusion matrix
plt.figure(figsize=(5,4))
sns.heatmap(cm_df, annot=True)
plt.title('Confusion Matrix')
plt.ylabel('Actual Bin')
plt.xlabel('Predicted Bin')
plt.show()
```



Regression:

For the given problem dataset, 3 different regression approaches are explored and compared in terms accuracy of prediction using regression metrics to predict the price of backpacks.

Regression algorithms checks:

1. **Stochastic Gradient Descent Model:**
   It is a variant of gradient descent optimization that iteratively updates the model parameters to minimize the cost function. Unlike Gradient descent algorithm, SGD regression updates the model parameters using only a randomly selected subset (batch) of the training data at each iteration. This makes the algorithm more efficient and scalable for large datasets.

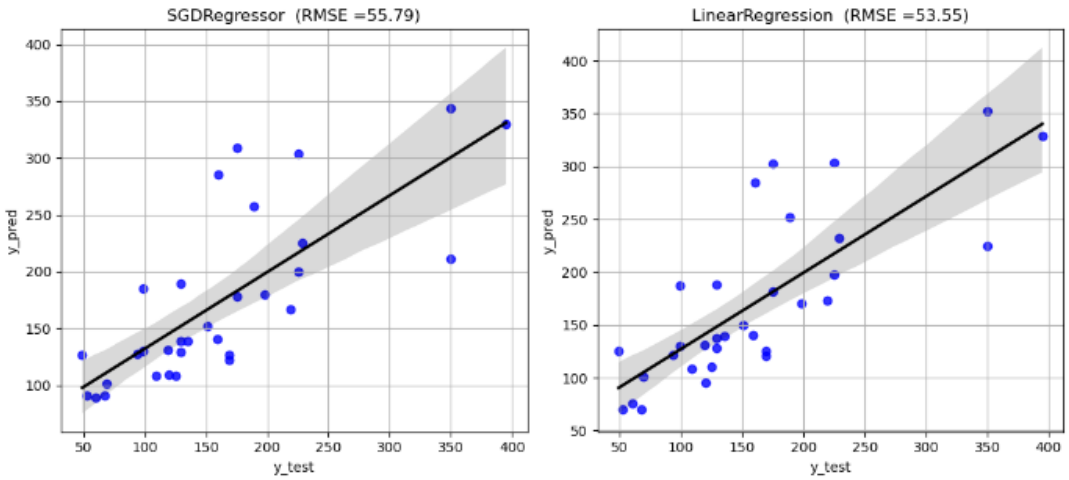
2. **Linear Regression Model:**
   The line of best fit is represented by a linear equation of the form y = b0 + b1x1 + b2x2 + … + bn*xn, where y is the dependent variable, b0 is the intercept, b1, b2, …, bn are the coefficients or weights of the independent variables x1, x2, …, xn, respectively.

3. **Linear Regression with Lasso (L2) Regularization:**
   It is a variant of linear regression that penalizes large coefficients to prevent overfitting. The Lasso regularization technique adds a penalty term to the objective function of linear regression that is proportional to the sum of the absolute values of the model coefficients.

```
print('\n------------------------ Regression -------------------------')
X = df[cols_].values
y = df['Price (USD)'].values

# Random splitting the entire data into Training Dataset (70%) and Test Dataset (30%)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=1)

# Scaling the data with RobustScaler() which was last used in classification model building activity
X_train_sc = sc.fit_transform(X_train)
X_test_sc = sc.fit_transform(X_test)

# Function to Regression line plot using seaborn library
def sns_plot(y_test,y_pred,ax_,rmse_,title_):
    df_temp = pd.DataFrame()
    df_temp['y_test'] = y_test
    df_temp['y_pred'] = y_pred
    ax = ax_
    ax.grid()
    sns.regplot(ax=ax, data=df_temp, x='y_test', y='y_pred', color='blue', line_kws={'color': 'black'})
    ax.set_title(title_+'  (RMSE =' + rmse_+ ')')
    plt.tight_layout()
    plt.draw()

fig, axes = plt.subplots(1, 2, sharey=False, figsize=(11,5))

