OBJECTIVE

To predicted city-cycle fuel consumption in miles per gallon in terms of 3 multivalued discrete and 5 continuous

DATASET INFORMATION

1. Title: Auto-Mpg Data

2. Sources:

   (a) Origin: This dataset was taken from the StatLib library which is maintained at Carnegie Mellon University. The dataset was used in the 1983 American Statistical Association Exposition.

   (c) Date: July 7, 1993

3. Past Usage:

   - See 2b (above)
   - Quinlan,R. (1993). Combining Instance-Based and Model-Based Learning. In Proceedings on the Tenth International Conference of Machine Learning, 236-243, University of Massachusetts, Amherst. Morgan Kaufmann.

4. Relevant Information:

   This dataset is a slightly modified version of the dataset provided in the StatLib library. In line with the use by Ross Quinlan (1993) in predicting the attribute "mpg", 8 of the original instances were removed because they had unknown values for the "mpg" attribute. The original dataset is available in the file "auto-mpg.data-original".

   "The data concerns city-cycle fuel consumption in miles per gallon, to be predicted in terms of 3 multivalued discrete and 5 continuous attributes." (Quinlan, 1993)

5. Number of Instances: 398

6. Number of Attributes: 9 including the class attribute

7. Attribute Information:

   1. mpg: continuous
   2. cylinders: multi-valued discrete
   3. displacement: continuous
   4. horsepower: continuous
   5. weight: continuous
   6. acceleration: continuous
   7. model year: multi-valued discrete
   8. origin: multi-valued discrete
   9. car name: string (unique for each instance)

8. Missing Attribute Values: horsepower has 6 missing values

Link(https://archive.ics.uci.edu/ml/datasets/Auto+MPG)

	mpg	cylinders	displacement	horsepower	weight	acceleration	model_year	origin	car_name
0	18.0	8	307	130.0	3504	12.0	70	1	""chevrolet chevelle malibu"""
1	15.0	8	350	165.0	3693	11.5	70	1	""buick skylark 320"""
2	18.0	8	318	150.0	3436	11.0	70	1	""plymouth satellite"""
3	16.0	8	304	150.0	3433	12.0	70	1	""amc rebel sst"""
4	17.0	8	302	140.0	3449	10.5	70	1	""ford torino"""

EXPLORATORY DATA ANALYSIS

1) MISSIN VALUES

- Feature horsepower is having 6 missing values

```
data.isna().sum()
```

```
mpg             0
cylinders       0
displacement    0
horsepower      6
weight          0
acceleration    0
model_year      0
origin          0
car_name        0
dtype: int64
```

- Horsepower is a continues feature
- We will impute missing values with mean

```
data['horsepower']=data['horsepower'].fillna(data['horsepower'].mean())
```

