CarPricePredictions

September 7, 2021

1 DataCamp Certification Case Study

1.0.1 Project Brief

You have been hired as a data scientist at a used car dealership in the UK. The sales team have been having problems with pricing used cars that arrive at the dealership and would like your help. Before they take any company wide action they would like you to work with the Toyota specialist to test your idea. They have already collected some data from other retailers on the price that a range of Toyota cars were listed at. It is known that cars that are more than £1500 above the estimated price will not sell. The sales team wants to know whether you can make predictions within this range.

The presentation of your findings should be targeted at the Head of Sales, who has no technical data science background.

The data you will use for this analysis can be accessed here: `"data/toyota.csv"`

[2]:
```python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

[3]:
```python
toyota_df = pd.read_csv('data/toyota.csv')
toyota_df.head()
```

[3]:

	model	year	price	transmission	mileage	fuelType	tax	mpg	engineSize
0	GT86	2016	16000	Manual	24089	Petrol	265	36.2	2.0
1	GT86	2017	15995	Manual	18615	Petrol	145	36.2	2.0
2	GT86	2015	13998	Manual	27469	Petrol	265	36.2	2.0
3	GT86	2017	18998	Manual	14736	Petrol	150	36.2	2.0
4	GT86	2017	17498	Manual	36284	Petrol	145	36.2	2.0

1.1 Understanding the dataset

In this section we will try to analyze what data is given to us and find if there are any missing values that needs to replaced.

[4]:
```python
toyota_df.info()
```

```
&lt;class 'pandas.core.frame.DataFrame'&gt;
RangeIndex: 6738 entries, 0 to 6737
Data columns (total 9 columns):
 #   Column        Non-Null Count  Dtype
---  ------        --------------  -----
 0   model         6738 non-null   object
 1   year          6738 non-null   int64
 2   price         6738 non-null   int64
 3   transmission  6738 non-null   object
 4   mileage       6738 non-null   int64
 5   fuelType      6738 non-null   object
 6   tax           6738 non-null   int64
 7   mpg           6738 non-null   float64
 8   engineSize    6738 non-null   float64
dtypes: float64(2), int64(4), object(3)
memory usage: 473.9+ KB
```

[5]:
```
toyota_df.describe()
```

[5]:
```
              year         price        mileage          tax          mpg  \
count  6738.000000   6738.000000    6738.000000  6738.000000  6738.000000
mean   2016.748145  12522.391066   22857.413921    94.697240    63.042223
std       2.204062   6345.017587   19125.464147    73.880776    15.836710
min    1998.000000    850.000000       2.000000     0.000000     2.800000
25%    2016.000000   8290.000000    9446.000000     0.000000    55.400000
50%    2017.000000  10795.000000   18513.000000   135.000000    62.800000
75%    2018.000000  14995.000000   31063.750000   145.000000    69.000000
max    2020.000000  59995.000000  174419.000000   565.000000   235.000000

        engineSize
count  6738.000000
mean      1.471297
std       0.436159
min       0.000000
25%       1.000000
50%       1.500000
75%       1.800000
max       4.500000
```

[6]:
```
toyota_df.nunique()
```

[6]:
```
model             18
year              23
price           2114
transmission       4
mileage         5699
fuelType           4
```

