Summative Capstone Project - (Kenny Lim)

To predict Audi used car price analysis

Methodology to use :

1. Random Forest Regressor (RF)
2. Decision Tree (DT)
3. Linear Regression (LR)

In [1]:
```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
import sklearn as sk
import pyodbc
import sys

from sklearn.linear_model import LinearRegression
from sklearn import metrics
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split, cross_val_score
```

Connect SQL database to python using open database connectivity = pyodbc

In [2]:
```
# Connect database and create cursor using python open database connectivity = pyodbc
conn = pyodbc.connect('Driver={SQL Server};'
                      'Server=DESKTOP-KLUE8FK;'
                      'Database=audi_dataset;'   #connect & retrieve from SQL database
                      'Trusted_Connection=yes;')

cur = conn.cursor()
```

Load Data & Create Dataframe

In [3]:
```
#create a dataframe opject df=DataFrame(cur.fetchall())

audi_sql = pd.read_sql_query('SELECT * FROM audi_dataset.dbo.audi', conn)
```

Details of our data frame : audi_sql

In [4]:
```
audi_sql.head(5)
```

Out[4]:

	model	year	price	transmission	mileage	fuelType	tax	mpg	engineSize
0	A1	2017	12500	Manual	15735	Petrol	150	55.4	1.4

	model	year	price	transmission	mileage	fuelType	tax	mpg	engineSize
1	A6	2016	16500	Automatic	36203	Diesel	20	64.2	2.0
2	A1	2016	11000	Manual	29946	Petrol	30	55.4	1.4
3	A4	2017	16800	Automatic	25952	Diesel	145	67.3	2.0
4	A3	2019	17300	Manual	1998	Petrol	145	49.6	1.0

In [5]:
```
print(audi_sql)    # showing head & tail
```

```
      model  year  price transmission  mileage fuelType  tax   mpg  engineSize
0        A1  2017  12500       Manual    15735   Petrol  150  55.4         1.4
1        A6  2016  16500    Automatic    36203   Diesel   20  64.2         2.0
2        A1  2016  11000       Manual    29946   Petrol   30  55.4         1.4
3        A4  2017  16800    Automatic    25952   Diesel  145  67.3         2.0
4        A3  2019  17300       Manual     1998   Petrol  145  49.6         1.0
...     ...   ...    ...          ...      ...      ...  ...   ...         ...
10663    A3  2020  16999       Manual     4018   Petrol  145  49.6         1.0
10664    A3  2020  16999       Manual     1978   Petrol  150  49.6         1.0
10665    A3  2020  17199       Manual      609   Petrol  150  49.6         1.0
10666    Q3  2017  19499    Automatic     8646   Petrol  150  47.9         1.4
10667    Q3  2016  15999       Manual    11855   Petrol  150  47.9         1.4

[10668 rows x 9 columns]
```

In [6]:
```
audi_sql.shape
```

Out[6]:
```
(10668, 9)
```

Step 1 :

Exploratory Data Analysis (EDA)

In [7]:
```
audi_sql.columns
```

Out[7]:
```
Index(['model', 'year', 'price', 'transmission', 'mileage', 'fuelType', 'tax',
       'mpg', 'engineSize'],
      dtype='object')
```

In [8]:
```
audi_sql.info()
```

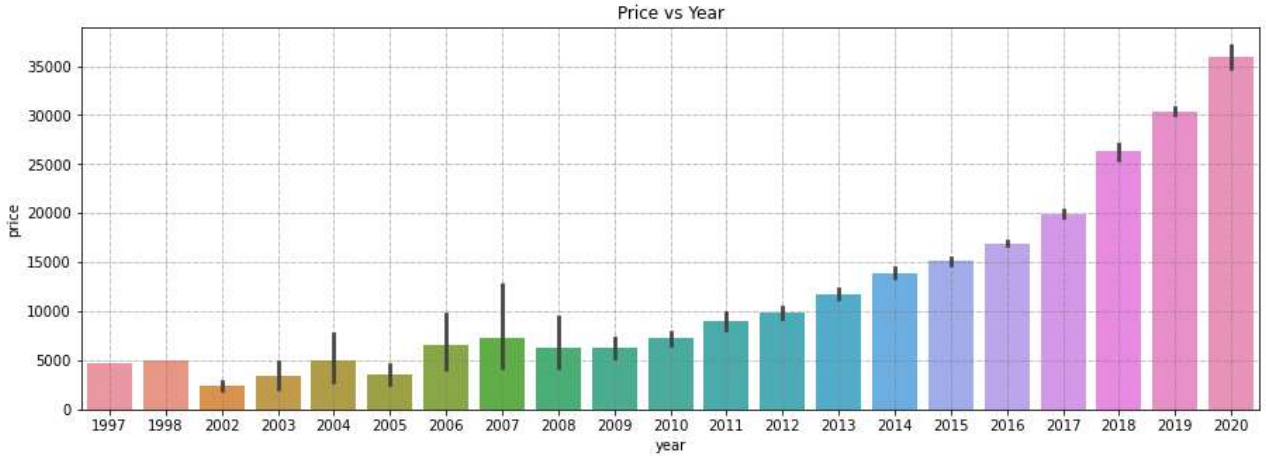
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10668 entries, 0 to 10667
Data columns (total 9 columns):
 #   Column        Non-Null Count  Dtype  
---  ------        --------------  -----  
 0   model         10668 non-null  object 
 1   year          10668 non-null  int64  
 2   price         10668 non-null  int64  
 3   transmission  10668 non-null  object 
 4   mileage       10668 non-null  int64  
 5   fuelType      10668 non-null  object 
 6   tax           10668 non-null  object 
 7   mpg           10668 non-null  float64
 8   engineSize    10668 non-null  float64
dtypes: float64(2), int64(3), object(4)
memory usage: 750.2+ KB
```

Visualizing Data

1. View the Average Price by Year

In [9]:
```
plt.figure(figsize=(15,5),facecolor='w')
#plt.title('Price vs Year')
sns.barplot(x = audi_sql["year"], y = audi_sql["price"])
plt.grid(color='grey', linestyle='-.', linewidth=0.5)
plt.title('Price vs Year')
```

Out[9]:
```
Text(0.5, 1.0, 'Price vs Year')
```



2. Analysis with respect to number of cars

Model vs Car Count

Let's check the price distribution among the Audi's model represented. Hypothesis : if price is very correlated with the model?

In [10]:
```
audi_sql['model'].value_counts()
```

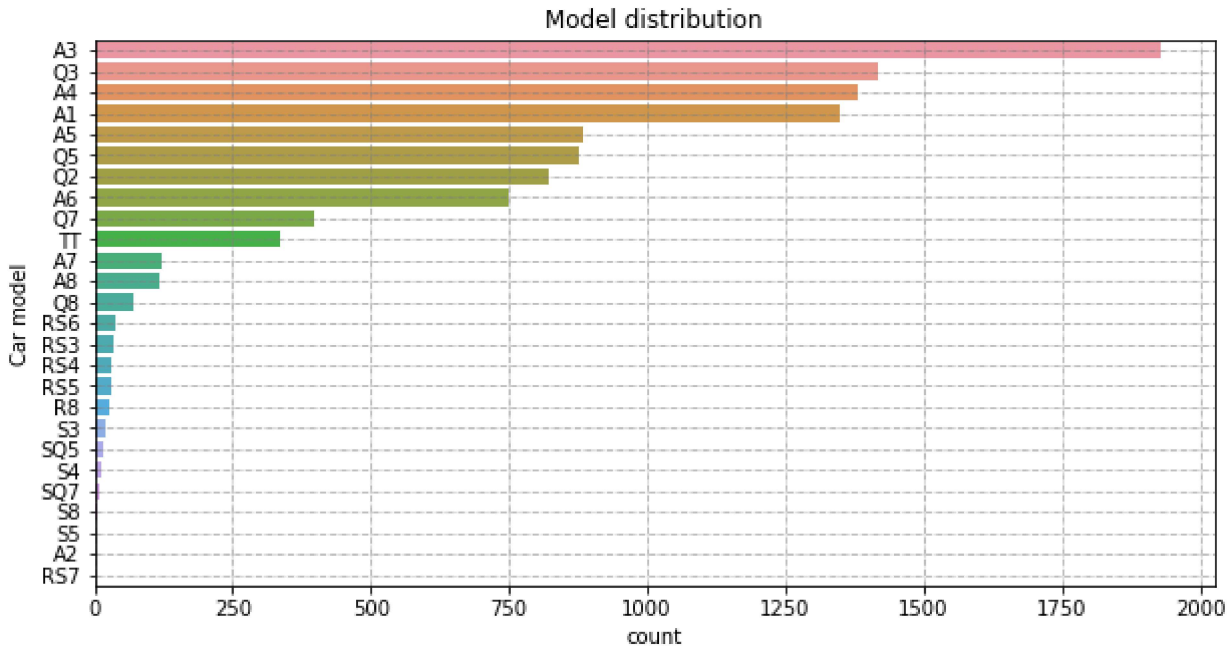
Out[10]:
```
A3      1929
Q3      1417
A4      1381
A1      1347
A5       882
Q5       877
Q2       822
A6       748
Q7       397
TT       336
A7       122
A8       118
Q8        69
RS6       39
RS3       33
RS4       31
RS5       29
R8        28
S3        18
SQ5       16
S4        12
SQ7        8
```

```
S8        4
S5        3
A2        1
RS7       1
Name: model, dtype: int64
```

In [11]:
```
print(audi_sql["model"].value_counts() / len(audi_sql))
fig, ax = plt.subplots(figsize = (10,5))
sns.countplot(y = 'model', data = audi_sql, order = audi_sql['model'].value_counts().in
plt.grid(color='grey', linestyle='-.', linewidth=0.5)
plt.ylabel('Car model')
plt.title('Model distribution')
```