# Stochastic Gradient Descent Regression Model
# Max_Itr increased to avoid convergence error
clf_ = SGDRegressor(max_iter=5000)
clf_.fit(X_train_sc, y_train)
y_pred=clf_.predict(X_test_sc)

df_temp = pd.DataFrame()
df_temp['y_test'] = y_test
df_temp['y_pred'] = y_pred
corr_ = df_temp[['y_test', 'y_pred']].corr().iloc[0,1].astype(float)
```

```
rmse_ = str('{:0.2f}'.format(round(np.sqrt(mean_squared_error(y_test, y_pred)),2)))
ax_ = axes[0]
sns_plot(y_test,y_pred,ax_,rmse_,title_ = 'SGDRegressor')
print('\nUsing Stochastic Gradient Descent Model : RMSE = ',rmse_)
print('Using Stochastic Gradient Descent Model : Pearsons r = ',round(corr_,2))

# Linear regression model - defined in Sci-kit_learn python library
reg = linear_model.LinearRegression()
reg.fit(X_train_sc, y_train)
y_pred = reg.predict(X_test_sc)

df_temp = pd.DataFrame()
df_temp['y_test'] = y_test
df_temp['y_pred'] = y_pred

# Pearson's r score metric for performance check
corr_ = df_temp[['y_test', 'y_pred']].corr().iloc[0,1].astype(float)

# Root Mean Square Error calculation
mse_ = str('{:0.2f}'.format(round(np.sqrt(mean_squared_error(y_test, y_pred)),2)))
ax_ = axes[1]
sns_plot(y_test,y_pred,ax_, mse_,title_ = 'LinearRegression')
print('\nUsing LinearRegression Model : RMSE = ',round(np.sqrt(mean_squared_error(y_test, y_pred)),2))
print('Using Stochastic Gradient Descent Model : Pearson r = ',round(corr_,2))
```

```
------------------------ Regression -------------------------

Using Stochastic Gradient Descent Model : RMSE =  55.79
Using Stochastic Gradient Descent Model : Pearsons r =  0.76

Using LinearRegression Model : RMSE =  53.55
Using Stochastic Gradient Descent Model : Pearson r =  0.79
```




```
# Linear Regression using Linear Regression with Lasso (L2) Regularization

print('\nUsing Linear Regression with Lasso (L2) Regularization:')
rmse_scores = []     # List to maintain the cross-validation scores
Lambda = []     # List to maintain the different values of Lambda
for i in range(1, 9):
    lassoModel = Lasso(alpha=i * 0.25, tol=0.0925)
    lassoModel.fit(X_train_sc, y_train)
    y_pred = lassoModel.predict(X_test_sc)

    df_temp = pd.DataFrame()
    df_temp['y_test'] = y_test
    df_temp['y_pred'] = y_pred

    # Pearson's r score and RMSE performance metric
    corr_ = df_temp[['y_test', 'y_pred']].corr().iloc[0,1].astype(float)

    rmse_scores.append(np.sqrt(mean_squared_error(y_test, y_pred)).round(2))
    Lambda.append(i * 0.25)
    print('Lamba = ',str(Lambda[i-1]) + ' gives MSE = ' + str(rmse_scores[i-1]),'Pearson r = ',round(corr_,2))

min_rmse_pos = rmse_scores.index(min(rmse_scores))
print('Minimum RMSE =',rmse_scores[min_rmse_pos],'obtained for Lambda =',Lambda[min_rmse_pos])
```