2) ENCODING DATA

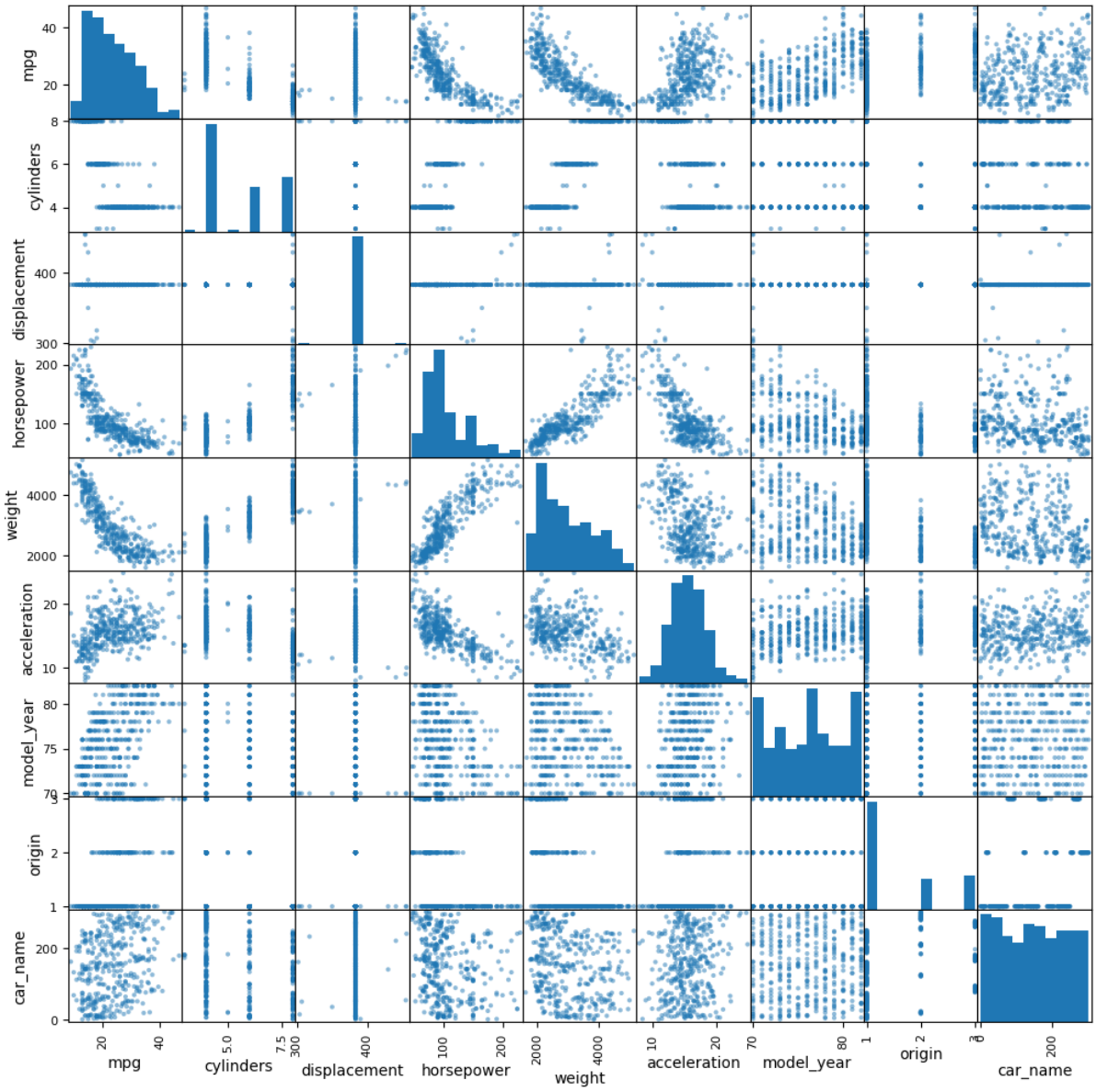
In my dataset car_name is having text data so we will encode it to use in further models

```
label_encoder = preprocessing.LabelEncoder()

data['car_name']= label_encoder.fit_transform(data['car_name'])
data['car_name'].unique()
```

3) SCATTER PLOT MATRIX

Plotiing Scatter plot matrix to see the relation between multiple features
A scatter plot matrix, also known as a pairs plot, is a visualization technique used to explore the relationships between multiple variables simultaneously. It displays a grid of scatter plots, where each plot represents the relationship between two variables.
In a scatter plot matrix, each variable is plotted against all other variables in the dataset. The diagonal of the matrix typically contains histograms or density plots of the individual variables. The off-diagonal cells show scatter plots with one variable on the x-axis and another on the y-axis, representing the relationship between those variables.



4) SPLITTING THE DATASET

```
y=data[['mpg']]
X=data[['cylinders','displacement','horsepower','weight','acceleration','model_year','origin','car_name']]
X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.25)
```

FEATURE ENGINEERING

1) FEATURE SCALING

StandardScaler is a popular technique used for standardizing or normalizing numerical features in machine learning and data preprocessing. It is part of the preprocessing module in libraries such as scikit-learn in Python.