```
A3      0.180821
Q3      0.132827
A4      0.129453
A1      0.126265
A5      0.082677
Q5      0.082208
Q2      0.077053
A6      0.070116
Q7      0.037214
TT      0.031496
A7      0.011436
A8      0.011061
Q8      0.006468
RS6     0.003656
RS3     0.003093
RS4     0.002906
RS5     0.002718
R8      0.002625
S3      0.001687
SQ5     0.001500
S4      0.001125
SQ7     0.000750
S8      0.000375
S5      0.000281
A2      0.000094
RS7     0.000094
Name: model, dtype: float64
```

Out[11]: Text(0.5, 1.0, 'Model distribution')



Most people buy Audi A3 model then other model, follows by Q3, A4/A1. These Top cars most people buy contributed 57% of all AUDI's car, follow with all other models contributing to 43%

'A3' 0.180821

'Q3' 0.132827

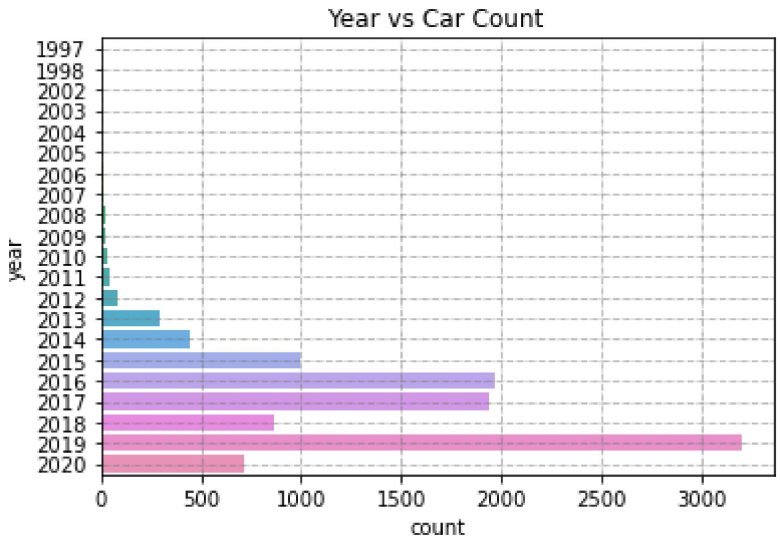
'A4' 0.129453

'A1' 0.126265

Year vs Car Count

In [12]:
```
sns.countplot(y = audi_sql["year"])
plt.grid(color='grey', linestyle='-.', linewidth=0.5)
plt.title("Year vs Car Count")
```

Out[12]:
```
Text(0.5, 1.0, 'Year vs Car Count')
```



Maximum no of cars in data frame is between 2015 to 2019

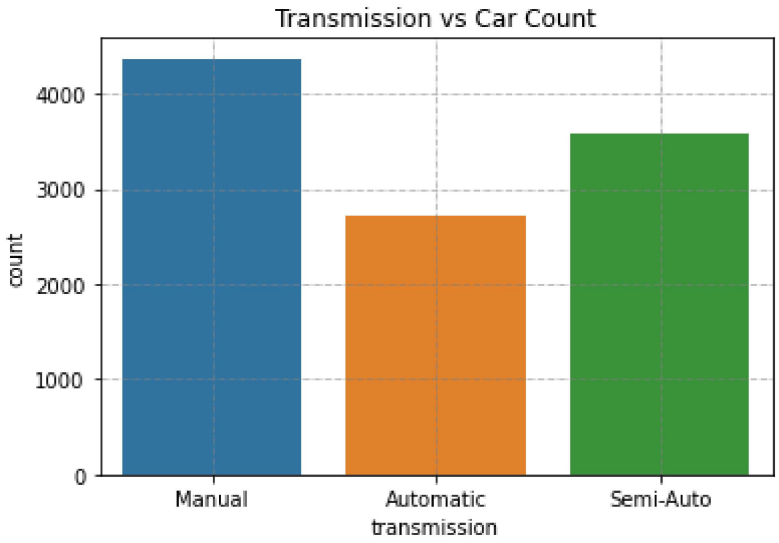
3. Analysis of other features with respect to car count.

In [13]:
```
audi_sql['transmission'].value_counts()
```

Out[13]:
```
Manual       4369
Semi-Auto    3591
Automatic    2708
Name: transmission, dtype: int64
```

In [14]:
```
sns.countplot(x=audi_sql["transmission"])
plt.grid(color='grey', linestyle='-.', linewidth=0.5)
plt.title('Transmission vs Car Count')
```

Out[14]:
```
Text(0.5, 1.0, 'Transmission vs Car Count')
```



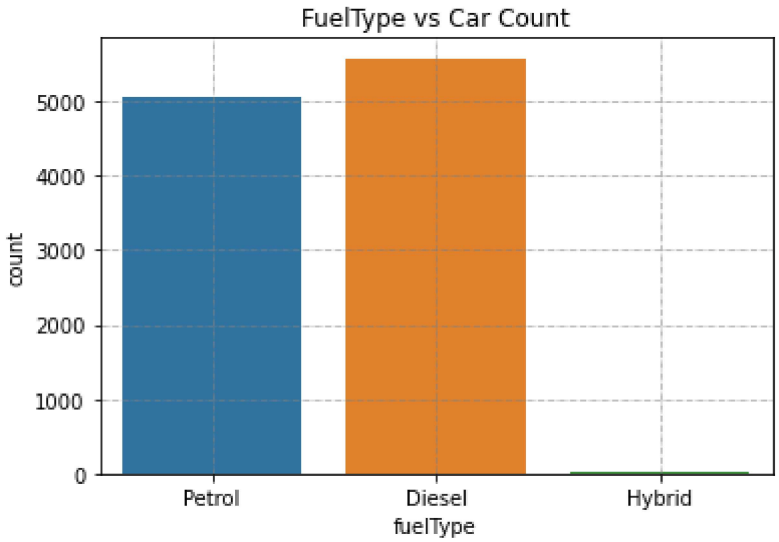
Above shows most people buy manual transmission, follow by semi-auto and then automatic transmission

In [15]:
```
audi_sql['fuelType'].value_counts()
```

Out[15]:
```
Diesel    5577
Petrol    5063
Hybrid      28
Name: fuelType, dtype: int64
```

In [16]:
```
sns.countplot(x = audi_sql["fuelType"])
plt.grid(color='grey', linestyle='-.', linewidth=0.5)
plt.title("FuelType vs Car Count")
```

Out[16]:
```
Text(0.5, 1.0, 'FuelType vs Car Count')
```



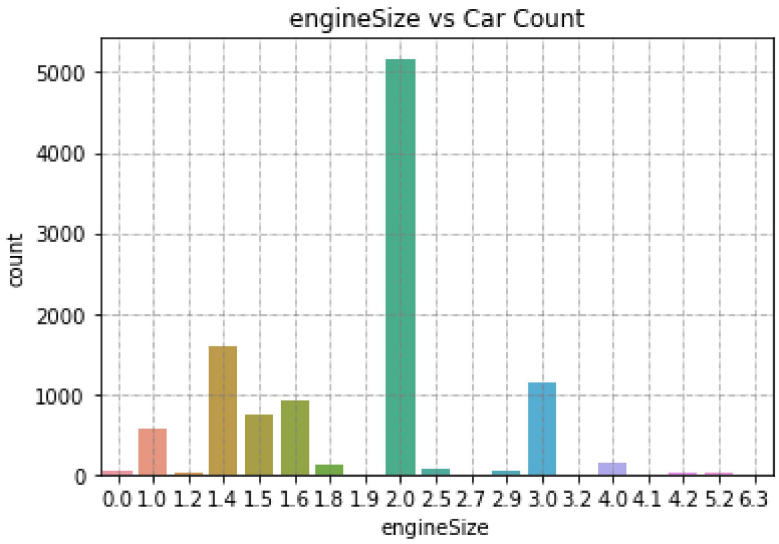
In [17]:
```
audi_sql['engineSize'].value_counts()
```

Out[17]:
```
2.0    5169
1.4    1594
3.0    1149
1.6     913
1.5     744
1.0     558
```

```
4.0     154
1.8     126
2.5      61
0.0      57
2.9      49
1.2      31
4.2      25
5.2      23
3.2       5
1.9       4
2.7       3
4.1       2
6.3       1
Name: engineSize, dtype: int64
```

In [18]:
```
sns.countplot(x = audi_sql["engineSize"])
plt.grid(color='grey', linestyle='-.', linewidth=0.5)
plt.title("engineSize vs Car Count")
```

Out[18]:
```
Text(0.5, 1.0, 'engineSize vs Car Count')
```



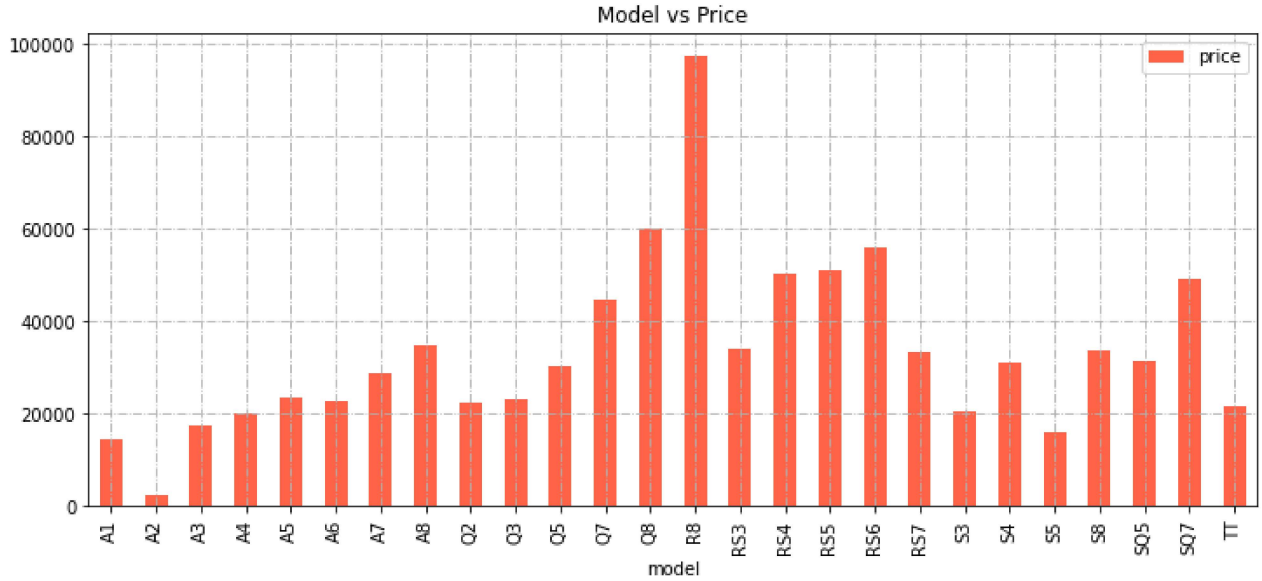
Conclusion of the above features :

1. More manual cars than semi-auto and auto being the least car sold
2. Maximum cars are of diesel and petrol fuelType
3. EngineSize 2.0 is the dominant

4. Analysis with respect to Price

Audi's Model vs Price

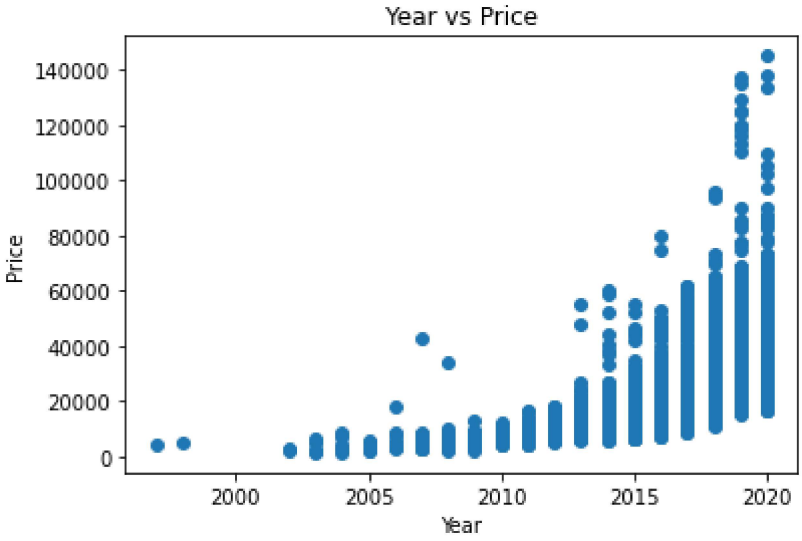
In [19]:
```
ModelvsPrice = pd.DataFrame(audi_sql.groupby('model')['price'].mean())
ModelvsPrice.plot.bar(color='tomato',figsize=(12,5))
plt.grid(linestyle='-.')
plt.title("Model vs Price")
plt.show()
```

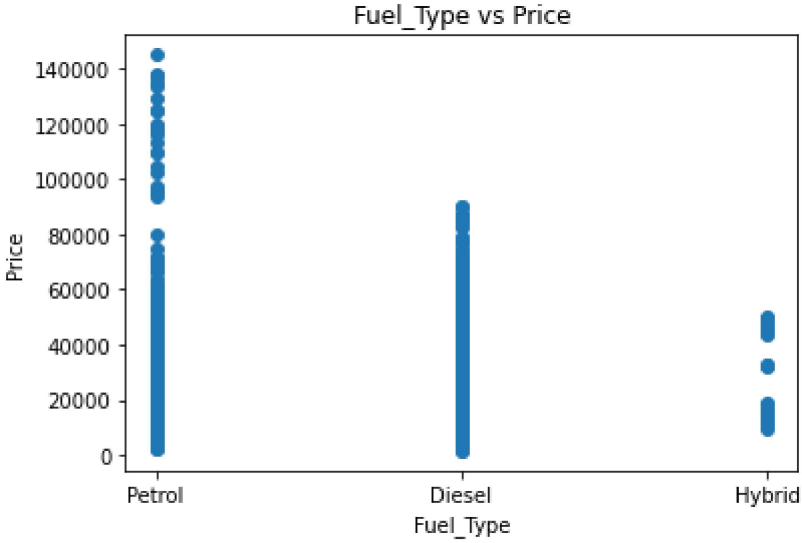


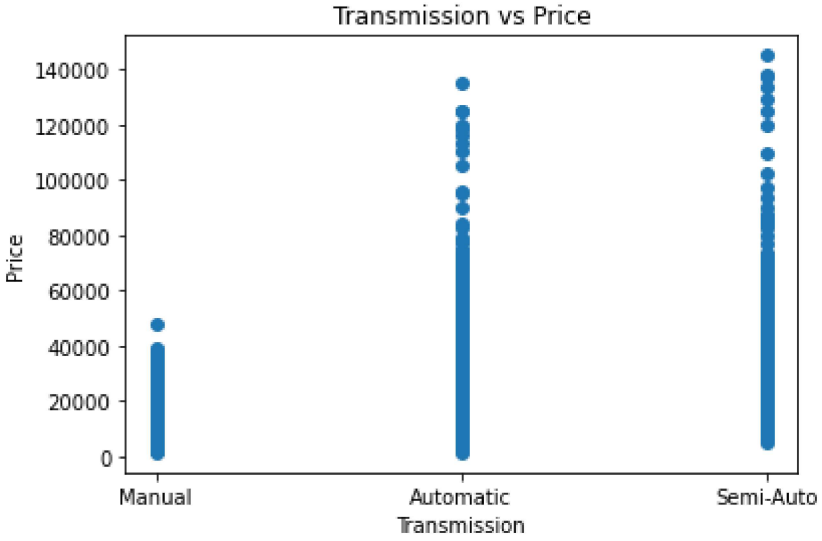
From above plot, it shows R8 being the most expensive car in the Audi series.

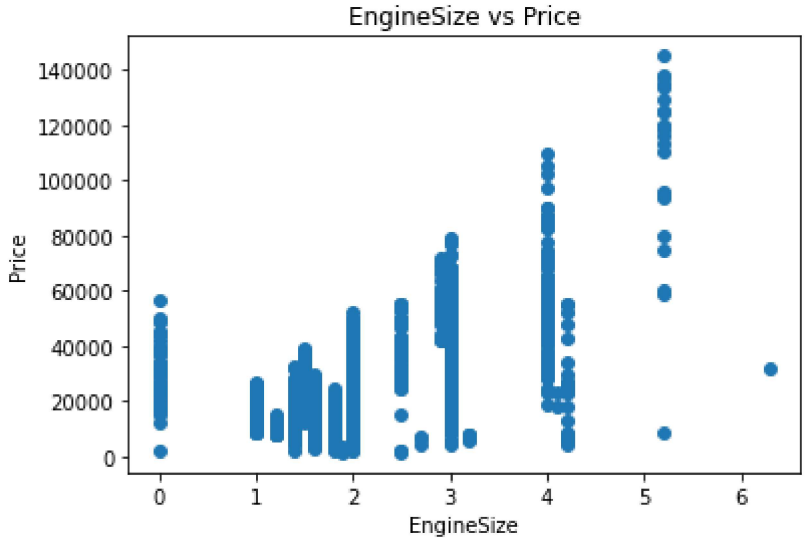
Analysis of other features with respect to Price:

In [20]:
```
#year vs price
plt.title("Year vs Price")
plt.xlabel("Year")
plt.ylabel("Price")
plt.scatter(audi_sql.year,audi_sql.price)
plt.show()

