Multiple Linear Regression Assignment

211091055 Gaytri Thakre

In [1]:
```
import pandas as pd
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score
from statsmodels.regression.linear_model import OLS
import statsmodels.api as sm
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
```

```
c:\Users\HOME\anaconda3\Lib\site-packages\pandas\core\arrays\masked.py:60: UserWarning: Pandas requires vers
ion '1.3.6' or newer of 'bottleneck' (version '1.3.5' currently installed).
  from pandas.core import (
```

In [2]:
```
data = pd.read_csv(r"DSA\Assignment\Toyota.csv")
data.head()
```

Out[2]:

	Id	Model	Price	Age_08_04	Mfg_Month	Mfg_Year	KM	Fuel_Type	HP	Met_Color	...	Central_Lock	Powered_Windows	Pow
0	1	TOYOTA Corolla 2.0 D4D HATCHB TERRA 2/3-Doors	13500	23	10	2002	46986	Diesel	90.0	1	...	1	1	
1	2	TOYOTA Corolla 2.0 D4D HATCHB TERRA 2/3-Doors	13750	23	10	2002	72937	Diesel	90.0	1	...	1	0	
2	3	TOYOTA Corolla 2.0 D4D HATCHB TERRA 2/3-Doors	13950	24	9	2002	41711	Diesel	90.0	1	...	0	0	
3	4	TOYOTA Corolla 2.0 D4D HATCHB TERRA 2/3-Doors	14950	26	7	2002	48000	Diesel	90.0	0	...	0	0	
4	5	TOYOTA Corolla 2.0 D4D HATCHB SOL 2/3-Doors	13750	30	3	2002	38500	Diesel	90.0	0	...	1	1	

5 rows × 38 columns

In [3]:
```
#checking for null values
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1436 entries, 0 to 1435
Data columns (total 38 columns):
 #   Column            Non-Null Count  Dtype  
---  ------            --------------  -----  
 0   Id                1436 non-null   int64  
 1   Model             1436 non-null   object 
 2   Price             1436 non-null   int64  
 3   Age_08_04         1436 non-null   int64  
 4   Mfg_Month         1436 non-null   int64  
 5   Mfg_Year          1436 non-null   int64  
 6   KM                1436 non-null   int64  
 7   Fuel_Type         1436 non-null   object 
 8   HP                1434 non-null   float64
 9   Met_Color         1436 non-null   int64  
 10  Color             1436 non-null   object 
 11  Automatic         1436 non-null   int64  
 12  cc                1436 non-null   int64  
 13  Doors             1436 non-null   int64  
 14  Cylinders         1436 non-null   int64  
 15  Gears             1436 non-null   int64  
 16  Quarterly_Tax     1436 non-null   int64  
 17  Weight            1436 non-null   int64  
 18  Mfr_Guarantee     1436 non-null   int64  
 19  BOVAG_Guarantee   1436 non-null   int64  
 20  Guarantee_Period  1436 non-null   int64  
 21  ABS               1436 non-null   int64  
 22  Airbag_1          1436 non-null   int64  
 23  Airbag_2          1436 non-null   int64  
 24  Airco             1436 non-null   int64  
 25  Automatic_airco   1436 non-null   int64  
 26  Boardcomputer     1436 non-null   int64  
 27  CD_Player         1436 non-null   int64  
 28  Central_Lock      1436 non-null   int64  
 29  Powered_Windows   1436 non-null   int64  
 30  Power_Steering    1436 non-null   int64  
 31  Radio             1436 non-null   int64  
 32  Mistlamps         1436 non-null   int64  
 33  Sport_Model       1436 non-null   int64  
 34  Backseat_Divider  1436 non-null   int64  
 35  Metallic_Rim      1436 non-null   int64  
 36  Radio_cassette    1436 non-null   int64  
 37  Tow_Bar           1436 non-null   int64  
dtypes: float64(1), int64(34), object(3)
memory usage: 426.4+ KB
```