The purpose of StandardScaler is to transform the data such that it has a mean of 0 and a standard deviation of 1. This process is also known as z-score normalization or standardization.

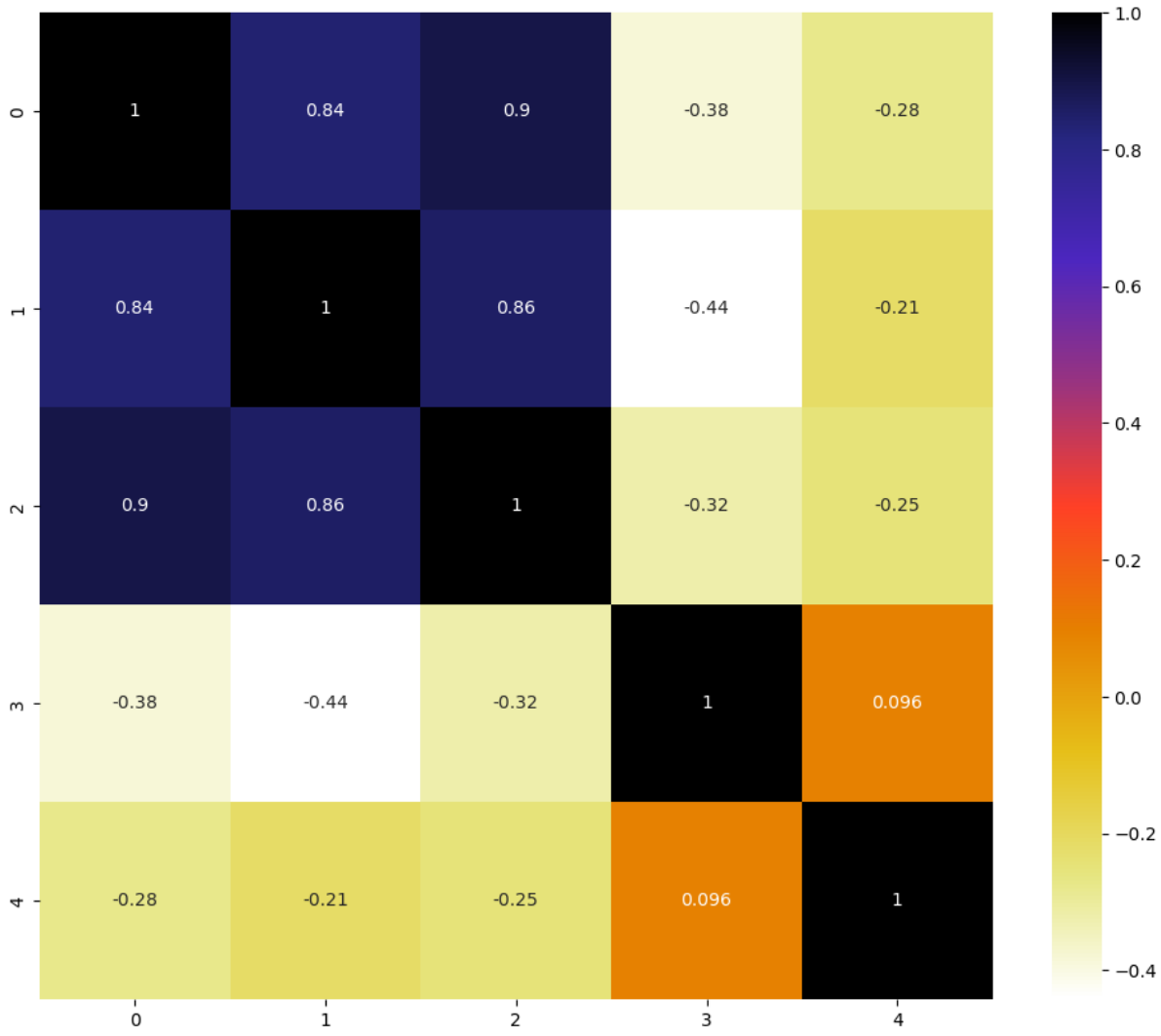
```
scaler = StandardScaler()
X_train=pd.DataFrame(scaler.fit_transform(X_train))
X_test=pd.DataFrame(scaler.fit_transform(X_test))
```