#fuel type vs price
plt.title("Fuel_Type vs Price")
plt.xlabel("Fuel_Type")
plt.ylabel("Price")
plt.scatter(audi_sql.fuelType,audi_sql.price)
plt.show()

#transmission vs price
plt.title("Transmission vs Price")
plt.xlabel("Transmission")
plt.ylabel("Price")
plt.scatter(audi_sql.transmission,audi_sql.price)
plt.show()

#engineSize vs price
plt.title("EngineSize vs Price")
plt.xlabel("EngineSize")
plt.ylabel("Price")
plt.scatter(audi_sql.engineSize,audi_sql.price)
plt.show()
```


Year vs Price
Price
Year
0
20000
40000
60000
80000
100000
120000
140000
2000
2005
2010
2015
2020

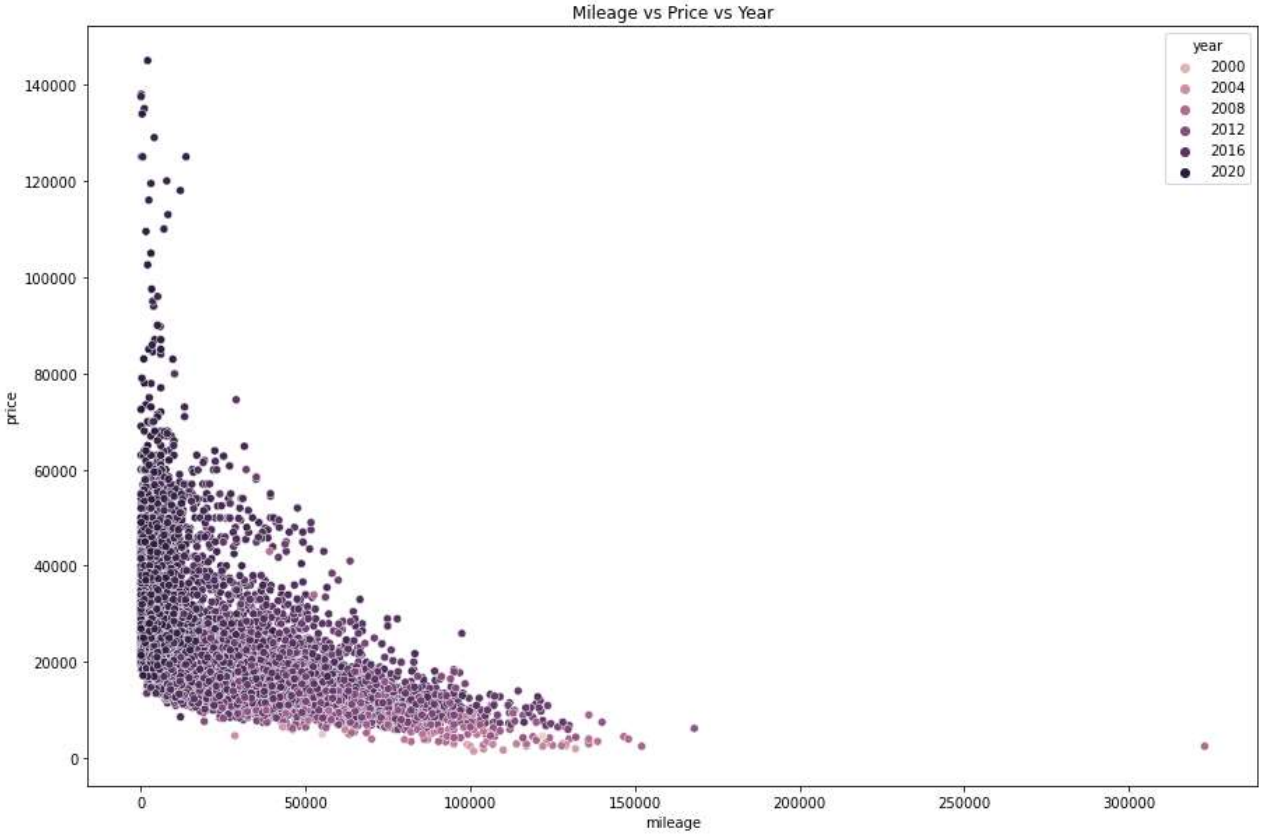

Fuel_Type vs Price
Price
Fuel_Type
0
20000
40000
60000
80000
100000
120000
140000
Petrol
Diesel
Hybrid


Transmission vs Price
Price
Transmission
0
20000
40000
60000
80000
100000
120000
140000
Manual
Automatic
Semi-Auto



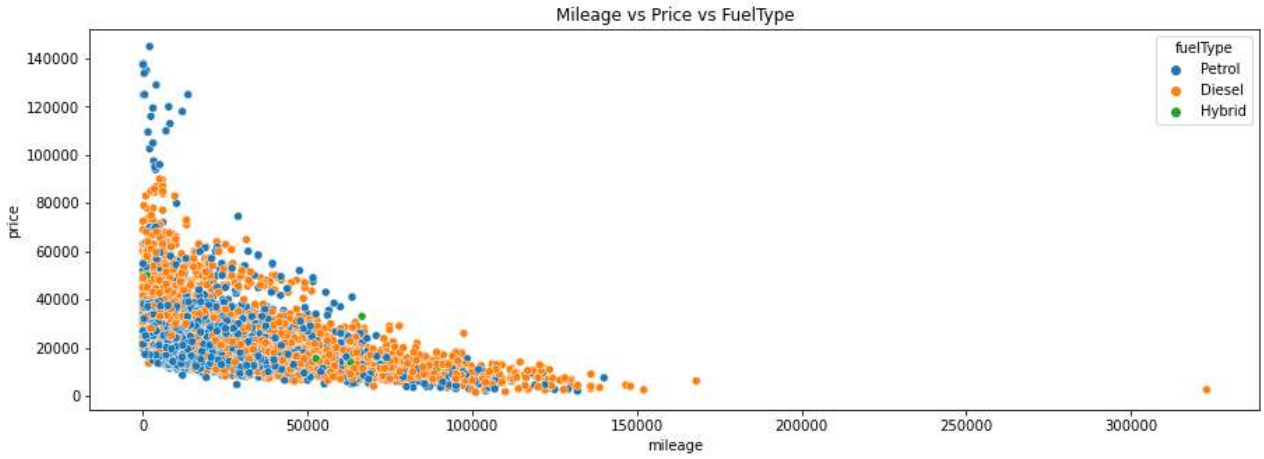
In [21]:
```
plt.figure(figsize=(15,10),facecolor='w')
sns.scatterplot(x=audi_sql["mileage"], y=audi_sql["price"], hue = audi_sql["year"])
plt.title("Mileage vs Price vs Year")
```

Out[21]: Text(0.5, 1.0, 'Mileage vs Price vs Year')



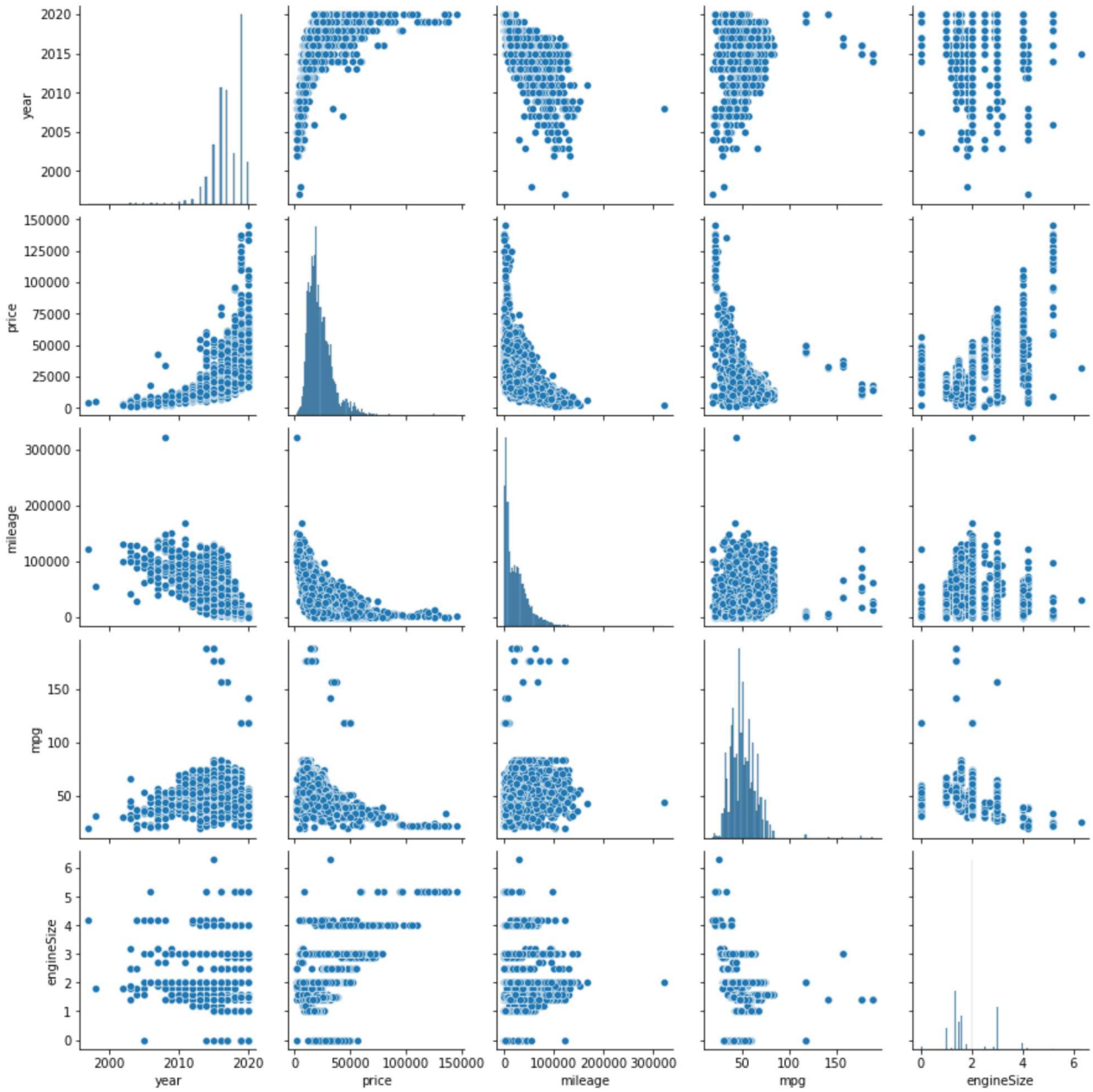
In [22]:
```
plt.figure(figsize=(15,5),facecolor='w')
sns.scatterplot(x=audi_sql["mileage"], y=audi_sql["price"], hue = audi_sql["fuelType"])
plt.title("Mileage vs Price vs FuelType")
```

Out[22]: Text(0.5, 1.0, 'Mileage vs Price vs FuelType')



In [23]:
```
sns.pairplot(audi_sql)
```

Out[23]: <seaborn.axisgrid.PairGrid at 0x1e74dcf7ac0>



Conclusion of the above features :

1. Cars ranging between year s 2018 to 2020 cost higher
2. Petrol cars more costly
3. Semi-auto cost higher than auto
4. EngineSize affect the price of a car
5. The highest engineSize of 6.3 seem to be the outlier and to remove

Step 2 :

Data Cleaning - Exploring Data

Cleaning Data (dataset Quality)

In [24]:
```
#removing outlier
audi_sql.drop(audi_sql[audi_sql['engineSize'] >= 6.3].index, axis=0, inplace=True)
```

In [25]:
```
audi_sql['engineSize'].value_counts() # outlier removed
```

Out[25]:
```
2.0    5169
1.4    1594
3.0    1149
1.6     913
1.5     744
1.0     558
4.0     154
1.8     126
2.5      61
0.0      57
2.9      49
1.2      31
4.2      25
5.2      23
3.2       5
1.9       4
2.7       3
4.1       2
Name: engineSize, dtype: int64
```

In [26]:
```
audi_sql.describe()
```

Out[26]:

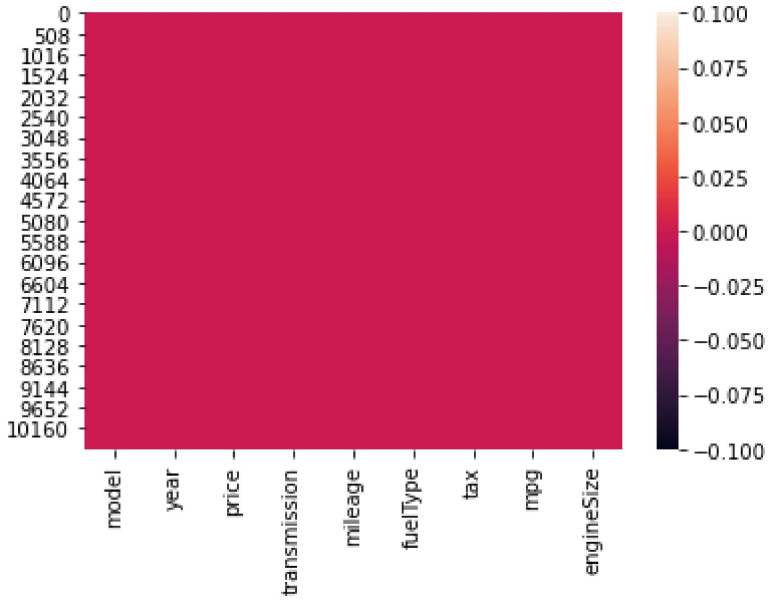
	year	price	mileage	mpg	engineSize
count	10667.000000	10667.00000	10667.000000	10667.000000	10667.000000
mean	2017.100872	22895.83163	24826.730383	50.772438	1.930299
std	2.167500	11715.05941	23506.299190	12.947984	0.601499
min	1997.000000	1490.00000	1.000000	18.900000	0.000000
25%	2016.000000	15120.50000	5964.500000	40.900000	1.500000
50%	2017.000000	20200.00000	19000.000000	49.600000	2.000000
75%	2019.000000	27990.00000	36467.000000	58.900000	2.000000
max	2020.000000	145000.00000	323000.000000	188.300000	5.200000

Check for non-zero missing or null values

In [27]:
```
#To show only feature that have non-zero missing or null values
# df_na[df_na!=0]