```
tax               29
mpg               81
engineSize        16
dtype: int64
```

From the data analysis we do see that there are no null values. Data consists of 18 distinct car models with 6738 individual cars and ranges from 1998 car till 2020.

Since the problem statement is to find the an algorithm for the sales inorder to give a pricing model. Price is our target variable. The price ranges from $850 - $59,995.

1.2 Data Visualization

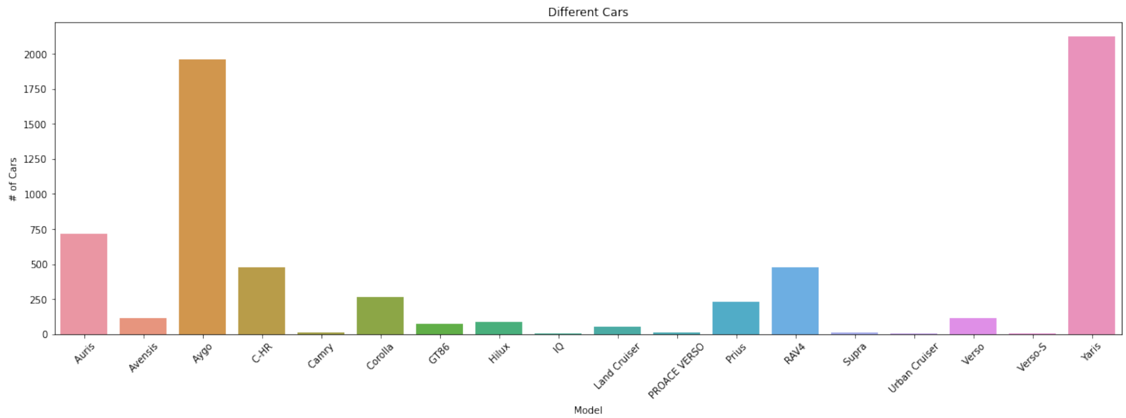
In this section we will try and find different parameters that are correlated with the price of the car.

[7]:
```
num_cars = toyota_df.groupby('model').size().reset_index(name='counts')
num_cars.sort_values(by='counts',ascending=False)
```

[7]:

	model	counts
17	Yaris	2122
2	Aygo	1961
0	Auris	712
3	C-HR	479
12	RAV4	473
5	Corolla	267
11	Prius	232
1	Avensis	115
15	Verso	114
7	Hilux	86
6	GT86	73
9	Land Cruiser	51
10	PROACE VERSO	15
13	Supra	12
4	Camry	11
8	IQ	8
14	Urban Cruiser	4
16	Verso-S	3

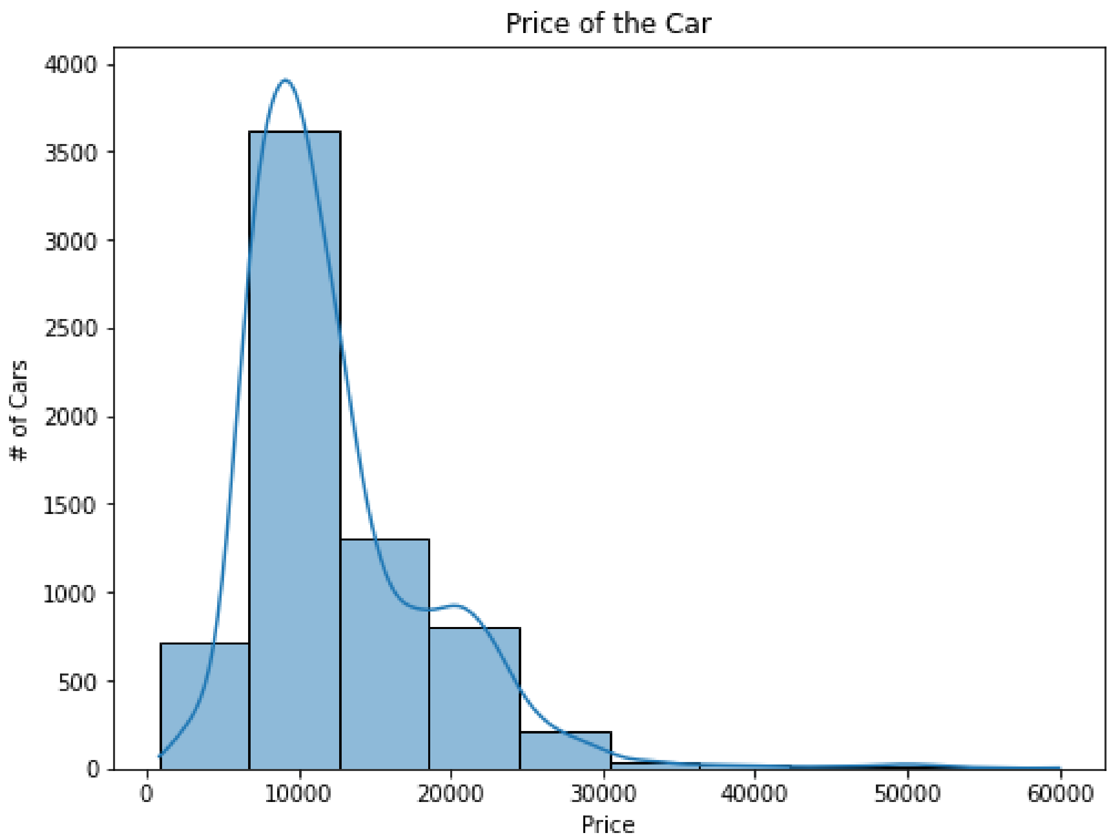
[8]:
```
fig, ax = plt.subplots(figsize=(20, 6))
sns.barplot(x='model',y='counts',data=num_cars)
plt.xticks(rotation=45)
plt.xlabel('Model')
plt.ylabel('# of Cars')
plt.title('Different Cars')
plt.show()
```



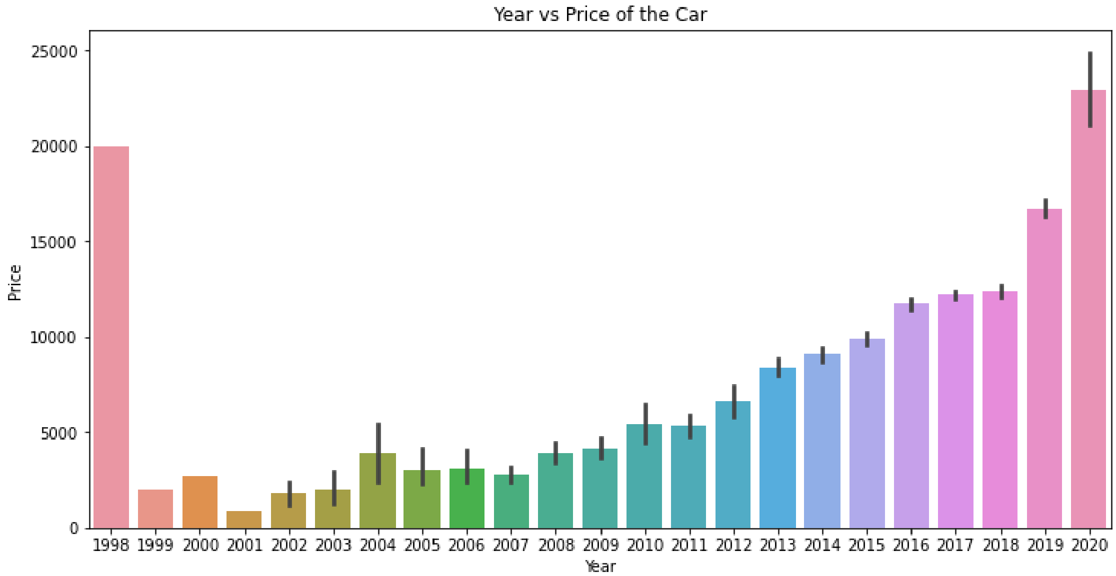