In [4]:
```
data.drop(columns=['Cylinders','Mfg_Month','Mfg_Year','Fuel_Type','Met_Color','Color','Automatic','Quarterly_
```

In [5]:
```
data.head()
```

Out[5]:

	Id	Model	Price	Age_08_04	KM	HP	cc	Doors	Gears	Weight
0	1	TOYOTA Corolla 2.0 D4D HATCHB TERRA 2/3-Doors	13500	23	46986	90.0	2000	3	5	1165
1	2	TOYOTA Corolla 2.0 D4D HATCHB TERRA 2/3-Doors	13750	23	72937	90.0	2000	3	5	1165
2	3	TOYOTA Corolla 2.0 D4D HATCHB TERRA 2/3-Doors	13950	24	41711	90.0	2000	3	5	1165
3	4	TOYOTA Corolla 2.0 D4D HATCHB TERRA 2/3-Doors	14950	26	48000	90.0	2000	3	5	1165
4	5	TOYOTA Corolla 2.0 D4D HATCHB SOL 2/3-Doors	13750	30	38500	90.0	2000	3	5	1170

In [6]:
```
data.isnull().sum()
```

Out[6]:
```
Id           0
Model        0
Price        0
Age_08_04    0
KM           0
HP           2
cc           0
Doors        0
Gears        0
Weight       0
dtype: int64
```

In [7]:
```
# Calculate the average of non-null values in the 'HP' column
average_hp = data['HP'].mean()

# Replace null values in the 'HP' column with the average
data['HP'].fillna(average_hp, inplace=True)
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1436 entries, 0 to 1435
Data columns (total 10 columns):
 #   Column      Non-Null Count  Dtype  
---  ------      --------------  -----  
 0   Id          1436 non-null   int64  
 1   Model       1436 non-null   object 
 2   Price       1436 non-null   int64  
 3   Age_08_04   1436 non-null   int64  
 4   KM          1436 non-null   int64  
 5   HP          1436 non-null   float64
 6   cc          1436 non-null   int64  
 7   Doors       1436 non-null   int64  
 8   Gears       1436 non-null   int64  
 9   Weight      1436 non-null   int64  
dtypes: float64(1), int64(8), object(1)
memory usage: 112.3+ KB

C:\Users\HOME\AppData\Local\Temp\ipykernel_25760\710003151.py:5: FutureWarning: A value is trying to be set
on a copy of a DataFrame or Series through chained assignment using an inplace method.
The behavior will change in pandas 3.0. This inplace method will never work because the intermediate object
on which we are setting values always behaves as a copy.

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method({col: value}, inplace=Tr
ue)' or df[col] = df[col].method(value) instead, to perform the operation inplace on the original object.


  data['HP'].fillna(average_hp, inplace=True)
```

In [8]:
```
X = data[['Age_08_04', 'KM', 'HP','cc','Doors', 'Gears', 'Weight']]
y = data['Price']
```

In [9]:
```
# Splitting data into training and testing
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

In [10]:
```
# model building