audi_sql_na = audi_sql.isna().sum()
audi_sql_na
```

Out[27]:
```
model           0
year            0
price           0
transmission    0
mileage         0
fuelType        0
tax             0
mpg             0
engineSize      0
dtype: int64
```

In [28]:
```
sns.heatmap(audi_sql.isnull())
plt.show()
```



Step 3 :

Data Preprocessing / Feature Engineering

Feature Engineering is the way of extracting features from data and transforming them into formats that are suitable for Machine Learning algorithms.

The models take features as input. A feature is generally a numeric representation of an aspect of real-world phenomena or data.

Numeric Columns

In [29]:
```
# limit to numerical data using df.select_dtypes()
```

```
# count number of items on each unique numberical feature

audi_sql_num = audi_sql.select_dtypes(include = ['number'])
audi_sql_num.nunique()
```

Out[29]:
```
year            21
price         3260
mileage       7724
mpg            103
engineSize      18
dtype: int64
```

In [30]:
```
audi_sql['year'].unique()
```

Out[30]:
```
array([2017, 2016, 2019, 2015, 2014, 2018, 2013, 2020, 2004, 2009, 2012,
       2010, 2007, 2011, 2008, 2003, 2005, 2002, 2006, 1998, 1997],
      dtype=int64)
```

In [31]:
```
audi_sql['price'].unique()
```

Out[31]:
```
array([12500, 16500, 11000, ..., 21291, 12380,  3750], dtype=int64)
```

In [32]:
```
audi_sql['mileage'].unique()
```

Out[32]:
```
array([15735, 36203, 29946, ...,  4018,  1978,  8646], dtype=int64)
```

In [33]:
```
audi_sql['mpg'].unique()
```

Out[33]:
```
array([ 55.4,  64.2,  67.3,  49.6,  58.9,  61.4,  70.6,  60.1,  57.6,
        52.3,  53.3,  56.5,  47.1,  47.9,  62.8,  76.3,  51.4,  65.7,
        68.9,  50.4,  72.4,  38.7,  74.3,  42.8,  83.1,  57.7,  54.3,
        40.4,  44.1,  33.2,  42.2,  44.8,  48.7,  37.7,  35.8,  40.9,
        45.6,  34. ,  29.4,  39.8,  39.2,  41.5,  38.2,  32.1,  30.7,
        34.9,  43.5,  36.2,  30.1,  32.8,  29.7,  26.4,  33.6,  34.5,
        46.3,  37.2,  80.7,  28.8,  31.7,  32.5,  31.4, 141.3,  31. ,
        35.3,  36.7, 117.7,  29.1,  21.4,  19.6,  26.9,  21.1,  76.4,
       156.9, 188.3,  24.1,  22.1,  21.6,  23. , 176.6,  22.8,  22.4,
        22.6,  24.8,  21.9,  38.1,  42.1,  50. ,  19.3,  28.5,  21. ,
        28.2,  30.4,  43.4,  37.6,  20.3,  30.8,  51.3,  27.4,  33.3,
        27.2,  34.4,  18.9,  44.5])
```

In [34]:
```
audi_sql['engineSize'].unique()
```

Out[34]:
```
array([1.4, 2. , 1. , 3. , 1.6, 1.8, 1.5, 4. , 2.5, 1.2, 4.2, 2.9, 5.2,
       1.9, 2.7, 0. , 3.2, 4.1])
```

Handling Categorical Columns

Machine Learning algorithms work with a numeric value

In [35]:
```
# limit to categorical data using df.select_dtypes()
# count number of items on each unique categories feature

audi_sql_cat1 = audi_sql.select_dtypes(include=['object'])  #data label
audi_sql_cat1.nunique()
```

Out[35]:
```
model           26
transmission     3
fuelType         3
```

```
tax             37
dtype: int64
```

model, transmission, fuelType and tax are categorical columns

In [36]:
```
print(audi_sql_cat1)
```

```
      model transmission fuelType  tax
0        A1       Manual   Petrol  150
1        A6    Automatic   Diesel   20
2        A1       Manual   Petrol   30
3        A4    Automatic   Diesel  145
4        A3       Manual   Petrol  145
...     ...          ...      ...  ...
10663    A3       Manual   Petrol  145
10664    A3       Manual   Petrol  150
10665    A3       Manual   Petrol  150
10666    Q3    Automatic   Petrol  150
10667    Q3       Manual   Petrol  150

[10667 rows x 4 columns]
```

==========

Create new dataframe to handle categorical value

==> audi_sql_cat2_n

In [37]:
```
#creating a new dataframe to handle categorical value
audi_sql_cat2_n = audi_sql.copy()
```

Using LabelEncoder to handle categorial columns

In label encoding in Python, we replace the categorical value with a numeric value between 0 and the number of classes minus 1. If the categorical variable value contains 5 distinct classes, we use (0, 1, 2, 3, and 4).

In [38]:
```
from sklearn.preprocessing import LabelEncoder

