5_March_Assignment

March 5, 2024

[ ]:
```python
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
```

[ ]:
```python
df = pd.read_excel("datasets/auto_prize data.xlt")
df.head()
```

[ ]:
```
   symboling  normalized-losses  wheel-base      length      width     height  \
0          5                164   99.800003  176.600006  66.199997  54.299999
1          5                164   99.400002  176.600006  66.400002  54.299999
2          4                158  105.800003  192.699997  71.400002  55.700001
3          4                158  105.800003  192.699997  71.400002  55.900002
4          5                192  101.199997  176.800003  64.800003  54.299999

   curb-weight  engine-size  bore  stroke  compression-ratio  horsepower  \
0         2337          109  3.19     3.4               10.0         102
1         2824          136  3.19     3.4                8.0         115
2         2844          136  3.19     3.4                8.5         110
3         3086          131  3.13     3.4                8.3         140
4         2395          108  3.50     2.8                8.8         101

   peak-rpm  city-mpg  highway-mpg  target
0      5500        24           30   13950
1      5500        18           22   17450
2      5500        19           25   17710
3      5500        17           20   23875
4      5800        23           29   16430
```

[ ]:
```python
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 159 entries, 0 to 158
Data columns (total 16 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
 0   symboling          159 non-null    int64
 1   normalized-losses  159 non-null    int64
 2   wheel-base         159 non-null    float64
```

```
 3   length             159 non-null    float64
 4   width              159 non-null    float64
 5   height             159 non-null    float64
 6   curb-weight        159 non-null    int64
 7   engine-size        159 non-null    int64
 8   bore               159 non-null    float64
 9   stroke             159 non-null    float64
 10  compression-ratio  159 non-null    float64
 11  horsepower         159 non-null    int64
 12  peak-rpm           159 non-null    int64
 13  city-mpg           159 non-null    int64
 14  highway-mpg        159 non-null    int64
 15  target             159 non-null    int64
dtypes: float64(7), int64(9)
memory usage: 20.0 KB
```

[ ]:
```
df.describe()
```

[ ]:
```
        symboling  normalized-losses  wheel-base      length       width  \
count  159.000000         159.000000  159.000000  159.000000  159.000000
mean     3.735849         121.132075   98.264151  172.413837   65.607547
std      1.193086          35.651285    5.167417   11.523177    1.947883
min      1.000000          65.000000   86.599998  141.100006   60.299999
25%      3.000000          94.000000   94.500000  165.650002   64.000000
50%      4.000000         113.000000   96.900002  172.399994   65.400002
75%      5.000000         148.000000  100.799999  177.800003   66.500000
max      6.000000         256.000000  115.599998  202.600006   71.699997

           height  curb-weight  engine-size        bore      stroke  \
count  159.000000   159.000000   159.000000  159.000000  159.000000
mean    53.899371  2461.138365   119.226415    3.300126    3.236352
std      2.268761   481.941321    30.460791    0.267336    0.294888
min     49.400002  1488.000000    61.000000    2.540000    2.070000
25%     52.250000  2065.500000    97.000000    3.050000    3.105000
50%     54.099998  2340.000000   110.000000    3.270000    3.270000
75%     55.500000  2809.500000   135.000000    3.560000    3.410000
max     59.799999  4066.000000   258.000000    3.940000    4.170000

       compression-ratio  horsepower     peak-rpm    city-mpg  highway-mpg  \
count         159.000000  159.000000   159.000000  159.000000   159.000000
mean           10.161132   95.836478  5113.836478   26.522013    32.081761
std             3.889475   30.718583   465.754864    6.097142     6.459189
min             7.000000   48.000000  4150.000000   15.000000    18.000000
25%             8.700000   69.000000  4800.000000   23.000000    28.000000
50%             9.000000   88.000000  5200.000000   26.000000    32.000000
75%             9.400000  114.000000  5500.000000   31.000000    37.000000
max            23.000000  200.000000  6600.000000   49.000000    54.000000
```

```
            target
count     159.000000
mean    11445.729560
std      5877.856195
min      5118.000000
25%      7372.000000
50%      9233.000000
75%     14719.500000
max     35056.000000
```