2) **FEATURE ELEMINATION**

For feature elimination we will use RFE RFE stands for Recursive Feature Elimination. It is a feature selection technique used in machine learning to select the most important features from a given dataset. RFE works by repeatedly training a model on the full set of features and eliminating the least important features until a desired number of features is reached.

```
X_train_RFE=X_train
X_test_RFE=X_test
estimator = RandomForestRegressor(criterion="squared_error")
model = RFE(estimator, n_features_to_select=5, step=1)
model = model.fit(X_train_RFE, y_train)
```

```
X_train_RFE.columns[(model.get_support())]
X_train_selected_RFE=model.transform(X_train_RFE)
X_test_selected_RFE=model.transform(X_test_RFE)
```



LINEAR REGRESSION MODEL

Linear regression is a statistical modeling technique used to model the relationship between a dependent variable and one or more independent variables. It assumes a linear relationship between the predictor variables and the response variable, aiming to find the best-fitting line or hyperplane that minimizes the sum of squared errors between the predicted and actual values.

```
regressor=LinearRegression()
regressor.fit(X_train_selected_RFE,y_train)
```

```
LinearRegression()
```

```
y_pred=regressor.predict(X_test_selected_RFE)
```

```
mae_LR=mean_absolute_error(y_test,y_pred)
mse_LR=mean_squared_error(y_test,y_pred)
rmse_LR=(mean_squared_error(y_test,y_pred))**(1/2)
r2_score_LR=r2_score(y_test,y_pred)
```

DECISION TREE MODEL

A decision tree is a predictive modeling technique used for both classification and regression tasks in machine learning. It is a flowchart-like structure where internal nodes represent features or attributes, branches represent decision rules, and leaf nodes represent the outcome or predicted value. Decision trees are particularly useful for understanding complex decision-making processes and extracting valuable insights.

The decision tree algorithm recursively partitions the data based on different features, aiming to create homogeneous subsets with respect to the target variable. This partitioning is done by selecting the best feature that optimally splits the data, usually based on metrics like Gini impurity or entropy. The process continues until a stopping criterion is met, such as reaching a maximum depth or achieving a minimum number of data points in a leaf node.

1) Parameter tuining to find best criterion for root node

```
clf=DecisionTreeRegressor()
parameters = {'criterion':["squared_error", "friedman_mse", "absolute_error", "poisson"],
              'splitter':["best", "random"],
              }
cv=GridSearchCV(clf,param_grid=parameters,cv=5,scoring='accuracy')
cv.fit(X_train_selected_RFE,y_train)
```

```
    return self._sign * self._score_func(y_true, y_pred, **self._kwargs)
  File "C:\ProgramData\Anaconda3\lib\site-packages\sklearn\metrics\_classification.py", lin
    y_type, y_true, y_pred = _check_targets(y_true, y_pred)
  File "C:\ProgramData\Anaconda3\lib\site-packages\sklearn\metrics\_classification.py", lin
    raise ValueError("{0} is not supported".format(y_type))
ValueError: continuous is not supported

  warnings.warn(
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\model_selection\_validation.py:770: User
on this train-test partition for these parameters will be set to nan. Details:
Traceback (most recent call last):
  File "C:\ProgramData\Anaconda3\lib\site-packages\sklearn\model_selection\_validation.py",
    scores = scorer(estimator, X_test, y_test)
  File "C:\ProgramData\Anaconda3\lib\site-packages\sklearn\metrics\_scorer.py", line 216, i
    return self._score(
  File "C:\ProgramData\Anaconda3\lib\site-packages\sklearn\metrics\_scorer.py", line 264, i
    return self._sign * self._score_func(y_true, y_pred, **self._kwargs)
  File "C:\ProgramData\Anaconda3\lib\site-packages\sklearn\metrics\_classification.py", lin
    y_type, y_true, y_pred = _check_targets(y_true, y_pred)
  File "C:\ProgramData\Anaconda3\lib\site-packages\sklearn\metrics\_classification.py", lin
```

```
cv.best_params_
```

```
{'criterion': 'squared_error', 'splitter': 'best'}
```