1.2.1 Price comparison to other features

Price is compared with Fuel Type, Year, Mileage on the car

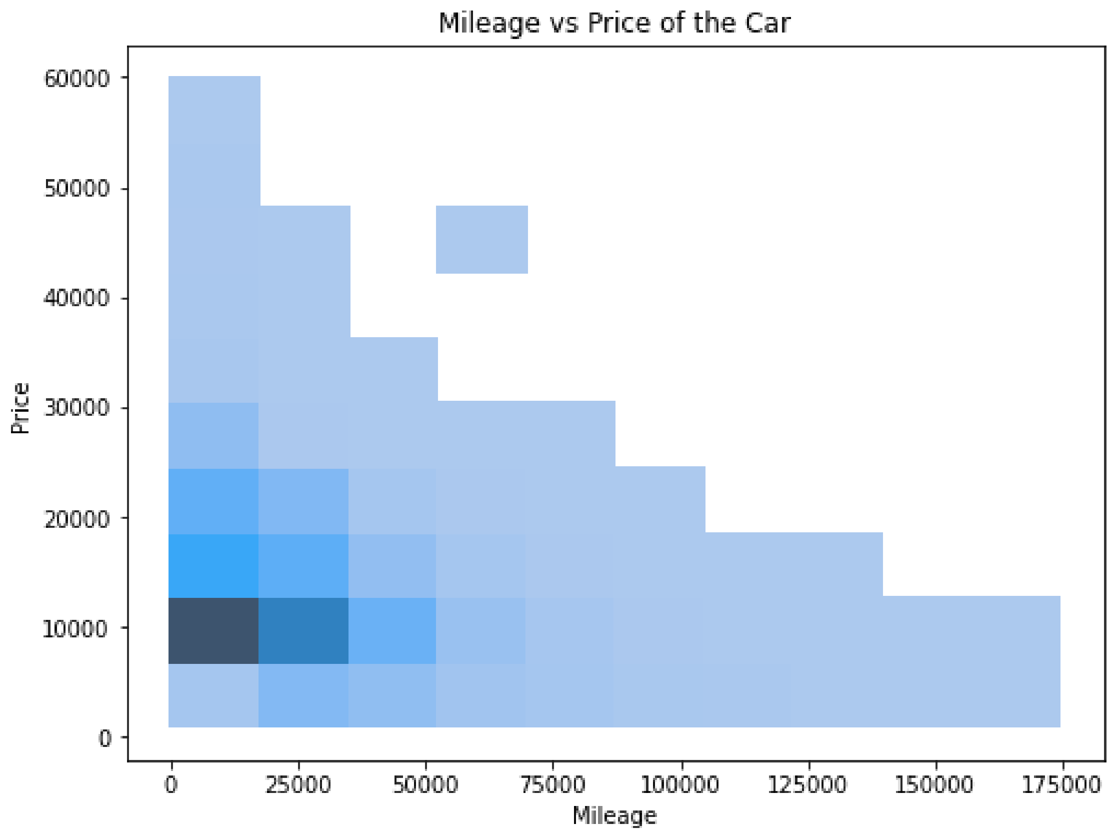
[9]:
```
fig, ax = plt.subplots(figsize=(8, 6))
sns.histplot(x='price',data=toyota_df,bins=10,kde=True)
plt.xlabel('Price')
plt.ylabel('# of Cars')
plt.title('Price of the Car')
plt.show()
```



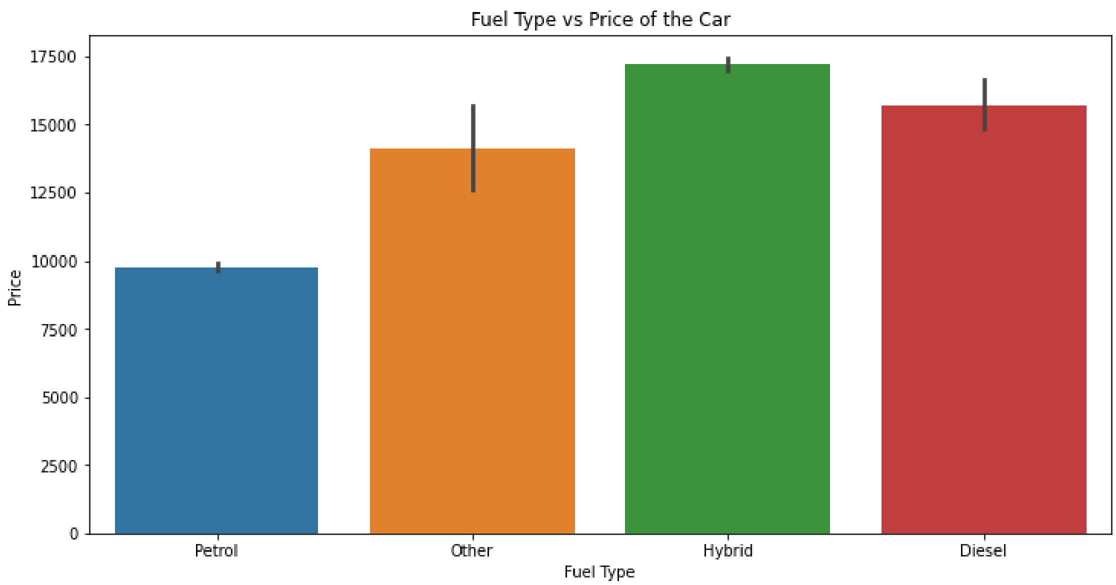
[10]:
```
fig, ax = plt.subplots(figsize=(12, 6))
sns.barplot(x='year',y='price',data=toyota_df)
plt.xlabel('Year')
plt.ylabel('Price')
plt.title('Year vs Price of the Car')
plt.show()
```



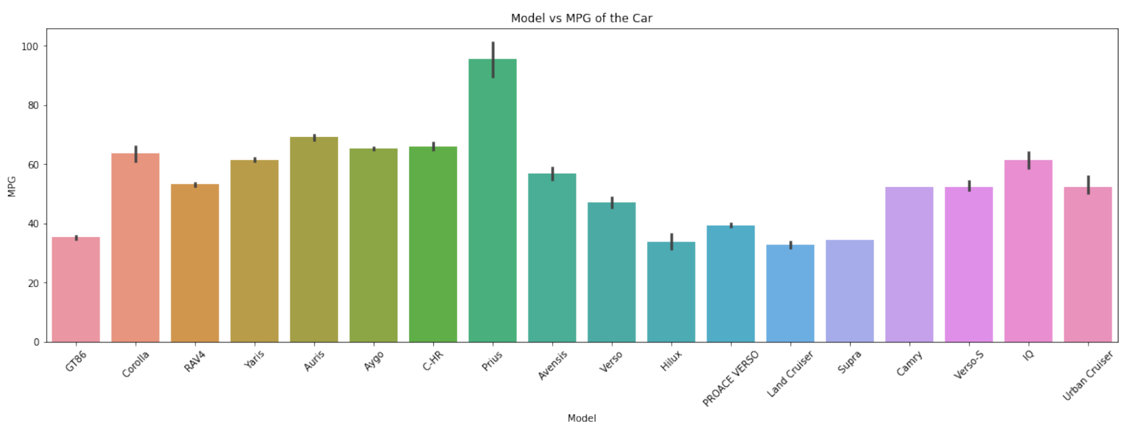
[11]:
```
fig, ax = plt.subplots(figsize=(8, 6))
sns.histplot(x='mileage',y='price',data=toyota_df,bins=10)
plt.xlabel('Mileage')
plt.ylabel('Price')
plt.title('Mileage vs Price of the Car')
plt.show()
```



[12]:
```
fig, ax = plt.subplots(figsize=(12, 6))
sns.barplot(x='fuelType',y='price',data=toyota_df)
plt.xlabel('Fuel Type')
plt.ylabel('Price')
plt.title('Fuel Type vs Price of the Car')
plt.show()
```



[13]:
```
fig, ax = plt.subplots(figsize=(20, 6))
sns.barplot(x='model',y='mpg',data=toyota_df)
plt.xticks(rotation=45)
plt.xlabel('Model')
plt.ylabel('MPG')
plt.title('Model vs MPG of the Car')
plt.show()
```