model = LinearRegression()
model.fit(X_train, y_train)
```

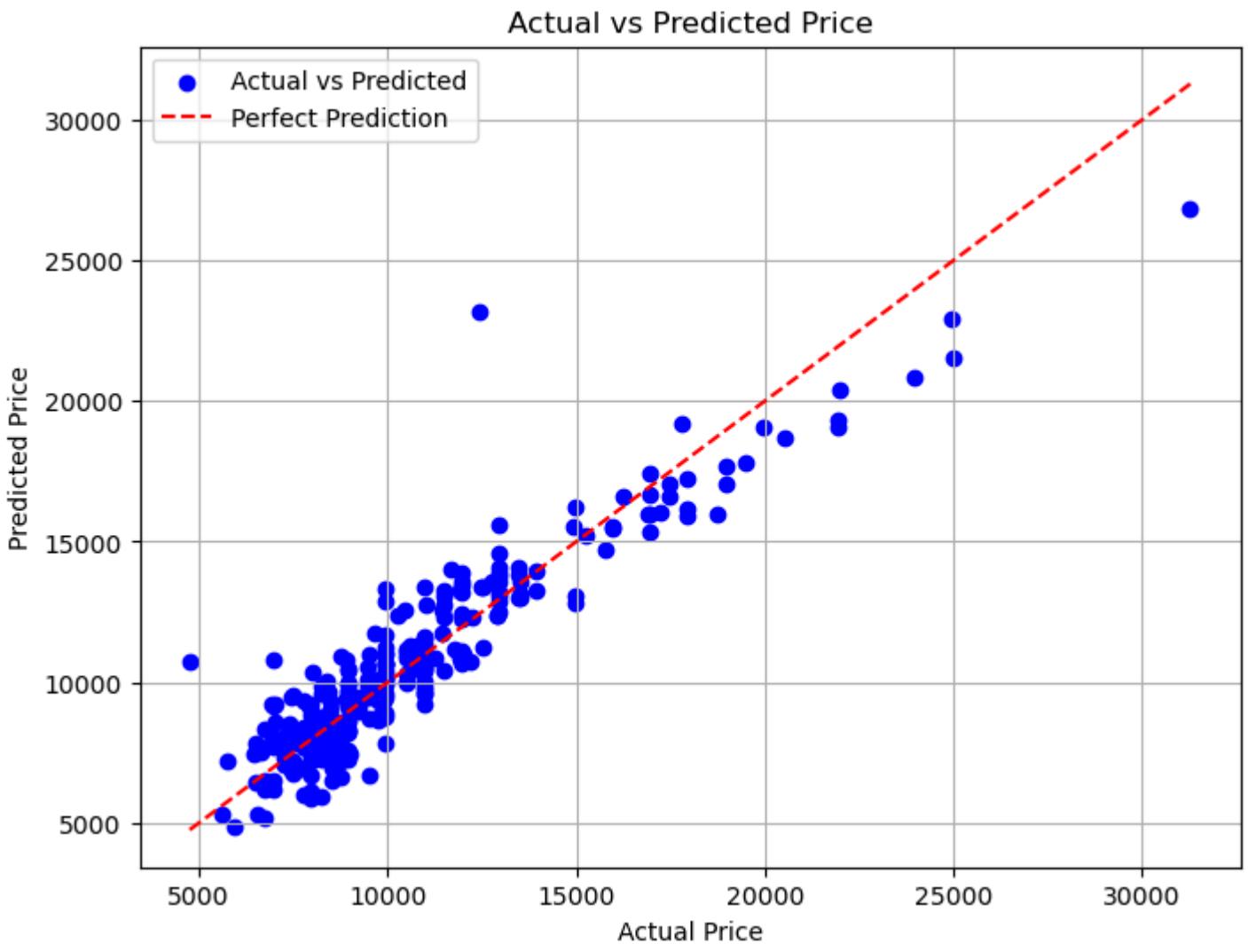
Out[10]:
```
LinearRegression()
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [11]:
```
# Making Predictions
y_pred = model.predict(X_test)
y_pred
```