```
Using Linear Regression with Lasso (L2) Regularization:
Lamba =  0.25 gives MSE = 51.1 Pearson r =  0.82
Lamba =  0.5 gives MSE = 52.61 Pearson r =  0.8
Lamba =  0.75 gives MSE = 53.81 Pearson r =  0.79
Lamba =  1.0 gives MSE = 54.61 Pearson r =  0.78
Lamba =  1.25 gives MSE = 55.78 Pearson r =  0.76
Lamba =  1.5 gives MSE = 57.46 Pearson r =  0.74
Lamba =  1.75 gives MSE = 59.67 Pearson r =  0.71
Lamba =  2.0 gives MSE = 62.3 Pearson r =  0.67
Minimum RMSE = 51.1 obtained for Lambda = 0.25
```

For regression, we tried and tested 3 ML models - Linear Regression, Stochastic Gradient Regression (SGD) and Linear Regression with Lasso (L2) regularization. Metric used to compare the accuracy of the ML models is RMSE (Root Mean Square Error) and Pearson's r Score.

For the problem dataset, minimum RMSE of 51.1 is obtained for Linear Regression with Lasso (L2) regularization with Lambda = 0.25. Thus it is the best for this dataset.

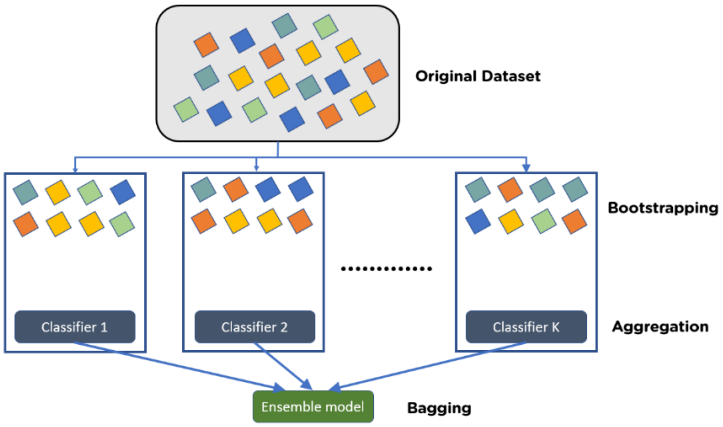
Both Classification and Regression show nearly similar performance with Classification having 0.79 and regression having 0.82 maximum Pearson’s r score value.

Ensemble:

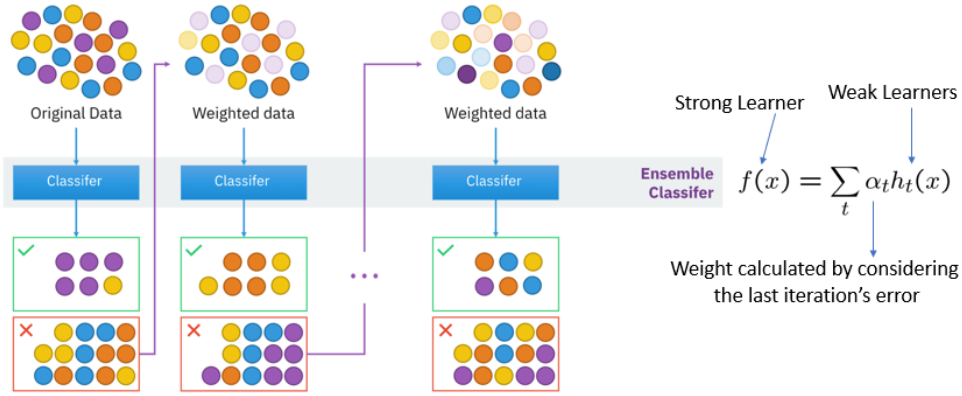
Ensemble learning is a technique in machine learning that involves combining multiple models to improve the accuracy and robustness of predictions. The idea is that by combining the predictions of multiple models, the resulting ensemble model will be more accurate and less prone to overfitting than any individual model.

There are several types of ensemble learning methods, including:

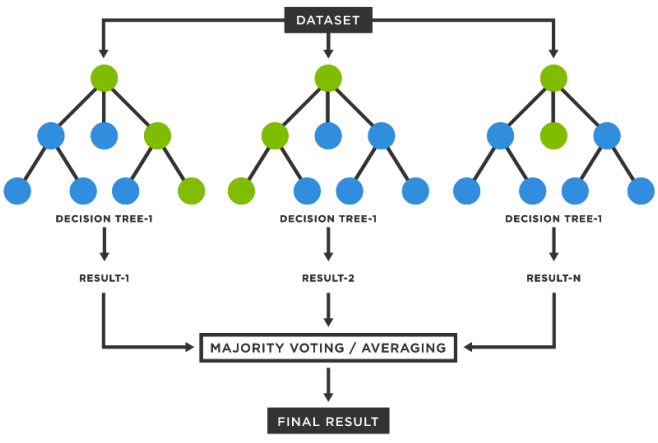
Bagging: This involves creating multiple models using bootstrap samples of the training data and then averaging their predictions.



Boosting: This involves creating multiple models sequentially, with each subsequent model focusing on the samples that the previous model misclassified.



Random Forest: This is a specific type of ensemble learning that involves creating multiple decision trees and then averaging their predictions.