1.2.2 Finding the most expensive cars

From the histogram we did see that the data consists of very few expensive cars and would like to analyze as to which model and years do they belong to inorder to better get an understanding for the pricing

[14]:
```
num_car_30k = toyota_df[toyota_df['price'] > 30000].value_counts().reset_index()
num_car_30k
```

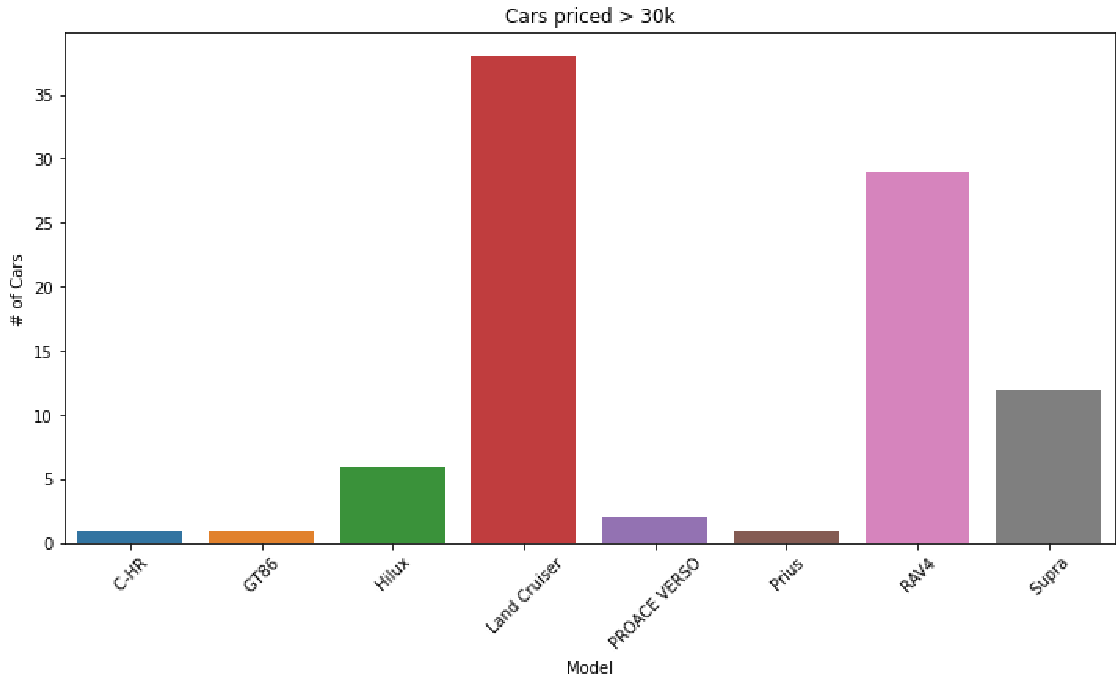
[14]:
```
             model  year  price transmission  mileage fuelType  tax   mpg  \
0             C-HR  2019  30990    Automatic     2500   Hybrid  140  54.3
1             RAV4  2019  33950    Automatic     9125   Hybrid  140  49.6
2             RAV4  2019  33880    Automatic     7576   Hybrid  140  49.6
3             RAV4  2019  33626    Automatic     7191   Hybrid  135  49.6
4             RAV4  2019  33595    Automatic     8000   Hybrid  140  49.6
..             …     …      …          …         …      …    …
85    Land Cruiser  2019  42990    Semi-Auto    22845   Diesel  150  30.1
86    Land Cruiser  2019  42444    Semi-Auto    10083   Diesel  145  30.1
87    Land Cruiser  2019  40999    Semi-Auto    11619   Diesel  145  30.1
88    Land Cruiser  2019  40995    Semi-Auto    11404   Diesel  145  30.1
89           Supra  2019  59995    Automatic     9909    Other  150  34.5