2) Creating Model

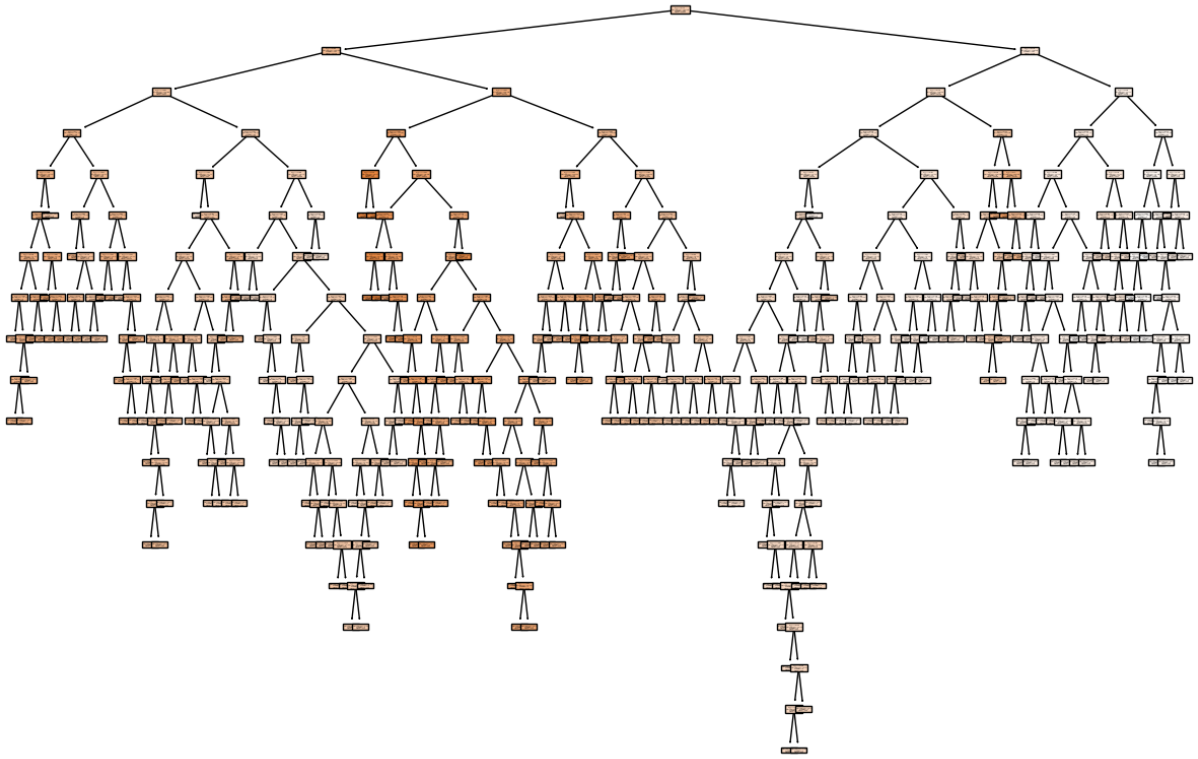
```
clf=DecisionTreeRegressor(criterion= 'squared_error',splitter='best')
```

```
clf.fit(X_train_selected_RFE,y_train)
```

```
DecisionTreeRegressor()
```

```
y_pred_DT=clf.predict(X_test_selected_RFE)
```

```
mae_DT=mean_absolute_error(y_test,y_pred_DT)
mse_DT=mean_squared_error(y_test,y_pred_DT)
rmse_DT=(mean_squared_error(y_test,y_pred_DT))**(1/2)
r2_score_DT=r2_score(y_test,y_pred_DT)
```



RANDOM FOREST MODEL

Random Forest is an ensemble learning method that combines multiple decision trees to create a more robust and accurate predictive model. It is widely used for both classification and regression tasks in machine learning.