Ensemble learning is widely used in machine learning applications and has been shown to be effective in improving accuracy and reducing overfitting. However, it can be computationally expensive and may require more data to train the multiple models involved.

For the dataset provided, Random Forest is chosen as the ensemble algorithm to be explored for prediction of price of backpacks.

Random forest: It creates multiple decision trees by using bootstrap samples of the training data and randomly selecting a subset of features at each split. Each decision tree in the random forest is trained independently, using a different subset of the training data and features. When making predictions, the random forest combines the predictions of all the decision trees to arrive at a final prediction.

Random forest is effective in reducing overfitting because it uses an ensemble of decision trees rather than a single tree. Random forest can handle both classification and regression tasks. Also, its adoption in the industry has gained great momentum due to its ease of use and flexibility.

Feature Engineering:
Robust scaling is only used for this algorithm as it shows best performance which was confirmed during Classification algorithm development.

Similar to classification algorithm, binning of price is used to convert continuous data into categorical values.

Random Forest Parameters (Attributes):

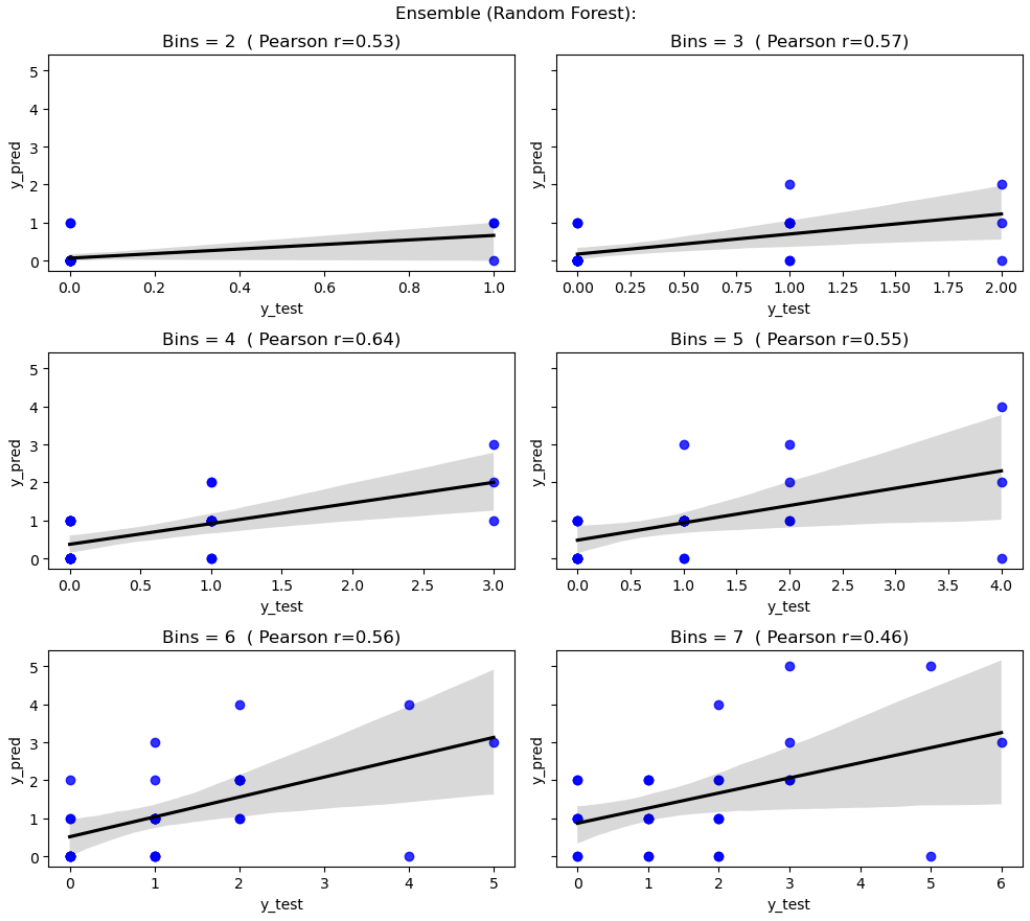
1. n_estimators (default=100) → The number of trees in the forest.
2. Criterion {“gini”, “entropy”, “log_loss”}, default=”gini” → The function to measure the quality of a split.
3. max_depth → The maximum depth of the tree. If None, then nodes are expanded until all leaves are pure or until all leaves contain less than min_samples_split samples.
4. random_state → Controls both the randomness of the bootstrapping of the samples used when building trees

and so on…