le_model=LabelEncoder()
le_transmission=LabelEncoder()
le_fuelType=LabelEncoder()
le_tax=LabelEncoder()
audi_sql_cat2_n['model2_n']= le_model.fit_transform(audi_sql_cat2_n['model'])
audi_sql_cat2_n['transmission2_n']=le_transmission.fit_transform(audi_sql_cat2_n['trans
audi_sql_cat2_n['fuelType2_n']=le_fuelType.fit_transform(audi_sql_cat2_n['fuelType'])
audi_sql_cat2_n['tax2_n']=le_tax.fit_transform(audi_sql_cat2_n['tax'])
```

In [39]:
```
audi_sql_cat2_n.head(5)
```

Out[39]:

	model	year	price	transmission	mileage	fuelType	tax	mpg	engineSize	model2_n	transmissio
0	A1	2017	12500	Manual	15735	Petrol	150	55.4	1.4	0	
1	A6	2016	16500	Automatic	36203	Diesel	20	64.2	2.0	5	
2	A1	2016	11000	Manual	29946	Petrol	30	55.4	1.4	0	

	model	year	price	transmission	mileage	fuelType	tax	mpg	engineSize	model2_n	transmissio
3	A4	2017	16800	Automatic	25952	Diesel	145	67.3	2.0	3	
4	A3	2019	17300	Manual	1998	Petrol	145	49.6	1.0	2	

NOTE : 4 new columns created with numeric value

In [40]:
```
audi_sql['model'].unique()    #old categorial column
```

Out[40]:
```
array(['A1', 'A6', 'A4', 'A3', 'Q3', 'Q5', 'A5', 'S4', 'Q2', 'A7', 'TT',
       'Q7', 'RS6', 'RS3', 'A8', 'Q8', 'RS4', 'RS5', 'R8', 'SQ5', 'S8',
       'SQ7', 'S3', 'S5', 'A2', 'RS7'], dtype=object)
```

In [41]:
```
audi_sql_cat2_n['model2_n'].unique()   #new binary column encoded
```

Out[41]:
```
array([ 0,  5,  3,  2,  9, 10,  4, 20,  8,  6, 25, 11, 17, 14,  7, 12, 15,
       16, 13, 23, 22, 24, 19, 21,  1, 18])
```

Above categorial value being encoded to boolean

Reshuffle columns & Remove unwanted columns

In [42]:
```
audi_sql_cat2_n.info()  #encoded NEW columns added
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 10667 entries, 0 to 10667
Data columns (total 13 columns):
 #   Column           Non-Null Count  Dtype  
---  ------           --------------  -----  
 0   model            10667 non-null  object 
 1   year             10667 non-null  int64  
 2   price            10667 non-null  int64  
 3   transmission     10667 non-null  object 
 4   mileage          10667 non-null  int64  
 5   fuelType         10667 non-null  object 
 6   tax              10667 non-null  object 
 7   mpg              10667 non-null  float64
 8   engineSize       10667 non-null  float64
 9   model2_n         10667 non-null  int32  
 10  transmission2_n  10667 non-null  int32  
 11  fuelType2_n      10667 non-null  int32  
 12  tax2_n           10667 non-null  int32  
dtypes: float64(2), int32(4), int64(3), object(4)
memory usage: 1000.0+ KB
```

Dropping columns with dtype = object (Removed)

In [43]:
```
audi_sql_cat2_n=audi_sql_cat2_n.drop(["model","transmission","fuelType","tax"],axis='co
```

Reshuffle the columns (Rearrange)

In [44]:
```
audi_sql_cat2_n=audi_sql_cat2_n[['model2_n','year', 'price', 'transmission2_n', 'mileag
```

In [45]:
```
audi_sql_cat2_n.info()   #final new dataframe created for Grid Search
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 10667 entries, 0 to 10667
Data columns (total 9 columns):
 #   Column           Non-Null Count  Dtype  
---  ------           --------------  -----  
 0   model2_n         10667 non-null  int32  
 1   year             10667 non-null  int64  
 2   price            10667 non-null  int64  
 3   transmission2_n  10667 non-null  int32  
 4   mileage          10667 non-null  int64  
 5   fuelType2_n      10667 non-null  int32  
 6   tax2_n           10667 non-null  int32  
 7   mpg              10667 non-null  float64
 8   engineSize       10667 non-null  float64
dtypes: float64(2), int32(4), int64(3)
memory usage: 666.7 KB
```

In [46]:
```
audi_sql_cat2_n.head(5)
```

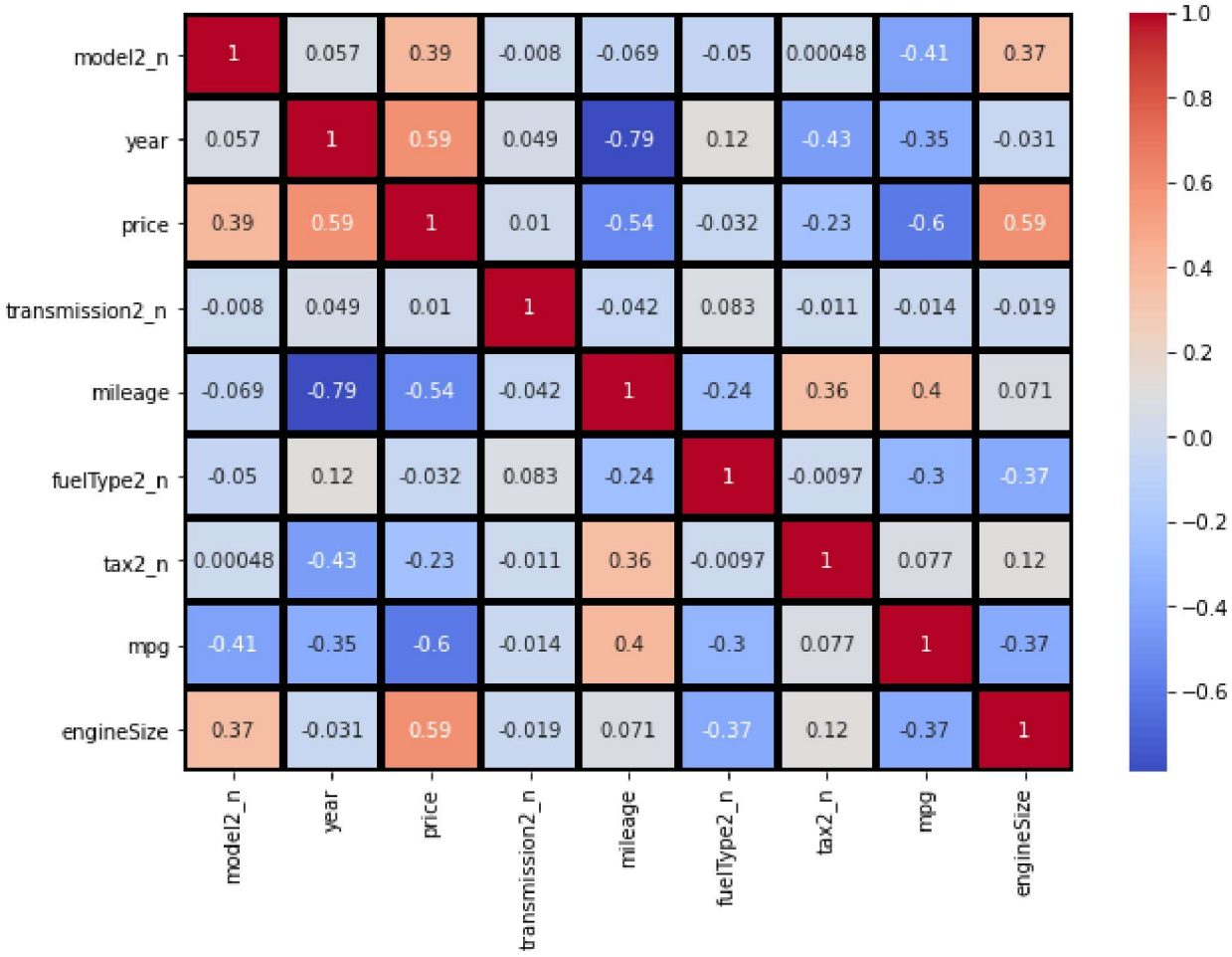
Out[46]:

	model2_n	year	price	transmission2_n	mileage	fuelType2_n	tax2_n	mpg	engineSize
0	0	2017	12500	1	15735	2	6	55.4	1.4
1	5	2016	16500	0	36203	0	12	64.2	2.0
2	0	2016	11000	1	29946	2	24	55.4	1.4
3	3	2017	16800	0	25952	0	5	67.3	2.0
4	2	2019	17300	1	1998	2	5	49.6	1.0

Correlation matrix - Heatmap Visualization

In [47]:
```
# look at correlations in the numerical independent variables, as well as the dependent
# audi_sql_num.corr()
```

In [48]:
```
# correlation matrix heatmap visualization
corrMatrix = audi_sql_cat2_n.corr()
plt.figure(figsize=(10,7))
sns.heatmap(corrMatrix, annot=True, cmap='coolwarm', linewidths=3, linecolor='black')
plt.show()
```



To determining important features that have strong relationship with the target(price) by identifying high correlation values (both positives and negatives)

In [49]:
```
# audi_sql.groupby(by='price').count().sort_values('price', ascending=True).head(10)
audi_sql_cat2_n.groupby(by='price').count().sort_values('price', ascending=True).head(1
```

Out[49]:

price	model2_n	year	transmission2_n	mileage	fuelType2_n	tax2_n	mpg	engineSize
1490	1	1	1	1	1	1	1	1
1699	1	1	1	1	1	1	1	1
1975	1	1	1	1	1	1	1	1
1990	1	1	1	1	1	1	1	1
2490	3	3	3	3	3	3	3	3
2495	3	3	3	3	3	3	3	3
2600	1	1	1	1	1	1	1	1
2675	1	1	1	1	1	1	1	1
2795	1	1	1	1	1	1	1	1
2876	1	1	1	1	1	1	1	1

In [50]:
```
audi_sql_cat2_n['price'].nlargest() #view the max price
                                    #view max = maybe the outliers
```

Out[50]:
```
4783    145000
2255    137995
4179    137500
3367    135000
5459    133900
Name: price, dtype: int64
```

In [51]:
```
audi_sql_cat2_n['price'].nsmallest()   #view the min price
```

Out[51]:
```
10588    1490
10552    1699
7795     1975
10108    1990
7404     2490
Name: price, dtype: int64
```

In [52]:
```
audi_sql_cat2_n.describe()
```

Out[52]:

	model2_n	year	price	transmission2_n	mileage	fuelType2_n	t
count	10667.000000	10667.000000	10667.00000	10667.000000	10667.000000	10667.000000	10667.0
mean	5.835193	2017.100872	22895.83163	1.082872	24826.730383	0.951720	8.4
std	5.194134	2.167500	11715.05941	0.763941	23506.299190	0.997566	7.0
min	0.000000	1997.000000	1490.00000	0.000000	1.000000	0.000000	0.0
25%	2.000000	2016.000000	15120.50000	0.000000	5964.500000	0.000000	5.0
50%	4.000000	2017.000000	20200.00000	1.000000	19000.000000	0.000000	5.0
75%	9.000000	2019.000000	27990.00000	2.000000	36467.000000	2.000000	12.0
max	25.000000	2020.000000	145000.00000	2.000000	323000.000000	2.000000	36.0

Create 2 new Dataframe from 'audi_sql_cat2_n'

For Grid Search testing & training

1. audi_sql_cat2_inputs : all features except price
2. audi_sql_cat2_target : Price (value that need to predict i.e price)

Table BEFORE dropping "price" column

In [53]:
```
audi_sql_cat2_n.head(5)   # BEFORE dropping "price"
```

Out[53]:

	model2_n	year	price	transmission2_n	mileage	fuelType2_n	tax2_n	mpg	engineSize
0	0	2017	12500	1	15735	2	6	55.4	1.4
1	5	2016	16500	0	36203	0	12	64.2	2.0
2	0	2016	11000	1	29946	2	24	55.4	1.4

	model2_n	year	price	transmission2_n	mileage	fuelType2_n	tax2_n	mpg	engineSize
3	3	2017	16800	0	25952	0	5	67.3	2.0
4	2	2019	17300	1	1998	2	5	49.6	1.0

In [54]:
```
audi_sql_cat2_inputs=audi_sql_cat2_n.drop(['price'], axis='columns')
audi_sql_cat2_target=audi_sql_cat2_n['price']
```

Table AFTER dropping "price" column

In [55]:
```
audi_sql_cat2_inputs.head(5)   # AFTER dropped "price"
```

Out[55]:

	model2_n	year	transmission2_n	mileage	fuelType2_n	tax2_n	mpg	engineSize
0	0	2017	1	15735	2	6	55.4	1.4
1	5	2016	0	36203	0	12	64.2	2.0
2	0	2016	1	29946	2	24	55.4	1.4
3	3	2017	0	25952	0	5	67.3	2.0
4	2	2019	1	1998	2	5	49.6	1.0

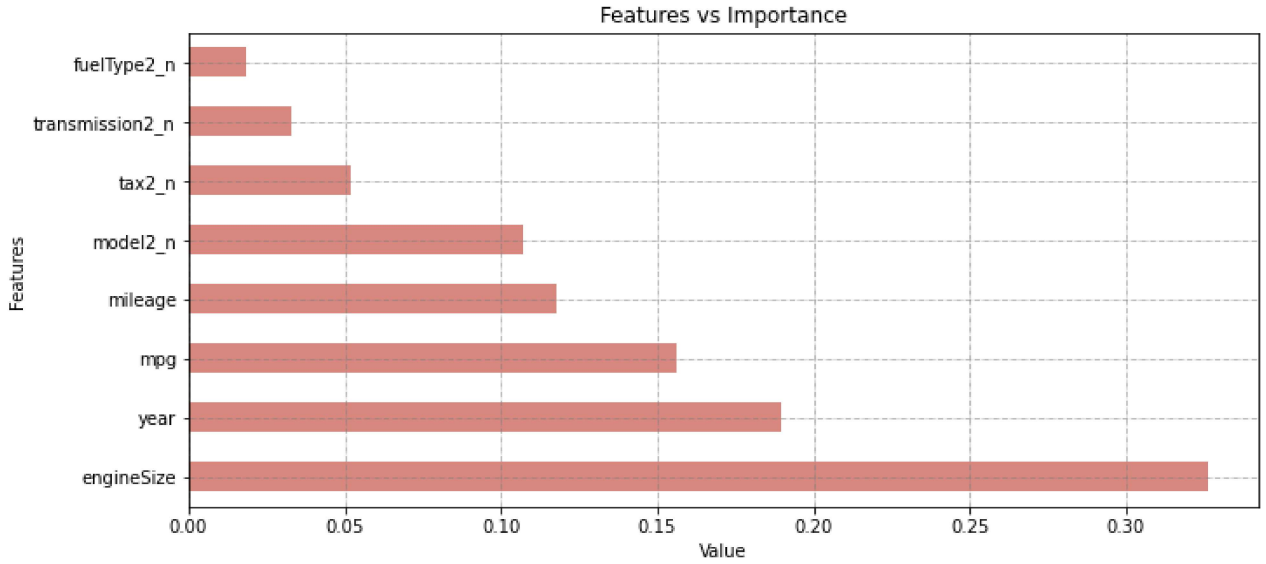
In [56]:
```
audi_sql_cat2_target.head(5)
```

Out[56]:
```
0    12500
1    16500
2    11000
3    16800
4    17300
Name: price, dtype: int64
```

Plot graph of feature importances for Better Visualization

To see which features are the most and least important for predicting the price of a used car:

In [57]:
```
from sklearn.ensemble import ExtraTreesRegressor
import matplotlib.pyplot as plt
model = ExtraTreesRegressor()
model.fit(audi_sql_cat2_inputs,audi_sql_cat2_target)

#use inbuilt class feature_importances of ExtraTreeRegressor
#plot graph of feature importances for better visualization

feat_importances = pd.Series(model.feature_importances_, index=audi_sql_cat2_inputs.col
plt.figure(figsize=(11,5))
plt.xlabel("Value")
plt.ylabel("Features")
plt.title("Features vs Importance")
plt.grid(feat_importances.nlargest(10).plot(kind='barh',color='#D98880'))##45B39D
plt.grid(color='grey', linestyle='-.', linewidth=0.5)
plt.show()
```



It seem engineSize is the most important and fuelType being the least important features for predicting the price of an Audi's used car

Step 4

Applying different models on the data

Use machine learning model hyperparameter optimization technique called as Grid Search for BEST estimator BEST score!

How to evaluate how well a model generalizes, we take the next step to improve the model's generalization performance by tuning its parameters.

It is important to understand what the parameters mean before trying to adjust them. Finding the values of the important parameters of a model (the ones that provide the best generalization performance) is a tricky task, but necessary for almost all models and datasets.

Because it is such a common task, there are standard methods in scikit-learn to help you with it. The most commonly used method is Grid Search, which basically means trying all possible combinations of the parameters of interest.

In [58]:
```
#Import libraries for training and testing of different models(estimators)

from sklearn import linear_model
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor
```

Dataframe used for test & train

1. audi_sql_cat2_inputs : all features except price
2. audi_sql_cat2_target : Price

In [59]:
```
#GridSearchCV is a library function that is a member of sklearn's model_selection packa
#It helps to loop through pre-defined hyperparameters and fit your estimator (model) on
# So, in the end, you can select the best parameters from the listed hyperparameters.