Out[11]:
```
array([11317.29846269,  8918.78448174,  9508.39211097,  8955.14888673,
       10121.39398108,  7808.73026425,  8751.48049021,  8331.5710554 ,
       14101.48850062, 13179.89223506,  9363.66103953,  9358.44792209,
       12983.08190044, 13084.74314686,  9723.53402998,  8260.78213152,
       12289.14865243, 17457.6433218 ,  7248.99190041,  9199.79143909,
       12781.98460541, 17805.25653403,  7891.731488  , 10501.38181749,
        7384.6863781 , 15982.73181616,  9587.63680684,  7209.61640308,
       15528.95640743, 14575.07570822,  8121.67265842,  9897.36562072,
        8527.39842461, 10105.47486   , 10290.01689199,  8009.79681932,
        8879.03383954,  9970.58683286, 16223.91417702,  9450.5482417 ,
        9895.41549571,  9549.12748319,  7874.0059817 ,  5304.66145181,
        6776.53298118, 17025.58771169,  9439.1487023 , 10676.62851279,
        8753.10136092, 13561.46150248, 13071.11791915,  6336.78660072,
       14720.80364294, 16192.68367949,  5895.88938661,  7710.08351801,
       10433.39346801, 10755.2298625 , 15989.2094808 , 19295.78399874,
       12582.92854649,  8177.98654965,  7820.53806335, 10969.42657911,
       13404.04353174,  8880.94751207, 13246.05392788,  7462.71599004,
        8212.48104622, 10794.32630058, 20392.77082581, 11104.22462733,
       13035.83359397,  7430.88845995, 18680.15237257, 13896.49842862,
        8125.88333407,  7515.31218552,  9823.96008124,  9138.66206306,
       10310.01478154,  8620.79406485, 10451.66616524, 10826.40903424,
        7528.26247066,  6204.35834531, 11113.81553548, 13072.1317895 ,
        9578.97035424, 10938.87174629,  7723.89975973,  6600.62794536,
       10533.16768182, 10858.32297228, 12767.01224126,  9242.24566937,
       21559.31488482,  7931.71163825, 14088.85459766,  8669.86991925,
       10823.74048077,  9633.83193524, 10660.21436746, 16606.48359058,
        8824.9011955 ,  7615.45801056, 16651.21852912,  7726.415331  ,
        9677.6767579 ,  8602.65146125, 11744.10729572, 10718.56852545,
        9842.43269038, 13839.84541861,  9435.55405949,  8149.96780275,
        6441.74808155, 13466.05550582, 12512.58761777, 13992.94875489,
        7855.31260592,  7328.48327203,  8295.43982644, 15343.59777183,
       11700.21465675,  8862.52819095, 13118.57313739,  9214.69253766,
        8484.18050782, 12594.57081096, 11235.38533731, 15437.22859294,
       10550.98892697,  9416.49468055,  6475.60577524,  9482.64225583,
       10336.40764056, 11588.81827877,  8282.88906872,  4859.41257292,
       12405.1571968 , 10868.51239783, 12447.21277639,  8373.85947313,
       12260.18950736, 11342.61022128,  7087.94750705,  8595.35448414,
        8311.14378414, 11308.13632058, 10362.86109319, 15941.00294438,
        8412.61015029,  9122.26913082, 11262.94516294, 11049.8388455 ,
        8739.34212925, 13362.36079671,  9954.97686301, 19185.67754138,
       11175.43728294, 15600.3067613 , 15243.45854883,  7715.34333289,
        8966.37123073, 12840.44466373,  7949.49211599, 13578.48210484,
        8179.69570164,  7553.96411107,  8138.46085534, 12400.4302999 ,
        9337.73397779, 12882.91283948, 10424.71028454,  9440.30235817,
       16630.50776774, 10727.07788155,  9743.30243675, 10580.19675798,
       12291.45071252, 10999.7571833 , 13735.81960649,  7840.44172986,
        9215.61148302,  7537.47949853,  5912.08712983,  8246.02203086,
        9655.60759342, 10955.67122004,  9133.70421686,  7426.82795062,
        6094.97081459, 11237.05601969, 10444.01282666, 10988.30999071,
        6911.83499103,  7176.67878244,  9987.45249597, 17657.61440416,
        9230.96178132,  7697.26348611,  6935.76282976,  8760.59180958,
       16001.67548641, 17033.80944857, 10641.12331639, 10052.91936623,
       10975.56213778,  9122.86725356,  9038.97576428, 11182.3946493 ,
        9880.69075703, 10168.89693847,  6885.99016714, 11747.89001919,
        7859.48484486,  9236.27583405,  7195.96225064, 11575.35928217,
        8612.71492981, 10640.02292362, 10841.13824371, 10145.3293956 ,
       17257.08062768,  6499.00639241, 16035.22803052,  7434.95059674,
       13248.06355443,  7307.32876259, 12346.85440714,  9595.67261001,
       26861.59651671,  7813.29385495,  9342.53176066,  8275.78687473,
       13892.7662371 , 13219.7823173 , 10101.32945727,  8555.39292519,
       13945.51552728, 19086.82280143, 13600.98214274,  9589.78188029,
       20843.86245257,  9723.40163446,  9303.93309249,  9951.7001493 ,
       10014.13073616,  9166.70807671,  8587.71156122, 10435.09180677,
        8612.82477957,  5293.7713379 , 10490.81406101, 10838.20679766,
        6693.82471313, 10048.30431112, 13391.79948721, 10974.06447497,
       12889.00492215,  6935.55416054,  5965.93379116, 10660.53999114,
        6489.80995654, 12382.13934489, 13347.55148902,  5205.90915787,
        8741.22660416,  8006.44385786, 13236.09993988, 15541.16399947,
        6659.60088182, 13551.91800103,  7793.79519491,  7242.56020753,
        8451.35592526,  7852.50242273,  6184.4414426 , 23169.42490472,
       22930.66625603, 10290.05142602,  7197.49733726, 19062.07312988,
        8114.75304238, 13578.89719728, 10788.35719806, 10783.62401512])
```