```
# Ensemble ML Model - Random Forest Classifier

print('\n------------------------- Ensemble (Random Forest Classifier) -------------------------')
print('Breaking down continuous values of PRICE to following bins and checking classification accuracy')
fig, axes = plt.subplots(3, 2, sharey=True, figsize=(10,9))
fig.suptitle('Ensemble (Random Forest):')
for i in range(6):

    # Label Encoding and binning to convert numeric contiuous data to categorical data for classification
    label_encoder = LabelEncoder()
    n_bins = i+2

    y = label_encoder.fit_transform(pd.cut(df['Price (USD)'], n_bins, retbins=True)[0])
    df_bins = pd.DataFrame()
    df_bins['bins'] = (pd.cut(df['Price (USD)'], n_bins,retbins=False))
    df_bins['bins'] = df_bins['bins'].astype(str)

    X = df[cols_].values

    # Random Splitting entire dataset into Training (70%) and Test (30%) Dataset
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=1)

    # Scaling the traing and Test X data using Robust scaler (Last used in Regression)
    X_train_sc = sc.fit_transform(X_train)
    X_test_sc = sc.fit_transform(X_test)

    # Random Forest Classifier with Gini index metric
    rfc = RandomForestClassifier(verbose=0, criterion = 'gini')
    clf = rfc.fit(X_train_sc,y_train)
    y_pred = clf.predict(X_test_sc)
```

```
    df_temp = pd.DataFrame()
    df_temp['y_test'] = y_test
    df_temp['y_pred'] = y_pred

    # Axis definition as per plot number - row and column location
    ax = axes[i // 2, i%2]
    sns.regplot(ax=ax, data=df_temp, x='y_test', y='y_pred', color='blue', line_kws={'color': 'black'})

    ax.set_title('Bins = '+str(n_bins) + '  ( Pearson r=' + str(df_temp[['y_test', 'y_pred']].corr().iloc[0,1].round(2)) + ')')

    # Pearson's r score metric
    print('\nBins =',n_bins, ", Pearson r =",df_temp[['y_test', 'y_pred']].corr().iloc[0,1].round(2))
    print('Categories:', df_bins['bins'].unique()[0:n_bins])
plt.tight_layout()
plt.draw()
```

```
------------------------- Ensemble (Random Forest Classifier) -------------------------
Breaking down continuous values of PRICE to following bins and checking classification accuracy

Bins = 2 , Pearson r = 0.53
Categories: ['(48.554, 236.965]' '(236.965, 425.0]']

Bins = 3 , Pearson r = 0.57
Categories: ['(48.554, 174.287]' '(174.287, 299.643]' '(299.643, 425.0]']

Bins = 4 , Pearson r = 0.64
Categories: ['(48.554, 142.948]' '(142.948, 236.965]' '(236.965, 330.982]'
 '(330.982, 425.0]']

Bins = 5 , Pearson r = 0.55
Categories: ['(124.144, 199.358]' '(48.554, 124.144]' '(199.358, 274.572]'
 '(274.572, 349.786]' '(349.786, 425.0]']

Bins = 6 , Pearson r = 0.56
Categories: ['(111.608, 174.287]' '(48.554, 111.608]' '(174.287, 236.965]'
 '(236.965, 299.643]' '(299.643, 362.322]' '(362.322, 425.0]']