    engineSize  0
0          2.0  1
1          2.5  1
2          2.5  1
3          2.5  1
4          2.5  1
..         … ..
85         2.8  1
86         2.8  1
87         2.8  1
88         2.8  1
89         3.0  1

[90 rows x 10 columns]
```

[15]:
```
model_30k = num_car_30k.groupby('model').size().reset_index(name='counts')
model_30k
```

[15]:
```
           model  counts
0           C-HR       1
1           GT86       1
2          Hilux       6
3   Land Cruiser      38
4   PROACE VERSO       2
5          Prius       1
6           RAV4      29
7          Supra      12
```

[16]:
```
fig, ax = plt.subplots(figsize=(12, 6))
sns.barplot(x='model',y='counts',data=model_30k)
plt.xticks(rotation=45)
```

```
plt.xlabel('Model')
plt.ylabel('# of Cars')
plt.title('Cars priced > 30k')
plt.show()
```



[17]:
```
model_year_30k = num_car_30k.groupby(['model','year']).size().
    ↪reset_index(name='counts')
model_year_30k
```

[17]:

	model	year	counts
0	C-HR	2019	1
1	GT86	2020	1
2	Hilux	2019	3
3	Hilux	2020	3
4	Land Cruiser	2014	2
5	Land Cruiser	2015	3
6	Land Cruiser	2016	2
7	Land Cruiser	2017	7
8	Land Cruiser	2018	1
9	Land Cruiser	2019	15
10	Land Cruiser	2020	8
11	PROACE VERSO	2020	2
12	Prius	2020	1
13	RAV4	2019	25
14	RAV4	2020	4