#grid search, an effective method for adjusting the parameters in supervised models for

#The grid search is implemented in Python Sklearn using the class, GridSearchCV that im


from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import ShuffleSplit
def find_best_model_using_gridsearchcv(audi_sql_cat2_inputs,audi_sql_cat2_target):
    algos = {
        'linear_regression' : {
            'model': linear_model.LinearRegression(),
            'params': {
                'normalize': [True, False]
            }
        },
       # 'lasso': {
       #     'model': Lasso(),
       #     'params': {
       #         'alpha': [1,2],
       #         'selection': ['random', 'cyclic']
       #     }
       # },
        'decision_tree': {
            'model': DecisionTreeRegressor(),
            'params': {
                'criterion' : ['mse','friedman_mse'],
                'splitter': ['best','random']
            }
        },
        'Random_forest': {
            'model': RandomForestRegressor(),
            'params': {
                'max_features': ['auto', 'sqrt'],
                'n_estimators': [50,60]
           }}}

    scores = []
    cv = ShuffleSplit(n_splits=5, test_size=0.2, random_state=0)
    for algo_name, config in algos.items():
        gs =  GridSearchCV(config['model'], config['params'], cv=cv, return_train_score
        gs.fit(audi_sql_cat2_inputs,audi_sql_cat2_target)
        scores.append({
            'model': algo_name,
            'best_score': gs.best_score_,
            'best_params': gs.best_params_
        })

    return pd.DataFrame(scores,columns=['model','best_score','best_params'])

result=find_best_model_using_gridsearchcv(audi_sql_cat2_inputs,audi_sql_cat2_target)
result
```

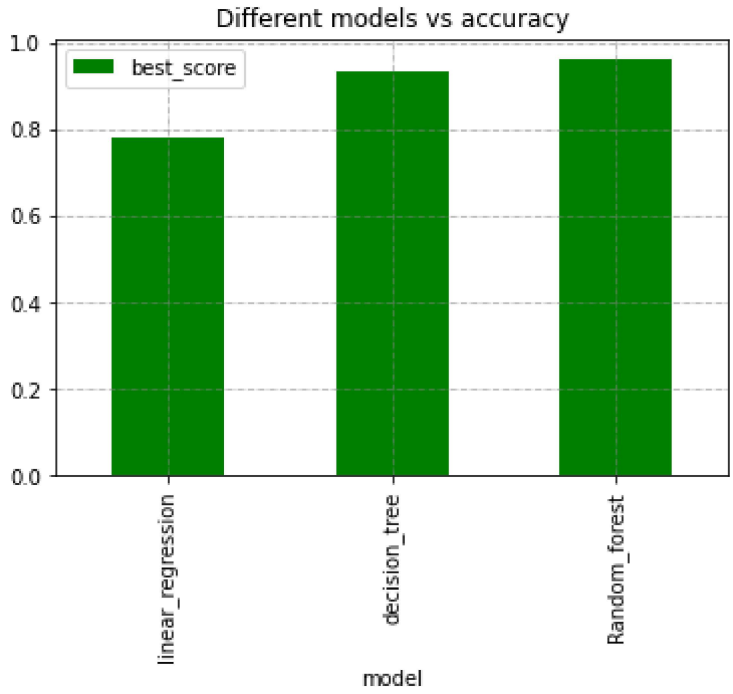
Out[59]:

	model	best_score	best_params
0	linear_regression	0.781948	{'normalize': False}
1	decision_tree	0.933965	{'criterion': 'mse', 'splitter': 'random'}

	model	best_score	best_params
2	Random_forest	0.961832	{'max_features': 'auto', 'n_estimators': 50}

Let see RESULT graphically :

In [60]:
```
result_map=result.drop(['best_params'],axis='columns')
result_map.plot(x="model",y='best_score',kind='bar',color='green',title='Different mode
plt.grid(color='grey', linestyle='-.', linewidth=0.5)
```



Random Forest to use since it has the highest accurecy of 96% to train our model

NOTE : Optimization in used in supervised Machine Learning

Here we have a model that initially set certain random values for it's parameter (more popularly known as weights). These parameter helps to build a function. For e.g. y=w0x0+w1x1+w2x2, where x0,x1,x2 are features (think study, play, social media in above example) and w0,w1,w2 are weights (think each of them as time given to study, play, social media in above example). y is the output or prediction (think as exam score in above example)

This function is used to make prediction on training data set. The prediction is then compared with the actual results of training set. Both predicted output and actual output is send to an error function. This error function calculates the offset or error between the predicted and actual output. This error is sent to an optimizer. The optimizer calculates that how much the initial values of weights should be changed so that the error is reduced further and we move towards expected output.

The weights of the model are adjusted accordingly for next iteration. And again predictions are made on training set, the error is calculated and optimizer again recommends for weight adjustment.

These iteration should keeps on going till there are not much changes in the error or we have reached desired goal in terms of prediction accuracy. At this point the iteration should be stopped.

==========

Random Forest Model

Random Forest used since it has the highest accurecy of 96% when training our model

Again, dataframe used for test & train

1. audi_sql_cat2_inputs : all features except price
2. audi_sql_cat2_target : Price

Split data frame for training on the random forest model

In [61]:
```
from sklearn.model_selection import train_test_split
import time; # This is required to include time module.

X_train, X_test, y_train, y_test = train_test_split(audi_sql_cat2_inputs, audi_sql_cat2
```

In [62]:
```
RF_len_train = len(X_train)
RF_len_train
```

Out[62]:
```
8533
```

In [63]:
```
RF_len_test = len(X_test)
RF_len_test
```

Out[63]:
```
2134
```

Training our model with BEST paraemters score

In [64]:
```
Model_RandomForest = RandomForestRegressor(max_features='sqrt', bootstrap='True')
```

In [65]:
```
Model_RandomForest
```

Out[65]:
```
RandomForestRegressor(bootstrap='True', max_features='sqrt')
```

Time taken for RF Training

In [66]:
```
# Including RF Training time measement
training_time_RF = time.time()

Model_RandomForest.fit(X_train, y_train)
training_time_RF = time.time() - training_time_RF

print('Time in ms : ', training_time_RF)
```

```
Time in ms :  1.0342702865600586
```

Random Forest Test & Train Score

In [67]:
```
Model_RandomForest.score(X_test, y_test)
```

Out[67]:
```
0.9649539455087978
```

In [68]:
```
Model_RandomForest.score(X_train, y_train)
```

Out[68]:
```
0.9936360672955789
```

In [69]:
```
y_pred_RF = Model_RandomForest.predict(X_test)
```

In [70]:
```
y_pred_RF
```

Out[70]:
```
array([22133.54166667, 19404.57      , 20068.33220238, ...,
       23138.64      , 22173.9       , 19856.13      ])
```

In [71]:
```
y_pred_RF_X_train = Model_RandomForest.predict(X_train)
```

In [72]:
```
y_pred_RF_X_train
```

Out[72]:
```
array([ 6892.75, 38890.58, 18208.34, ..., 24943.23, 17006.17, 33629.11])
```

Random Forest Training score and Test score

In [73]:
```
# Evaluate the model's training score and test score
print("Random Forest model's training score = {:.2f}".format(Model_RandomForest.score(X
print("Random Forest model's test score     = {:.2f}".format(Model_RandomForest.score(X
```

```
Random Forest model's training score = 0.99
Random Forest model's test score     = 0.96
```

Random Forest R2, MSE & MAE

In [74]:
```
from sklearn.metrics import mean_squared_error, r2_score, mean_absolute_error

# The mean squared error MSE
print('Mean squared error: {:.2f}'.format(mean_squared_error(y_test, y_pred_RF)))

# MSE (Mean Squared Error) represents the difference between the original and predicted
# extracted BY SQUARED the average difference over the data set.

# MSE is the average of the square of the errors.
# The larger the number the larger the error.
# Error in this case means the difference between the observed values y1, y2, y3, …
# and the predicted ones pred(y1), pred(y2), pred(y3)

# There is no correct value for MSE, simply put, the lower the value the better and 0 m



# The Mean absolute error MAE - To measure of model quality.
print('Mean absolute error (model predictions are off by approximately) : {:.2f}'.forma

# MAE (Mean absolute error) represents the difference between the original and predicte
# extracted BY AVERAGED the absolute difference over the number of data set.
```

```
# The coefficient of determination (r2_score) normally ranges from 0 to 1 and 1 being p
print('Coefficient of determination R2_score: {:.2f}'.format(r2_score(y_test, y_pred_RF

# Similarly, there is also no correct answer as to what R2 should be. 100% means perfec
# Yet, there are models with a low R2 that are still good models.
# Closely related to the MSE
```

```
Mean squared error: 5051577.40
Mean absolute error (model predictions are off by approximately) : 1531.26
Coefficient of determination R2_score: 0.96
```

Importing the model in a file

In [75]:
```
#pickle method
import pickle
```

In [76]:
```
#writing the model in a fle
pickle.dump(Model_RandomForest, open('RF_model.pkl', 'wb'))
```

In [77]:
```
#reading the file
RF_model=pickle.load(open('RF_model.pkl','rb'))
```

Testing the model

In [78]:
```
#RFmodel.predict(pd.DataFrame([['A3', 2017, 'Manual', 15000, 'Petrol', 150, 55.4, 1.4]]
#RFmodel.predict([['A3', 2017, 'Manual', 15000, 'Petrol', 150, 55.4, 1.4]])

RF_model.predict([[0, 2017, 1, 15000, 2, 6, 55.4, 1.4]])
```

Out[78]:
```
array([14045.1])
```

Testing the Model (Price prediction)

In [79]:
```
audi_sql_cat2_inputs.head(5)
```

Out[79]:

	model2_n	year	transmission2_n	mileage	fuelType2_n	tax2_n	mpg	engineSize
0	0	2017	1	15735	2	6	55.4	1.4
1	5	2016	0	36203	0	12	64.2	2.0
2	0	2016	1	29946	2	24	55.4	1.4
3	3	2017	0	25952	0	5	67.3	2.0
4	2	2019	1	1998	2	5	49.6	1.0

In [80]:
```
Model_RandomForest.predict([['0', 2017, '1', 15000, '2', '6', 55.4, 1.4]])
```

Out[80]:
```
array([14045.1])
```

In [81]:
```
list(np.array(le_model.__dict__['classes_']))
model_Dictionary = dict(zip( list(np.array(le_model.__dict__['classes_'])), le_model.tr
model_Dictionary
```

Out[81]:
```
{'A1': 0,
 'A2': 1,
 'A3': 2,
 'A4': 3,
 'A5': 4,
 'A6': 5,
 'A7': 6,
 'A8': 7,
 'Q2': 8,
 'Q3': 9,
 'Q5': 10,
 'Q7': 11,
 'Q8': 12,
 'R8': 13,
 'RS3': 14,
 'RS4': 15,
 'RS5': 16,
 'RS6': 17,
 'RS7': 18,
 'S3': 19,
 'S4': 20,
 'S5': 21,
 'S8': 22,
 'SQ5': 23,
 'SQ7': 24,
 'TT': 25}
```

In [82]:
```
list(np.array(le_transmission.__dict__['classes_']))
transmission_Dictionary = dict(zip( list(np.array(le_transmission.__dict__['classes_'])
transmission_Dictionary
```

Out[82]:
```
{'Automatic': 0, 'Manual': 1, 'Semi-Auto': 2}
```

In [83]:
```
list(np.array(le_fuelType.__dict__['classes_']))
fuelType_Dictionary = dict(zip( list(np.array(le_fuelType.__dict__['classes_'])), le_fu
fuelType_Dictionary
```

Out[83]:
```
{'Diesel': 0, 'Hybrid': 1, 'Petrol': 2}
```

In [84]:
```
list(np.array(le_tax.__dict__['classes_']))
tax_Dictionary = dict(zip( list(np.array(le_tax.__dict__['classes_'])), le_tax.transfor
tax_Dictionary
```

Out[84]:
```
{'0': 0,
 '115': 1,
 '125': 2,
 '135': 3,
 '140': 4,
 '145': 5,
 '150': 6,
 '155': 7,
 '160': 8,
 '165': 9,
 '190': 10,
 '195': 11,
 '20': 12,
 '200': 13,
 '205': 14,
 '220': 15,
 '230': 16,
 '235': 17,
 '240': 18,
 '260': 19,
```