Bins = 7 , Pearson r = 0.46
Categories: ['(102.654, 156.379]' '(48.554, 102.654]' '(156.379, 210.103]'
 '(210.103, 263.827]' '(317.551, 371.276]' '(371.276, 425.0]'
 '(263.827, 317.551]']
```

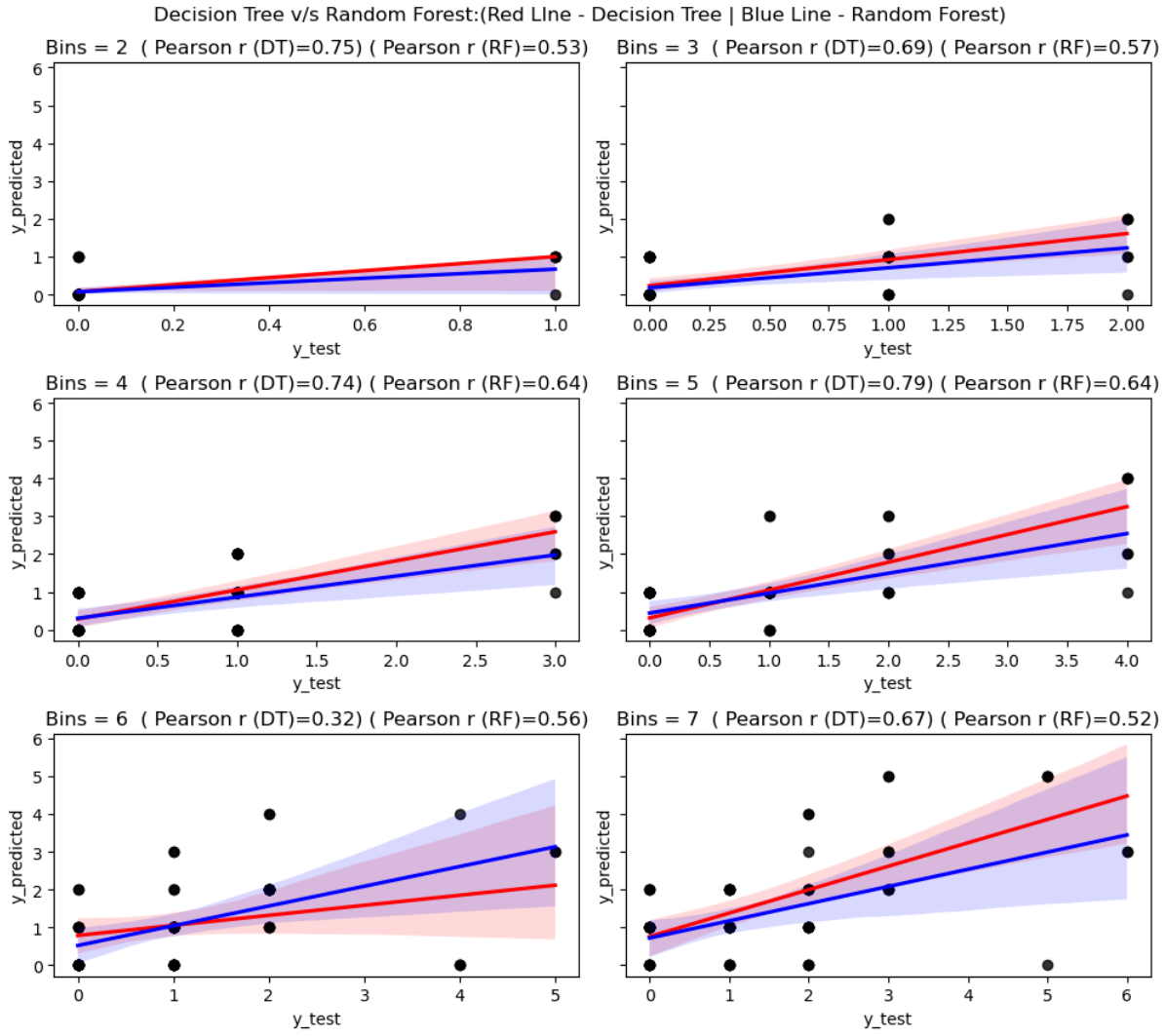


Maximum Pearson r score observed by using Random Forest algorithm is 0.64 with bins = 4. As was highlighted during classification algorithm development, Pearson's R score is a better metric than using confusion matrix with Accuracy and F1 Score.

A Pearson r score of 0.64 shows that the algorithm is of not a good quality and is not able to predict price to a good extent. Any value greater than 0.75 is considered a good score in terms of prediction accuracy. However, since Random Forest is an extension of Decision Tree algorithm, an extra step has been taken to have a one to one comparison of both with same Train and Test datasets. This is explored as below.