[ ]:
```
df.isnull().sum()
```

[ ]:
```
symboling            0
normalized-losses    0
wheel-base           0
length               0
width                0
height               0
curb-weight          0
engine-size          0
bore                 0
stroke               0
compression-ratio    0
horsepower           0
peak-rpm             0
city-mpg             0
highway-mpg          0
target               0
dtype: int64
```

[ ]:
```
X = df.drop(columns=["target"])
y = df.target
# store X in df for preprocessing
df = X
```

1 Categorise Columns

[ ]:
```
num_cols = list(df.select_dtypes(include=['int', 'float']).columns)
cat_cols = list(df.select_dtypes(exclude=['int', 'float']).columns)
```

[ ]:
```
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler, OrdinalEncoder
from sklearn.impute import SimpleImputer
from sklearn.base import BaseEstimator, TransformerMixin
```

2 Custom IQR Removal Transformer

[ ]:
```
class Outlier_Remover(BaseEstimator, TransformerMixin):

    def __init__(self, action='keep'):
        self.action = action

    def fit(self, X, y=None):
        self.median_ = np.median(X, axis=0)
        return self

    def transform(self, X):
        Q1 = np.percentile(X, 25, axis=0)
        Q3 = np.percentile(X, 75, axis=0)

        IQR = Q3 - Q1

        lower = Q1 - 1.5*IQR
        upper = Q3 + 1.5*IQR

        outlier_mask = (X < lower) | (X > upper)

        if self.action == 'drop':
            return X[~outlier_mask]
        else:
            for i in range(X.shape[1]):
                X[:, i][outlier_mask[:, i]] = self.median_[i]
            return X
```

3 Define Pipeline's for both columns

[ ]:
```
cat_preprocessor = Pipeline(steps=[
    ("cat_null_handler",SimpleImputer(missing_values=np.
 ↪nan,strategy="most_frequent")),
    ("cat_enocder",OrdinalEncoder()),
])


num_preprocessor = Pipeline(steps=[
    ("num_null_handler",SimpleImputer(missing_values=np.nan,strategy='median')),
    ("num_outlier_remover",Outlier_Remover(action="keep")),
    ("num_scaler",StandardScaler()),
])
```

4 Preprocess data with Pipeline

[ ]:
```
df[num_cols] = num_preprocessor.fit_transform(df[num_cols])
```

[ ]:
```
# Set x back to df
X = df
```

5 Train Test Split

[ ]:
```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X,y,test_size=0.2,␣
 ↪random_state=42)
```

[ ]:
```
def evaluate_model(y_test,y_pred):
    from sklearn import metrics
    import numpy as np
    mse = metrics.mean_squared_error(y_test,y_pred)
    rmse  =np.sqrt(mse)
    mae = metrics.mean_absolute_error(y_test,y_pred)
    r2 = metrics.r2_score(y_test,y_pred)

    print(f'Mean Squared Error (MSE): {mse}')
    print(f'Root Mean Squared Error (RMSE): {rmse}')
    print(f'Mean Absolute Error (MAE): {mae}')
    print(f'R-squared (R^2): {r2}')
```

6 Train Model's and Evaluation

- Let's try training the linear regression model

[ ]:
```
from sklearn.linear_model import LinearRegression
from sklearn.metrics import r2_score
le = LinearRegression()
le.fit(X_train,y_train)
print("Training: ")
evaluate_model(y_train,le.predict(X_train))
print("\n\nTesting: ")
evaluate_model(y_test,le.predict(X_test))
```

```
Training:
Mean Squared Error (MSE): 8726283.555379074
Root Mean Squared Error (RMSE): 2954.0283606253806
Mean Absolute Error (MAE): 2199.5944062267413
R-squared (R^2): 0.772354657345086


Testing:
```

```
Mean Squared Error (MSE): 5936549.426168845
Root Mean Squared Error (RMSE): 2436.503524760193
Mean Absolute Error (MAE): 1946.2785585578536
R-squared (R^2): 0.6663853415029894
```