```
15          Supra  2019      12
```

[18]:
```
min_max_value = toyota_df.groupby('model')['price'].agg(['min','max','mean']).
 ↪reset_index()
min_max_value
```

[18]:

	model	min	max	mean
0	Auris	1599	19300	12507.911517
1	Avensis	850	16495	9884.356522
2	Aygo	1295	15000	7905.414584
3	C-HR	11995	30990	20651.540710
4	Camry	24990	29990	26910.090909
5	Corolla	899	29450	20942.734082
6	GT86	9995	31000	19908.849315
7	Hilux	7750	39257	21504.593023
8	IQ	2495	5995	4247.250000
9	Land Cruiser	5975	54991	36487.156863
10	PROACE VERSO	23950	47990	28680.200000
11	Prius	2495	31995	18998.844828
12	RAV4	1600	37440	18161.059197
13	Supra	47498	59995	50741.000000
14	Urban Cruiser	3995	4995	4617.500000
15	Verso	2300	17995	12169.157895
16	Verso-S	4450	6995	5746.666667
17	Yaris	950	25995	10553.083883

Interesting facts are derived from our above analysis

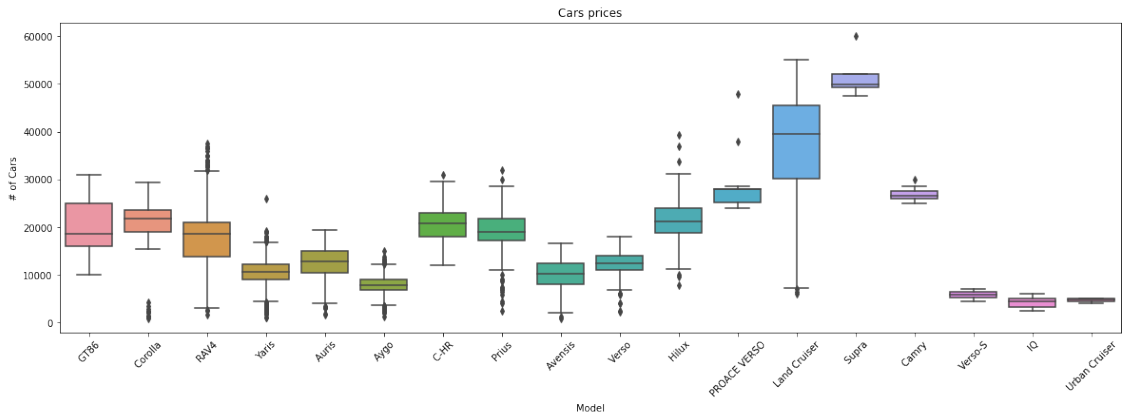
A total of 90 cars that are price greater than 30k and comprises of 8 different models.

Majority of the cars belong to models - Land Cruiser and RAV4.

The most expensive car is Supra which is priced at 59.95k and the cheapest car being Avensis.

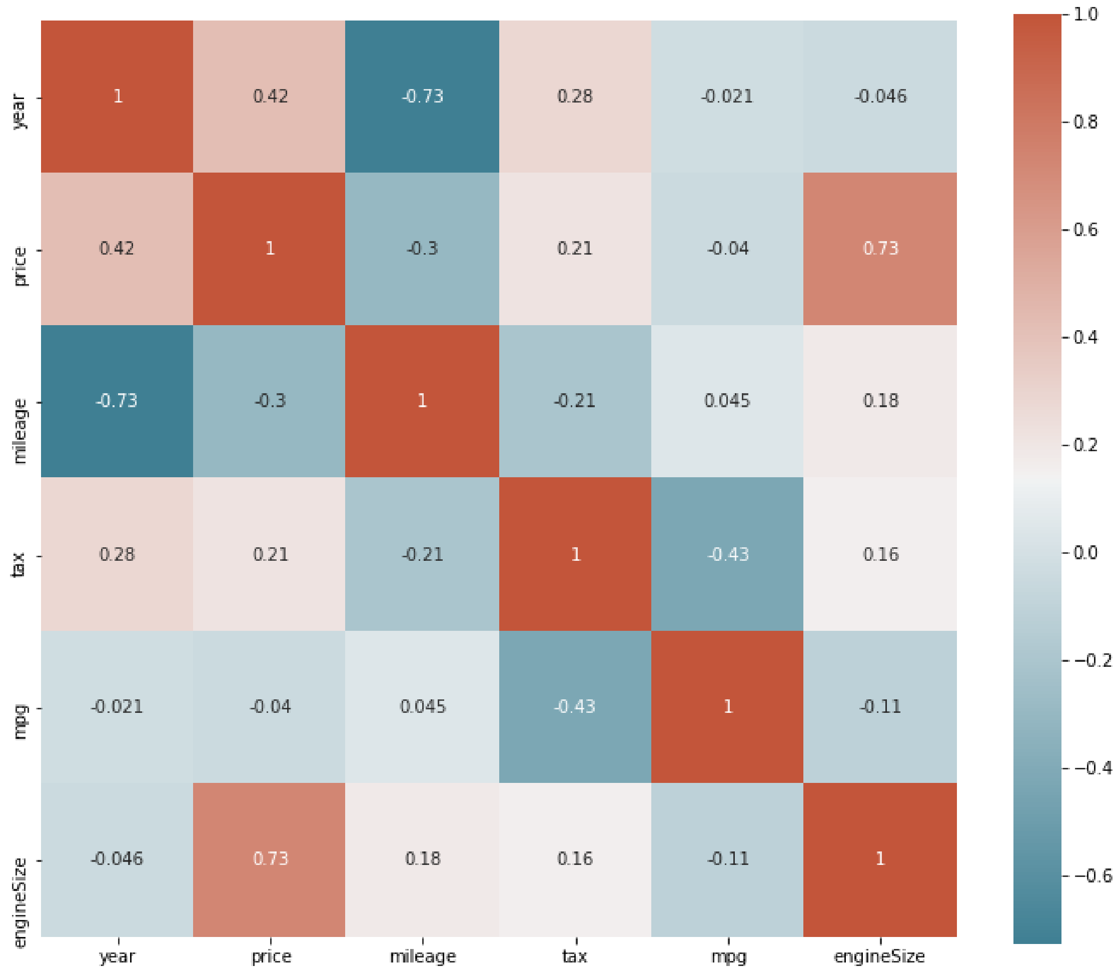
Lets try and visualize different correlations

[19]:
```
fig, ax = plt.subplots(figsize=(20, 6))
sns.boxplot(x='model',y='price',data=toyota_df)
plt.xticks(rotation=45)
plt.xlabel('Model')
plt.ylabel('# of Cars')
plt.title('Cars prices')
plt.show()
```

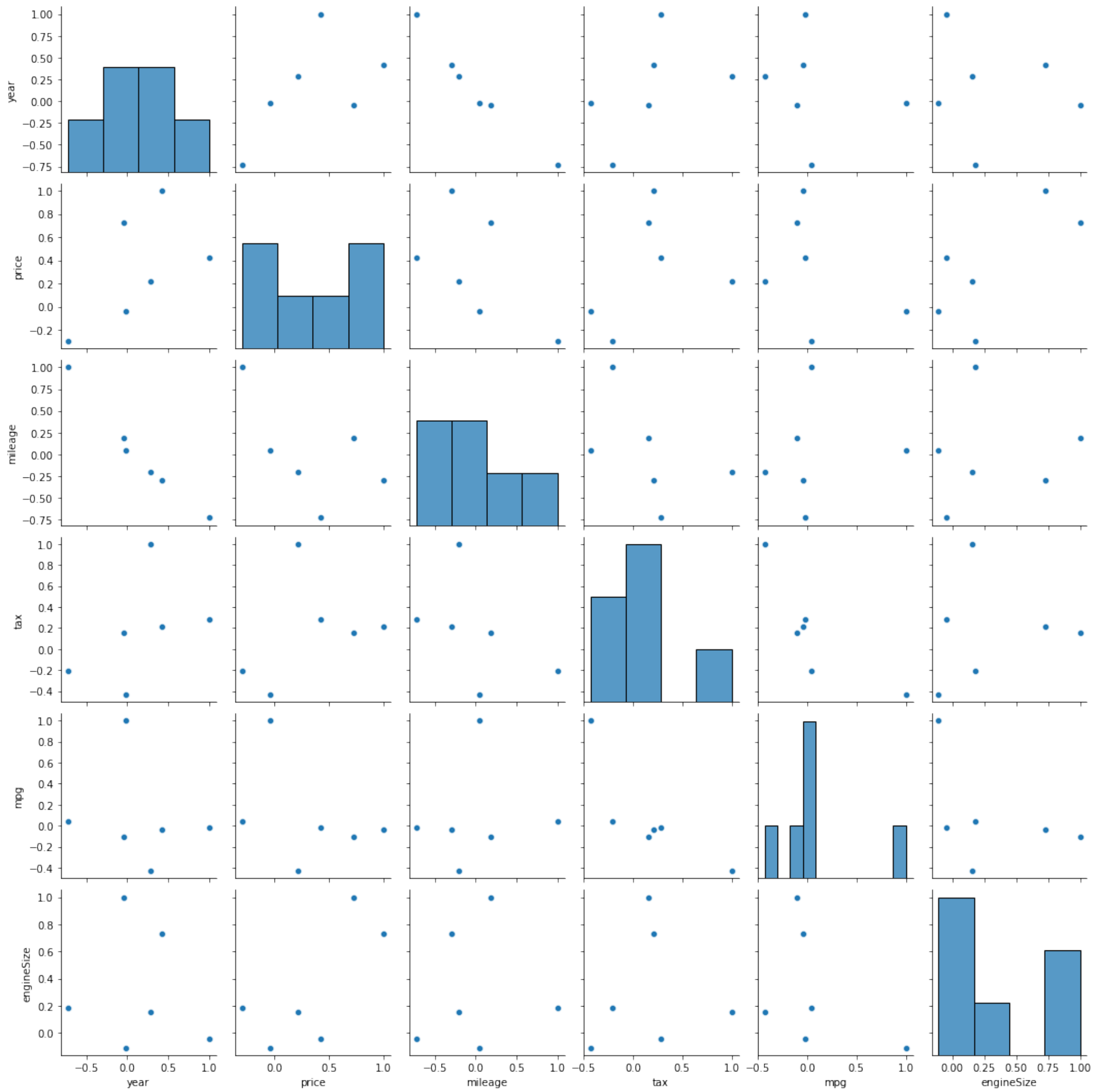


[20]:

```
corr_df = toyota_df.corr()
fig=plt.gcf()
fig.set_size_inches(12,10)
sns.heatmap(data=corr_df, square=True, annot=True, cbar=True,cmap=sns.
 ↪diverging_palette(220, 20, as_cmap=True))
plt.show()
```



[21]:
```
fig.set_size_inches(10,10)
sns.pairplot(corr_df)
plt.show()
```



From the correlation we can see that there is positive correlation with EngineSize and Price. However, from the scatter plot we see that there is no linear correlation between any values.

1.3 Data Cleaning & Train Test Split

In this section, we will be achieving the following things

Converting the object columns into numerical columns inorder to feed it to different Regression Algorithms.

Since the numerical values are widely ranged we will convert them using MinMax Scaler.

[22]:
```
toyota_df.select_dtypes('object').columns
```

[22]:
```
Index(['model', 'transmission', 'fuelType'], dtype='object')
```

[23]:
```
from sklearn.preprocessing import OneHotEncoder,MinMaxScaler
from sklearn.model_selection import␣
↪train_test_split,GridSearchCV,cross_val_score
from sklearn.linear_model import LinearRegression
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import GradientBoostingRegressor,RandomForestRegressor
from sklearn.metrics import mean_squared_error as MSE
```

[24]:
```
toyota_df.columns
```

[24]:
```
Index(['model', 'year', 'price', 'transmission', 'mileage', 'fuelType', 'tax',
       'mpg', 'engineSize'],
      dtype='object')
```

[25]:
```
X = toyota_df.iloc[:, [0,1,3,4,5,6,7,8]]
y = toyota_df.iloc[:,2]
print("Shape of X - {}".format(X.shape))
print("Shape of y - {}".format(y.shape))
```