```
 '265': 20,
 '280': 21,
 '290': 22,
 '295': 23,
 '30': 24,
 '300': 25,
 '305': 26,
 '315': 27,
 '325': 28,
 '330': 29,
 '515': 30,
 '535': 31,
 '540': 32,
 '555': 33,
 '565': 34,
 '570': 35,
 '580': 36}
```

In [85]:
```
Audi_model = model_Dictionary
print('model  Label_1')
for model, Label_1 in Audi_model.items():
    print('{0:10}  {1:2d}'.format(model, Label_1))
```

```
model  Label_1
A1           0
A2           1
A3           2
A4           3
A5           4
A6           5
A7           6
A8           7
Q2           8
Q3           9
Q5          10
Q7          11
Q8          12
R8          13
RS3         14
RS4         15
RS5         16
RS6         17
RS7         18
S3          19
S4          20
S5          21
S8          22
SQ5         23
SQ7         24
TT          25
```

In [86]:
```
transmission = transmission_Dictionary
print('transmission  Label_2')
for transmission, Label_2 in transmission.items():
    print('{0:10}  {1:2d}'.format(transmission, Label_2))
```

```
transmission  Label_2
Automatic     0
Manual        1
Semi-Auto     2
```

In [87]:
```
fuelType = fuelType_Dictionary
print('fuelType  Label_3')
```

```
for fuelType, Label_3 in fuelType.items():
    print('{0:10}  {1:2d}'.format(fuelType, Label_3))
```

```
fuelType  Label_3
Diesel       0
Hybrid       1
Petrol       2
```

In [88]:
```
tax = tax_Dictionary
print('tax  Label_4')
for tax, Label_4 in tax.items():
    print('{0:10}  {1:2d}'.format(tax, Label_4))
```

```
tax  Label_4
0            0
115          1
125          2
135          3
140          4
145          5
150          6
155          7
160          8
165          9
190         10
195         11
20          12
200         13
205         14
220         15
230         16
235         17
240         18
260         19
265         20
280         21
290         22
295         23
30          24
300         25
305         26
315         27
325         28
330         29
515         30
535         31
540         32
555         33
565         34
570         35
580         36
```

Compare 'Desired Output' vs 'Predicted Output'

In [89]:
```
compare_audi_sql_cat2_n = pd.DataFrame({"Desired Output (price)": y_test,
                                        "Predicted Output (price)": y_pred_RF})
```

In [90]:
```
compare_audi_sql_cat2_n[:5]
```

Out[90]:

	Desired Output (price)	Predicted Output (price)
2641	21491	22133.541667

	Desired Output (price)	Predicted Output (price)
10423	20690	19404.570000
3291	19752	20068.332202
5596	31950	31220.870000
10499	4999	6320.830000

==========

Save the cleaned dataset to a new file

In [91]:
```
# backup=audi_sql.copy()    # backup dataset
```

In [92]:
```
# audi_sql.to_csv('Cleaned car_audi.') # file for building website using Python
```

==========

Decision Tree Model

Again, dataframe used for test & train

1. audi_sql_cat2_inputs : all features except price
2. audi_sql_cat2_target : price

1. Dateframe ==> audi_sql_cat2_n
2. Orignal Dataframe ==> audi_sql

In [93]:
```
# audi_sql_cat2_inputs.head(5) #all features except price
# audi_sql_cat2_n.head(5) #all columns
```

In [94]:
```
# X = audi_sql_cat2_n.drop(['price'],axis=1)
# y = audi_sql_cat2_n['price']
```

In [95]:
```
audi_sql_cat2_inputs.head(5)   # with price column DROPPED
```

Out[95]:

	model2_n	year	transmission2_n	mileage	fuelType2_n	tax2_n	mpg	engineSize
0	0	2017	1	15735	2	6	55.4	1.4
1	5	2016	0	36203	0	12	64.2	2.0
2	0	2016	1	29946	2	24	55.4	1.4
3	3	2017	0	25952	0	5	67.3	2.0
4	2	2019	1	1998	2	5	49.6	1.0

Split data frame for training on Decision Tree Model

In [96]:
```
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
X_train, X_test, y_train, y_test = train_test_split(audi_sql_cat2_inputs, audi_sql_cat2
```

In [97]:
```
from sklearn.tree import DecisionTreeRegressor
DT_regressor = DecisionTreeRegressor(criterion='mse',splitter='best', random_state=0)
```

In [98]:
```
DT_regressor
```

Out[98]:
```
DecisionTreeRegressor(random_state=0)
```

Time taken for DT Training

In [99]:
```
# Including DT Training time measement
training_time_DT = time.time()

DT_regressor.fit(X_train, y_train)
training_time_DT = time.time() - training_time_DT

print('Time in Sec : ', training_time_DT)
```

```
Time in Sec :  0.0608525276184082
```

In [100…
```
#DT_regressor.fit(X_train,y_train)
```

Decision Tree Test Score

In [101…
```
DT_regressor.score(X_test, y_test)
```

Out[101…
```
0.9375285566569695
```

In [102…
```
# DT_regressor.score(X_train, y_train)
```

In [103…
```
y_pred_DT = DT_regressor.predict(X_test)
```

In [104…
```
y_pred_DT
```

Out[104…
```
array([19291., 18930., 21000., ..., 23490., 22382., 19575.])
```

In [105…
```
from sklearn.metrics import r2_score

r2_score(y_test, y_pred_DT)
```

Out[105…
```
0.9375285566569695
```

Decision Tree Training score and Test score

In [106…
```
# Evaluate the model's training score and test score
print("Random Forest model's training score = {:.2f}".format(DT_regressor.score(X_train
print("Random Forest model's test score     = {:.2f}".format(DT_regressor.score(X_test,
```

```
Random Forest model's training score = 1.00
Random Forest model's test score     = 0.94
```

Decision Tree R2, MSE & MAE

In [107…
```
from sklearn.metrics import mean_squared_error, r2_score, mean_absolute_error

# The mean squared error
print('Mean squared error: {:.2f}'.format(mean_squared_error(y_test, y_pred_DT)))
print('Mean absolute error: {:.2f}'.format(mean_absolute_error(y_test, y_pred_DT)))


# The coefficient of determination (r2_score): 1 is perfect prediction
print('Coefficient of determination R2_score: {:.2f}'.format(r2_score(y_test, y_pred_DT
```

```
Mean squared error: 9004703.54
Mean absolute error: 1967.19
Coefficient of determination R2_score: 0.94
```

Importing the model in a file

In [108…
```
#pickle method
import pickle
```

In [109…
```
#writing the model in a fle
pickle.dump(DT_regressor, open('DT_model.pkl', 'wb'))
```

In [110…
```
#reading the file
DT_model=pickle.load(open('DT_model.pkl','rb'))
```

Testing the Model (Price prediction)

In [111…
```
audi_sql_cat2_inputs.head(5)
```

Out[111…

	model2_n	year	transmission2_n	mileage	fuelType2_n	tax2_n	mpg	engineSize
0	0	2017	1	15735	2	6	55.4	1.4
1	5	2016	0	36203	0	12	64.2	2.0
2	0	2016	1	29946	2	24	55.4	1.4
3	3	2017	0	25952	0	5	67.3	2.0
4	2	2019	1	1998	2	5	49.6	1.0

In [112…
```
# DT_regressor.predict(pd.DataFrame([['A3', 2017, 'Manual', 15000, 'Petrol', 150, 55.4,

DT_model.predict([[0, 2017, 1, 15000, 2, 6, 55.4, 1.4]])
```

Out[112…
```
array([13990.])
```

In [113…
```
DT_regressor.predict([['0', 2017, '1', 15000, '2', '6', 55.4, 1.4]])
```

Out[113…
```
array([13990.])
```

Compare 'Desired Output' vs 'Predicted Output'

In [114…
```
compare_audi_sql_cat2_n = pd.DataFrame({"Desired Output (price)": y_test,
                                        "Predicted Output (price)": y_pred_DT})
```

In [115…
```
compare_audi_sql_cat2_n[:5]
```

Out[115…

	Desired Output (price)	Predicted Output (price)
2641	21491	19291.0
10423	20690	18930.0
3291	19752	21000.0
5596	31950	30796.0
10499	4999	5995.0

==========

Linear Regression Model

Again, dataframe used for test & train

1. audi_sql_cat2_inputs : all features except price
2. audi_sql_cat2_target : price

1. Dateframe ==> audi_sql_cat2_n
2. Orignal Dataframe ==> audi_sql

In [116…
```
audi_sql_cat2_inputs.head(5)   # with price column DROPPED
```

Out[116…

	model2_n	year	transmission2_n	mileage	fuelType2_n	tax2_n	mpg	engineSize
0	0	2017	1	15735	2	6	55.4	1.4
1	5	2016	0	36203	0	12	64.2	2.0
2	0	2016	1	29946	2	24	55.4	1.4
3	3	2017	0	25952	0	5	67.3	2.0
4	2	2019	1	1998	2	5	49.6	1.0

Split data frame for training on Linear Regression Model

In [117…
```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(audi_sql_cat2_inputs, audi_sql_cat2
```

In [118…
```
LR_pipe = Pipeline([('scaler', StandardScaler()), ('LinReg', LinearRegression())])
```

In [119…
```
LR_pipe
```

Out[119…
```
Pipeline(steps=[('scaler', StandardScaler()), ('LinReg', LinearRegression())])
```

Time taken for LR Training

In [120…
```
# Including LR Training time measement
training_time_LR = time.time()

LR_pipe.fit(X_train, y_train)
training_time_LR = time.time() - training_time_LR

print('Time in Sec : ', training_time_LR)
```

```
Time in Sec :  0.011980772018432617
```

In [121…
```
# LR_pipe.fit(X_train, y_train)
```

LinearRegression Test Score

In [122…
```
LR_pipe.score(X_test, y_test)
```

Out[122…
```
0.8034565659958568
```

In [123…
```
# LR_pipe.score(X_train, y_train)
```

In [124…
```
y_pred_LR = LR_pipe.predict(X_test)
```

In [125…
```
y_pred_LR
```

Out[125…
```
array([30833.65920843, 19120.4824019 , 18547.02223798, ...,
       26540.85199651, 23835.86481621, 21730.56737006])
```

In [126…
```
from sklearn.metrics import r2_score

r2_score(y_test, y_pred_LR)
```

Out[126…
```
0.8034565659958568
```

Linear Regression Training score and Test score

In [127…
```
# model's training score and test score
print("Linear Regression model's training score = {:.2f}".format(LR_pipe.score(X_train,
print("Linear Regression model's test score     = {:.2f}".format(LR_pipe.score(X_test,
```

```
Linear Regression model's training score = 0.78
Linear Regression model's test score     = 0.80
```

Linear Regression R2, MSE & MAE

In [128…
```
from sklearn.metrics import mean_squared_error, r2_score, mean_absolute_error

# The mean squared error
print('Mean squared error MSE: {:.2f}'.format(mean_squared_error(y_test, y_pred_LR)))
print('Mean absolute error MAE : {:.2f}'.format(mean_absolute_error(y_test, y_pred_LR))


# The coefficient of determination: 1 is perfect prediction
print('R2_score : {:.2f}'.format(r2_score(y_test, y_pred_LR)))
```

```
Mean squared error MSE: 28329989.86
Mean absolute error MAE : 3377.43
R2_score : 0.80
```

NOTE :

R-Squared (R^2 or the coefficient of determination) is a statistical measure in a regression model that determines the proportion of variance in the dependent variable that can be explained by the independent variable. In other words, r-squared shows how well the data fit the regression model (the goodness of fit).

R2 can take values from 0 to 1. A value of 1 indicates that the regression predictions perfectly fit the data. Results look like the model is not a good fit for car price prediction.

R-squared can take any values between 0 to 1, therefore it look like the Rando

Although the statistical measure provides some useful insights regarding the regression model, the user should not rely only on the measure in the assessment of a statistical model. The figure does not disclose information about the causation relationship between the independent and dependent variables.

In addition, it does not indicate the correctness of the regression model. Therefore, the user should always draw conclusions about the model by analyzing r-squared together with the other variables in a statistical model.

A high r-squared is good for the regression model however, sometimes a high r-squared can indicate the problems with the regression model.

A low r-squared figure is generally a bad sign for predictive models. However, in some cases, a good model may show a small value.

There is no universal rule on how to incorporate the statistical measure in assessing a model. The context of the experiment or forecast is extremely important and, in different scenarios, the insights from the metric can vary.

Compare 'Desired Output' vs 'Predicted Output'

In [129…
```
compare_audi_sql_cat2_n = pd.DataFrame({"Desired Output (price)": y_test,
                                        "Predicted Output (price)": y_pred_LR})
```

In [130…
```
compare_audi_sql_cat2_n[:5]
```

Out[130…

	Desired Output (price)	Predicted Output (price)
2641	21491	30833.659208
10423	20690	19120.482402
3291	19752	18547.022238
5596	31950	30925.236972
10499	4999	1988.639082

My own prediction using LinearRegression Model

In [131…
```
#Create portable serialized representations of Python objects.
import pickle
```

In [132…
```
#Write a pickled representation of obj to the open object file.