The Random Forest algorithm builds a collection of decision trees, where each tree is trained on a random subset of the training data and a random subset of

the features. This randomness helps to reduce overfitting and increase the diversity among the trees in the ensemble.

During the training process, each tree in the Random Forest independently makes predictions for the input data. For classification tasks, the final prediction is determined by majority voting, where the class with the most votes across all trees is selected as the predicted class. For regression tasks, the final prediction is typically the average or median of the predictions from all the trees.

```
from sklearn.ensemble import RandomForestRegressor
rf_regressor=RandomForestRegressor(criterion= 'squared_error')
rf_regressor.fit(X_train_selected_RFE,y_train)
y_pred_RF=rf_regressor.predict(X_test_selected_RFE)
mae_RF=mean_absolute_error(y_test,y_pred_RF)
mse_RF=mean_squared_error(y_test,y_pred_RF)
rmse_RF=(mean_squared_error(y_test,y_pred_RF))**(1/2)
r2_score_RF=r2_score(y_test,y_pred_RF)
```

BASIC NEURAL NETWORK

A basic neural network, also known as a feedforward neural network or multi-layer perceptron (MLP), is a fundamental architecture used in machine learning and artificial intelligence. It consists of three main components: an input layer, one or more hidden layers, and an output layer. Each layer is composed of interconnected neurons or nodes that process and propagate information through the network.

The input layer serves as the entry point for the data into the network. It receives the input features or attributes and passes them forward to the subsequent layers. Each neuron in the input layer represents a specific feature of the input data, but no computation is performed within this layer. Its primary role is to transmit the input data to the hidden layers.

The hidden layers are located between the input and output layers. These layers contain neurons that perform computations on the input data received from the previous layer. Each neuron takes the weighted sum of the inputs, applies an activation function to introduce non-linearity, and produces an output. The number of hidden layers and neurons in each layer can be adjusted based on the complexity of the problem and the available data. Common

activation functions used in hidden layers include the rectified linear unit (ReLU), sigmoid, or hyperbolic tangent.

The output layer is the final layer of the neural network and generates the predicted output based on the computations performed in the hidden layers. The number of neurons in the output layer depends on the nature of the problem. For regression tasks, there is usually a single output neuron that represents the predicted numerical value. In classification tasks, the output layer may have multiple neurons, with each neuron corresponding to a class label. The activation function used in the output layer depends on the task. For example, a linear activation function may be used for regression, while a sigmoid or softmax activation function is commonly used for binary or multi-class classification, respectively.

Connections between neurons in adjacent layers are established, and each connection is assigned a weight that determines the strength or importance of the connection. During training, these weights are adjusted through a process called backpropagation, where the neural network learns to minimize the discrepancy between the predicted output and the actual output by iteratively updating the weights based on the error.

In addition to weights, each neuron (except those in the input layer) typically has an associated bias term. The bias allows the neuron to shift the activation function's output, providing more flexibility in fitting the data.