```
# Comparison of Random Forest with Decision Tree ML algorithms

print('\n------------------------- Decision Tree v/s Random Forest -------------------------')
print('Breaking down continuous values of PRICE to following bins and checking classification accuracy')
fig, axes = plt.subplots(3, 2, sharey=True, figsize=(10,9))
fig.suptitle('Decision Tree v/s Random Forest:(Red LIne - Decision Tree | Blue Line - Random Forest)')
for i in range(6):

    # Label Encoding and binning to convert numeric continuous data to categorical
    label_encoder = LabelEncoder()
    n_bins = i+2
    y = label_encoder.fit_transform(pd.cut(df['Price (USD)'], n_bins, retbins=True)[0])
    df_bins = pd.DataFrame()
    df_bins['bins'] = (pd.cut(df['Price (USD)'], n_bins,retbins=False))
    df_bins['bins'] = df_bins['bins'].astype(str)

    X = df[cols_].values

    # Random splitting entire dataset - 70% Training and 30% Test Datasets
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=1) # 70% training and 30% test

    # Robust scaling
    X_train_sc = sc.fit_transform(X_train)
    X_test_sc = sc.fit_transform(X_test)

    # Decision Tree Classifier
    DTC = DecisionTreeClassifier()
    clf = DTC.fit(X_train_sc,y_train)
    y_pred = clf.predict(X_test_sc)
    df_temp = pd.DataFrame()
    df_temp['y_test'] = y_test
    df_temp['y_pred1'] = y_pred
    ax = axes[i // 2, i % 2]
    sns.regplot(ax=ax, data=df_temp, x='y_test', y='y_pred1', color='black', line_kws={'color': 'red'})
```

```
    # Random Forest Classifier
    rfc = RandomForestClassifier(verbose=0,criterion='gini')
    clf = rfc.fit(X_train_sc,y_train)
    y_pred = clf.predict(X_test_sc)
    df_temp['y_test'] = y_test
    df_temp['y_predicted'] = y_pred
    sns.regplot(ax=ax, data=df_temp, x='y_test', y='y_predicted', color='black', line_kws={'color': 'blue'})

    ax.set_title('Bins = '+str(n_bins) + '  ( Pearson r (DT)=' +
                 str(df_temp[['y_test', 'y_pred1']].corr().iloc[0,1].round(2)) + ')'+
                 ' ( Pearson r (RF)=' +
                 str(df_temp[['y_test', 'y_predicted']].corr().iloc[0,1].round(2)) + ')')

    print('Bins = ',n_bins,' , Categories:', df_bins['bins'].unique()[0:n_bins])
plt.tight_layout()
plt.show()
```



It is observed that Decision Tree is showing higher value of Pearson **r** score than Random Forest. This may be because Random Forest gives majority voting / average output of multiple decision trees.

However, these numbers may change if the algorithm is rerun, since the entire dataset is split using train_test_split function which splits the data randomly into train and test datasets.

Conclusion:

Machine learning activity was performed successfully for the given dataset. This was done by using a knowledge discovery process consisting of the phases, gathering data, pre-processing data, choosing an appropriate data mining approach to find patterns among the data and interpreting them.

After KDD process, feature engineering is carried out on the dataset for machine learning readiness. Then 3 different types of supervised ML algorithms are developed namely Classification type, Regression type and Ensemble type. Out of these 3, classification and regression, both gave best performance with maximum Pearson r score of 0.79 and 0.82 respectively. Generally, a good Pearson r score is assumed to be above 0.75. However, this is not a hard written threshold.

Overall, classification and regression are two fundamental types of supervised learning used to make predictions on data, while ensemble learning is a technique to improve the accuracy and robustness of these predictions by combining multiple models.

Applications of Machine Learning:

Now, we know that Machine Learning is a technique of training machines to perform the activities a human brain can do, albeit bit faster and better than an average human-being. Today we have seen that the machines can beat human champions in games such as Chess, AlphaGO, which are considered very complex. You have seen that machines can be trained to perform human activities in several areas and can aid humans in living better lives.

However, machine learning also has some challenges, such as the need for high-quality data, the potential for biased or inaccurate models, and the lack of transparency in some models.

Overall, machine learning is a rapidly growing field with numerous applications and potential benefits, as well as challenges that must be addressed to ensure the responsible and ethical use of machine learning technologies.

Thank You