pickle.dump(LR_pipe,open('LR_pipe.pkl', 'wb'))
```

In [133…
```
audi_sql_cat2_inputs.head(5)
```

Out[133…

	model2_n	year	transmission2_n	mileage	fuelType2_n	tax2_n	mpg	engineSize
0	0	2017	1	15735	2	6	55.4	1.4
1	5	2016	0	36203	0	12	64.2	2.0
2	0	2016	1	29946	2	24	55.4	1.4
3	3	2017	0	25952	0	5	67.3	2.0
4	2	2019	1	1998	2	5	49.6	1.0

In [134…
```
# LR_pipe.predict(pd.DataFrame([['0', 2017, '1', 15000, '2', '6', 55.4, 1.4]], columns=

# LR_pipe.predict(pd.DataFrame([[A3, 2017, Manual, 15000, Petrol, 150, 55.4, 1.4]], col

LR_pipe.predict([['0', 2017, '1', 15000, '2', '6', 55.4, 1.4]])
```

Out[134…
```
array([16742.63854985])
```

In [135…
```
LR_pipe.predict([[0, 2017, 1, 15000, 2, 6, 55.4, 1.4]])
```

Out[135…
```
array([16742.63854985])
```

In [136…
```
compare_car_audi = pd.DataFrame({"Desired Output (Price)": y_test,
                                 "Predicted Output (Price)": y_pred_LR})
```

In [137…
```
compare_car_audi[:5]
```

Out[137…

	Desired Output (Price)	Predicted Output (Price)
2641	21491	30833.659208
10423	20690	19120.482402
3291	19752	18547.022238
5596	31950	30925.236972
10499	4999	1988.639082

==========

Step 5 :

Export predicted results back to SQL with time append

In [138…
```
import sys
import pyodbc as odbc
import datetime

datetime.datetime.now()
```

Out[138… datetime.datetime(2021, 5, 26, 22, 25, 59, 984273)

In [139…
```
cnxn = odbc.connect('Driver={SQL Server};'
                    'Server=DESKTOP-KLUE8FK;'
                    'Database=audi_dataset;'    #Link to SQL database
                    'Trusted_Connection=yes;')

cursor = cnxn.cursor()
cursor1 = cnxn.cursor()
cursor2 = cnxn.cursor()
cursor3 = cnxn.cursor()
cursor4 = cnxn.cursor()
cursor5 = cnxn.cursor()
cursor6 = cnxn.cursor()
```

NEW Tables create in SQL - (RUN ONCE ONLY !!!!!)

1. Predicted_Results_RF Table

In [ ]:
```
# Create Predicted_Results_RF Table
cursor.execute('''
                CREATE TABLE Predicted_Results_RF
                (audi_model nvarchar(50),
                 year int,
                 transmission nvarchar(50),
                 mileage int,
                 fuelType nvarchar(50),
                 tax nvarchar(50),
                 mpg float,
                 engineSize float,
                 Desired_Price float,
                 Predict_Price float,
                 Datetime DATETIME
                )
                ''')
cnxn.commit()
```

2. Parameter_Score Table

In [ ]:
```
# Create Parameter_Score Table
cursor1.execute('''
                CREATE TABLE Parameter_Score
                (Datetime DATETIME,
                 RF_r2 float,
                 RF_MSE float,
                 RF_MAE float,
                 RF_Training_Time float,
```

```
                 DT_r2 float,
                 DT_MSE float,
                 DT_MAE float,
                 DT_Training_Time float,
                 LR_r2 float,
                 LR_MSE float,
                 LR_MAE float,
                 LR_Training_Time float
                )
                ''')
cnxn.commit()
```

3. GridSearchCV_Table

In [ ]:
```
# Create GridSearchCV_Table
cursor2.execute('''
                CREATE TABLE GridSearchCV_Table
                (model nvarchar(50),
                 best_score float,
                 best_params nvarchar(50),
                 Datetime DATETIME
                )
                ''')
cnxn.commit()
#cursor2.close()
```

4. Comparsion Predict_results Table

In [ ]:
```
# Combine 3 tables into one table

cursor3.execute('''
                CREATE TABLE Comparison_output_Table
                (Desired_output_y_test int,
                 Predicted_output_RF_X_test int,
                 Predicted_output_DT_X_test int,
                 Predicted_output_LR_X_test int,
                 Datetime DATETIME
                )
                ''')
cnxn.commit()
```

In [ ]:
```
# DT Table
cursor4.execute('''
                CREATE TABLE Compare_output_Table_DT
                (Desired_output_y_test int,
                 Predicted_output_DT_X_test int,
                 Datetime DATETIME
                )
                ''')
cnxn.commit()
```

In [ ]:
```
# LR Table
cursor5.execute('''
                CREATE TABLE Compare_output_Table_LR
                (Desired_output_y_test int,
                 Predicted_output_LR_X_test int,
                 Datetime DATETIME
                )
```

```
                ''')
cnxn.commit()
#cursor3.close()
```

cursor.execute("ALTER TABLE " + 'Predicted_Results_Test4' + " ADD " + 'Datetime' + " " + 'DATETIME') cursor.commit()

In [140…
```
# cursor6.execute('alter table'+'Predicted_Results_RF'+'ADD'+'id'+ '' + 'int'+'not null
# cursor6.commit()
```

==========

In [141…
```
import datetime

datetime.datetime.now()
```

Out[141… datetime.datetime(2021, 5, 26, 22, 26, 4, 35563)

Step 6

Final Random Forest Model Test : Output Car_Spec & Predicted Price to SQL

Create User's entry on Car_Spec

In [142…
```
audi_sql_cat2_n.head(5)
```

Out[142…

	model2_n	year	price	transmission2_n	mileage	fuelType2_n	tax2_n	mpg	engineSize
0	0	2017	12500	1	15735	2	6	55.4	1.4
1	5	2016	16500	0	36203	0	12	64.2	2.0
2	0	2016	11000	1	29946	2	24	55.4	1.4
3	3	2017	16800	0	25952	0	5	67.3	2.0
4	2	2019	17300	1	1998	2	5	49.6	1.0

In [143…
```
#audi_sql_year = audi_sql[(audi_sql['year']>=2010 ) & (audi_sql['year']<=2020)]
#audi_sql_year
```

In [144…
```
#audi_sql.head(60)
```

In [145…
```
#audi_sql.tail(70)
```

In [146…
```
### audi_sql = pd.read_sql_query('SELECT * FROM audi_dataset.dbo.audi', conn)

# audi_sql[audi_sql["year"].between(2008, 2008)]
```

audi's Specification - User Entry

In [147…
```
# Method 1
# User input

a = str(input('Enter audi_model :'))
b = int(input('Enter year :'))
c = str(input('Enter transmission :'))
d = int(input('Enter mileage :'))
e = str(input('Enter fuelType :'))
f = str(input('Enter tax :'))
g = float(input('Enter mpg :'))
h = float(input('Enter engineSize :'))
i = float(input('Actual_Price :'))
```

```
Enter audi_model :A1
Enter year :2017
Enter transmission :Manual
Enter mileage :15735
Enter fuelType :Petrol
Enter tax :150
Enter mpg :55.4
Enter engineSize :1.4
Actual_Price :12500
```

In [148…
```
Predict_Price = float(Model_RandomForest.predict(pd.DataFrame([[model_Dictionary[a],b,t

timestamp = datetime.datetime.now()

cursor.execute("INSERT INTO Predicted_Results_RF (audi_model,year,transmission,mileage,
                    a,b,c,d,e,f,g,h,i, Predict_Price, timestamp)
cnxn.commit()
cursor.close()

print("Input the Audi Car_Spec done!")
```

```
Input the Audi Car_Spec done!
```

In [149…
```
# Method 2
# Insert Dataframe into SQL Server

# print("Input the Car_Spec")
# model = 'A3'
# year = 2017
# transmission = 'Manual'
# mileage = 15000
# fuelType = 'Petrol'
# tax = '150'
# mpg = 55.4
# engineSize = 1.4
# Predict_Price = float(pipe.predict(pd.DataFrame([['A3', 2017, 'Manual', 15000, 'Petro

# cursor.execute("INSERT INTO Predicted_Results (model,year,transmission,mileage,fuelTy
#              model,year,transmission,mileage,fuelType,tax,mpg,engineSize,Predict_Pric
# cnxn.commit()
# cursor.close()
```

Output Parameter Score to SQL

In [150…
```
# Insert Dataframe into SQL Server

RF_r2 = float(Model_RandomForest.score(X_test, y_test))
RF_MSE = float(mean_squared_error(y_test, y_pred_RF))
RF_MAE = float(mean_absolute_error(y_test, y_pred_RF))
RF_Training_Time = float(training_time_RF)
DT_r2 = float(DT_regressor.score(X_test, y_test))
DT_MSE = float(mean_squared_error(y_test, y_pred_DT))
DT_MAE = float(mean_absolute_error(y_test, y_pred_DT))
DT_Training_Time = float(training_time_DT)
LR_r2 = float(LR_pipe.score(X_test, y_test))
LR_MSE = float(mean_squared_error(y_test, y_pred_LR))
LR_MAE = float(mean_absolute_error(y_test, y_pred_LR))
LR_Training_Time = float(training_time_LR)

cursor1.execute("INSERT INTO Parameter_Score (Datetime,RF_r2,RF_MSE,RF_MAE,RF_Training_
              timestamp,RF_r2,RF_MSE,RF_MAE,RF_Training_Time,DT_r2,DT_MSE,DT_MAE,DT_Tra
cnxn.commit()
cursor1.close()

print("Input Paramater Score Done!")
```

```
Input Paramater Score Done!
```

Output Grid Search score to SQL

In [151…
```
insert_query = '''INSERT INTO GridSearchCV_Table (model,best_score,best_params,Datetime

result=find_best_model_using_gridsearchcv(audi_sql_cat2_inputs,audi_sql_cat2_target)
result

for res in result.iterrows():
    values = (res[1]['model'], res[1]['best_score'], str(res[1]['best_params']), timest
    cursor2.execute(insert_query, values)

cnxn.commit()
cursor2.close()

print("Input GridSearchCV_Table done!")
```

```
Input GridSearchCV_Table done!
```

In [152…
```
print("Run Completed!!!!")
```

```
Run Completed!!!!
```

==========

Output Desired_vs_Predicted table to SQL (RUN ONCE DAILY ONLY !!!!!)

In [ ]:
```
# Create Comparision (Desired vs Predicted) Table - combine 3 (RF, DT & LR) into 1

insert_query = '''INSERT INTO Comparison_output_Table (Desired_output_y_test,Predicted_
```

```
compare_audi_sql_cat2_n = pd.DataFrame({"Desired_output_y_test" : y_test, "Predicted_ou
compare_audi_sql_cat2_n


for res in compare_audi_sql_cat2_n.iterrows():
    values = (res[1]['Desired_output_y_test'], res[1]['Predicted_output_RF_X_test'], re
    cursor3.execute(insert_query, values)

cnxn.commit()
cursor3.close()
```

Create Comparision (Desired vs Predicted) Table - DT

insert_query = '''INSERT INTO Compare_output_Table_Test_3_DT (Desired_output_y_test,Predicted_output_DT_X_test,Datetime) values(?,?,?);'''

compare_audi_sql_cat2_n = pd.DataFrame({"Desired_output_y_test" : y_test, "Predicted_output_DT_X_test": y_pred_DT}) compare_audi_sql_cat2_n

for res in compare_audi_sql_cat2_n.iterrows(): values = (res[1]['Desired_output_y_test'], res[1]['Predicted_output_DT_X_test'], timestamp) cursor4.execute(insert_query, values) cnxn.commit() cursor4.close()

Create Comparision (Desired vs Predicted) Table - LR

insert_query = '''INSERT INTO Compare_output_Table_Test_3_LR (Desired_output_y_test,Predicted_output_LR_X_test,Datetime) values(?,?,?);'''

compare_audi_sql_cat2_n = pd.DataFrame({"Desired_output_y_test" : y_test, "Predicted_output_LR_X_test": y_pred_LR}) compare_audi_sql_cat2_n

for res in compare_audi_sql_cat2_n.iterrows(): values = (res[1]['Desired_output_y_test'], res[1]['Predicted_output_LR_X_test'], timestamp) cursor5.execute(insert_query, values) cnxn.commit() cursor5.close()

End of the notebook