By configuring the number of layers, the number of neurons in each layer, the activation functions, and the weights, a basic neural network can learn complex patterns and relationships in the input data, making it suitable for various tasks such as regression and classification. The versatility of neural networks lies in their ability to capture non-linear relationships, making them powerful tools in machine learning and AI applications.

```
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from tensorflow.keras.activations import linear, relu
```

```
input_shape = [X_train_selected_RFE.shape[1]]
model = Sequential(
    [

        tf.keras.Input(shape=input_shape),    #specify input shape
        Dense(units=34, activation='linear',name='L1'),
        Dense(units=17,activation='relu',name='L2'),
        Dense(units=8,activation='relu',name='L3'),
        Dense(units=1,activation='linear',name='L4'),

    ], name = "my_model"
)
```

```
model.summary()
```

```
Model: "my_model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 L1 (Dense)                  (None, 34)                204

 L2 (Dense)                  (None, 17)                595

 L3 (Dense)                  (None, 8)                 144

 L4 (Dense)                  (None, 1)                 9

=================================================================
Total params: 952
Trainable params: 952
Non-trainable params: 0
_________________________________________________________________
```

```
model.compile(optimizer="Adam", loss="mae", metrics=["mae", "acc"])
```

```
model.fit(
    X_train_selected_RFE,y_train,
    epochs=100
)
```

```
Epoch 1/100
10/10 [==============================] - 0s 1ms/step - loss: 23.5116 - mae: 23.5116 - acc: 0.0000e+00
Epoch 2/100
10/10 [==============================] - 0s 1ms/step - loss: 23.1443 - mae: 23.1443 - acc: 0.0000e+00
Epoch 3/100
10/10 [==============================] - 0s 967us/step - loss: 22.7982 - mae: 22.7982 - acc: 0.0000e+00
Epoch 4/100
10/10 [==============================] - 0s 1ms/step - loss: 22.4045 - mae: 22.4045 - acc: 0.0000e+00
Epoch 5/100
10/10 [==============================] - 0s 902us/step - loss: 21.9457 - mae: 21.9457 - acc: 0.0000e+00
Epoch 6/100
10/10 [==============================] - 0s 1ms/step - loss: 21.3428 - mae: 21.3428 - acc: 0.0000e+00
Epoch 7/100
10/10 [==============================] - 0s 1ms/step - loss: 20.6072 - mae: 20.6072 - acc: 0.0000e+00
Epoch 8/100
10/10 [==============================] - 0s 1ms/step - loss: 19.7663 - mae: 19.7663 - acc: 0.0000e+00
Epoch 9/100
10/10 [==============================] - 0s 1ms/step - loss: 18.9272 - mae: 18.9272 - acc: 0.0000e+00
Epoch 10/100
```

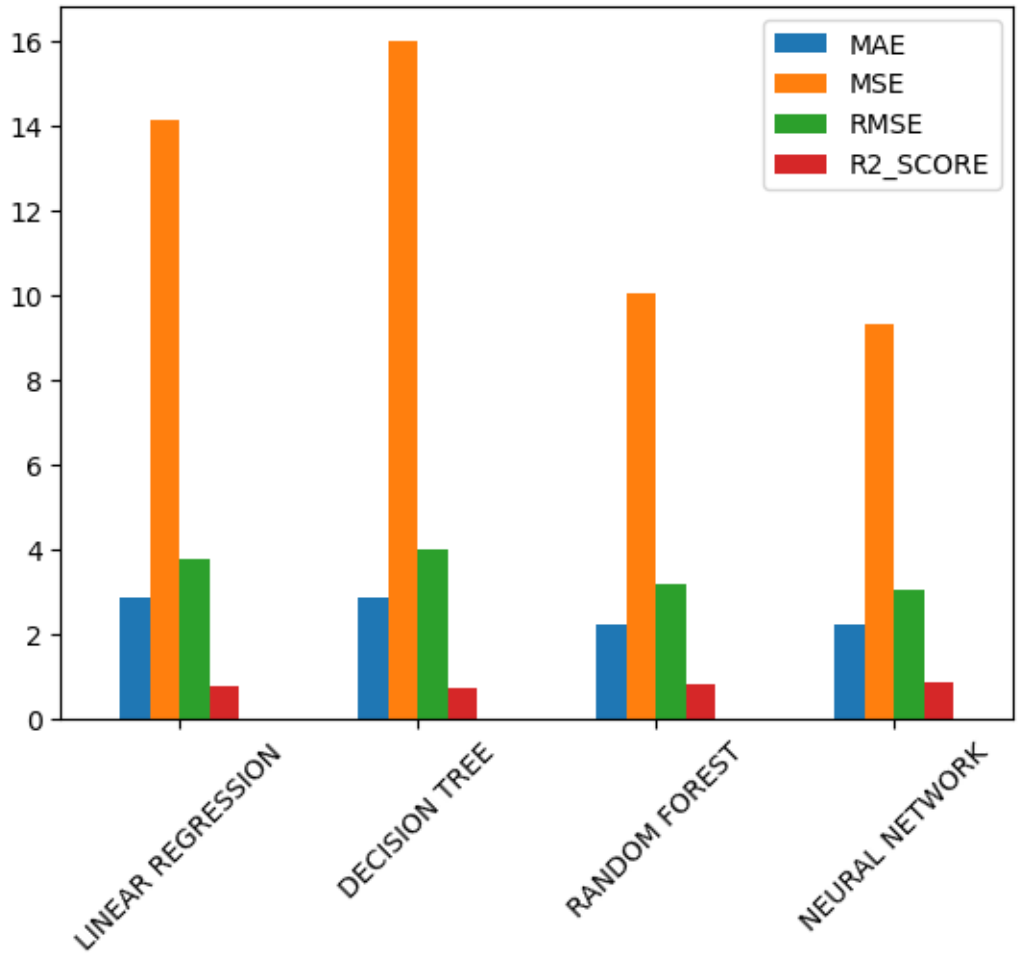
```
y_pred_DL=model.predict(X_test_selected_RFE)
```

```
4/4 [==============================] - 0s 2ms/step
```

```
mae_DL=mean_absolute_error(y_test,y_pred_DL)
mse_DL=mean_squared_error(y_test,y_pred_DL)
rmse_DL=(mean_squared_error(y_test,y_pred_DL))**(1/2)
r2_score_DL=r2_score(y_test,y_pred_DL)
```