- Let's try training Decision Tree regressor

[ ]:
```
from sklearn.tree import DecisionTreeRegressor
dt = DecisionTreeRegressor(criterion="poisson")
dt.fit(X_train,y_train)
print("Training: ")
evaluate_model(y_train,dt.predict(X_train))
print("\n\nTesting: ")
evaluate_model(y_test,dt.predict(X_test))
```

```
Training:
Mean Squared Error (MSE): 28454.72440944882
Root Mean Squared Error (RMSE): 168.68528213643543
Mean Absolute Error (MAE): 35.826771653543304
R-squared (R^2): 0.9992576925277259


Testing:
Mean Squared Error (MSE): 5641829.53125
Root Mean Squared Error (RMSE): 2375.253571989736
Mean Absolute Error (MAE): 1672.78125
R-squared (R^2): 0.6829476355289112
```

[ ]:
```
from sklearn.ensemble import RandomForestRegressor
rfr = RandomForestRegressor(random_state=42)
rfr.fit(X_train,y_train)
print("Training: ")
evaluate_model(y_train,rfr.predict(X_train))
print("\n\nTesting: ")
evaluate_model(y_test,rfr.predict(X_test))
```

```
Training:
Mean Squared Error (MSE): 1516010.1972459753
Root Mean Squared Error (RMSE): 1231.263658704331
Mean Absolute Error (MAE): 595.3748031496062
R-squared (R^2): 0.96045135840129


Testing:
Mean Squared Error (MSE): 2404421.3582149316
Root Mean Squared Error (RMSE): 1550.6196691048813
Mean Absolute Error (MAE): 1214.8290625000004
R-squared (R^2): 0.8648793848548044
```

[ ]:
```python
from sklearn.linear_model import SGDRegressor
sgdr = SGDRegressor()
sgdr.fit(X_train,y_train)
print("Training: ")
evaluate_model(y_train,sgdr.predict(X_train))
print("\n\nTesting: ")
evaluate_model(y_test,sgdr.predict(X_test))
```

```
Training:
Mean Squared Error (MSE): 8857164.141244184
Root Mean Squared Error (RMSE): 2976.0988124126834
Mean Absolute Error (MAE): 2174.159986512722
R-squared (R^2): 0.7689403337527965


Testing:
Mean Squared Error (MSE): 5819114.038455376
Root Mean Squared Error (RMSE): 2412.2839879366143
Mean Absolute Error (MAE): 1976.1818775654574
R-squared (R^2): 0.6729848261454978
```

[ ]:
```python
from sklearn.neighbors import KNeighborsRegressor
knr = KNeighborsRegressor()
knr.fit(X_train,y_train)
print("Training: ")
evaluate_model(y_train,knr.predict(X_train))
print("\n\nTesting: ")
evaluate_model(y_test,knr.predict(X_test))
```

```
Training:
Mean Squared Error (MSE): 6305130.136062991
Root Mean Squared Error (RMSE): 2511.0018192074235
Mean Absolute Error (MAE): 1406.7244094488194
R-squared (R^2): 0.8355160589042385


Testing:
Mean Squared Error (MSE): 5623782.782499999
Root Mean Squared Error (RMSE): 2371.4516192619235
Mean Absolute Error (MAE): 1694.3624999999997
R-squared (R^2): 0.6839618037753126
```

7 Conclusion

The RandomForestRegressor model seems to be good as it performs good compared to all other's with some good testing performance : > Testing: Mean Squared Error (MSE): 2404421.3582149316 Root Mean Squared Error (RMSE): 1550.6196691048813 Mean Absolute Error (MAE): 1214.8290625000004 R-squared (R^2): 0.8648793848548044

So the model is good with the least MSE. Hence the RandomForestRegressor Model is the best.

```
[ ]:
```