In [12]:
```
# Plot actual vs predicted prices
plt.figure(figsize=(8, 6))
plt.scatter(y_test, y_pred, color='blue', label='Actual vs Predicted')
plt.plot([min(y_test), max(y_test)], [min(y_test), max(y_test)], color='red', linestyle='--', label='Perfect
plt.xlabel('Actual Price')
plt.ylabel('Predicted Price')
plt.title('Actual vs Predicted Price')
plt.legend()
plt.grid(True)
plt.show()
```



In [13]:
```
# Model Evaluation

mae = mean_absolute_error(y_test, y_pred)
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)
```

In [14]:
```
print("Mean Absolute Error:", mae)
print("Mean Squared Error:", mse)
print("R-squared:", r2)
```

```
Mean Absolute Error: 996.0847838224187
Mean Squared Error: 1978084.8261539815
R-squared: 0.8517487723649542
```

In [15]:
```
# Create a DataFrame with the features of the new data
new_data = pd.DataFrame({'Age_08_04': 22,'KM': 43610,'HP': 192,'cc': 2000,  'Doors': 3,'Gears': 6,'Weight': 1

# Predict the price using the trained model
predicted_price = model.predict(new_data)

print("Predicted Price:", predicted_price)

# without removing outliers
```

```
Predicted Price: [20584.99836958]
```

Removing Outliers

In [16]:
```
# Training initial linear regression model
initial_model = OLS(y_train, sm.add_constant(X_train)).fit()
```

In [17]:
```
# Calculate Cook's distance
influence = initial_model.get_influence()
cooks_distance = influence.cooks_distance[0]

# Set the threshold for Cook's distance
threshold = 1

# Identify outliers
outliers_indices = [i for i, d in enumerate(cooks_distance) if d > threshold]
```

In [18]:
```
X_train_clean = X_train.drop(outliers_indices, errors='ignore')
y_train_clean = y_train.drop(outliers_indices, errors='ignore')

# Retraining
clean_model = LinearRegression()
clean_model.fit(X_train_clean, y_train_clean)
```

Out[18]:
```
LinearRegression()
```
In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [ ]:
```

new_data = {
    'Age_08_04': 22,
    'KM': 43610,
    'HP': 192,
    'cc': 2000,
    'Doors': 3,
    'Gears': 6,
    'Weight': 1185
}
```

In [19]:
```

# Converting the new data to a DataFrame
new_data_df = pd.DataFrame([new_data])

# Predicting price
predicted_price = clean_model.predict(new_data_df)

print("Predicted Price:", predicted_price[0])
```

```
Predicted Price: 20584.998369577228
```