RESULTS

	MAE	MSE	RMSE	R2_SCORE
LINEAR REGRESSION	2.871618	14.149374	3.761565	0.765743
DECISION TREE	2.864000	16.012200	4.001525	0.734903
RANDOM FOREST	2.231970	10.060684	3.171858	0.833436
NEURAL NETWORK	2.199449	9.299958	3.049583	0.846030



CONCLUSION

Based on the evaluation metrics obtained for the different regression models (Linear Regression, Decision Tree, Random Forest, Neural Network), we can draw the following conclusions.

The Linear Regression model performs reasonably well with moderate accuracy. It achieves a relatively low Mean Absolute Error (MAE) and Root Mean Squared Error (RMSE), indicating that the average prediction error is around 2.87 and 3.76 units, respectively. The R-squared score of 0.765743 suggests that approximately 76.57% of the variability in the target variable can be explained by the model.

The Decision Tree model exhibits comparable performance to Linear Regression. It slightly outperforms Linear Regression in terms of MAE and RMSE, indicating a slightly lower average prediction error. However, it has a slightly higher Mean Squared Error (MSE) and lower R-squared score, suggesting that the model's ability to explain the variability in the target variable is slightly lower.

The Random Forest model demonstrates improved performance compared to both Linear Regression and Decision Tree. It achieves the lowest MAE, MSE, and RMSE among the evaluated models, indicating better prediction accuracy and lower average error. Additionally, the R-squared score of 0.833436 suggests that approximately 83.34% of the variability in the target variable can be explained by the model, which is relatively high.

The Neural Network model performs the best among the evaluated models. It achieves the lowest MAE, MSE, and RMSE, indicating the highest prediction accuracy and lowest average error. Furthermore, the R-squared score of 0.846030 suggests that approximately 84.60% of the variability in the target variable can be explained by the model, demonstrating a strong ability to capture the data patterns.

In summary, based on the provided evaluation metrics, the Neural Network model exhibits the best performance, followed by the Random Forest model. These models outperform both Linear Regression and Decision Tree models in terms of accuracy and the ability to explain the variability in the target variable.

GITHUB REPOSIRATORY:-RAJATMOUNDEKAR/Data_Analytics (github.com)