```
Shape of X - (6738, 8)
Shape of y - (6738,)
```

[26]:
```
# Splitting the data into Train Test
X_train, X_test,y_train,y_test =␣
↪train_test_split(X,y,random_state=42,test_size=0.2)
```

[27]:
```
categorical_var = ['model', 'transmission', 'fuelType']

OH_encoder = OneHotEncoder(handle_unknown='ignore', sparse=False)
OH_cols_train = pd.DataFrame(OH_encoder.fit_transform(X_train[categorical_var]))
OH_cols_test = pd.DataFrame(OH_encoder.transform(X_test[categorical_var]))

# One-hot encoding removed index; put it back
OH_cols_train.index = X_train.index
OH_cols_test.index = X_test.index

# Remove categorical columns (will replace with one-hot encoding)
num_X_train = X_train.drop(categorical_var, axis=1)
num_X_test = X_test.drop(categorical_var, axis=1)

# Add one-hot encoded columns to numerical features
OH_X_train = pd.concat([num_X_train, OH_cols_train], axis=1)
OH_X_test = pd.concat([num_X_test, OH_cols_test], axis=1)
```

[28]:
```
print("Shape of training Data - {}".format(OH_X_train.shape))
print("Shape of test Data - {}".format(OH_X_test.shape))
```

```
Shape of training Data - (5390, 31)
Shape of test Data - (1348, 31)
```

[29]:
```
minmax = MinMaxScaler(feature_range=[0,1])
scaled_X_train = minmax.fit_transform(OH_X_train)
scaled_X_test = minmax.transform(OH_X_test)
```

[30]:
```
print("Shape of final X_train - {}".format(scaled_X_train.shape))
```

```
Shape of final X_train - (5390, 31)
```

1.4 Defining BaseLine model

We will start with the basic Linear Regression model. Inorder to form a baseline that will be used to compare our future evaluations

[31]:
```
lreg = LinearRegression(n_jobs=-1)
lreg.fit(scaled_X_train,y_train)

# Use logreg to predict instances from the test set and store it
y_pred = lreg.predict(scaled_X_test)

rsme = MSE(y_test,y_pred) ** (1/2)
print("RSME of the base model - {}".format(rsme))
```

```
RSME of the base model - 1768.284258766275
```

1.5 Trying different Regression models

In this approach we are trying to find the model that best fits the data and reduces the RSME score.

[32]:
```
def get_models():
    model = dict()
    model['LR'] = LinearRegression()
    model['CART'] = DecisionTreeRegressor()
    model['RF'] = RandomForestRegressor()
    model['GBM'] = GradientBoostingRegressor()
    return model
```

[33]:
```
def evaluate_model(X,y,model):
    scores = cross_val_score(model, X, y, scoring='neg_mean_squared_error',␣
↪cv=5, n_jobs=-1)
    score = (np.mean(scores)*-1)**(1/2)
    return score
```

[34]:
```
models = get_models()
results = dict()
for key,value in models.items():
```

```
    results[key] = evaluate_model(scaled_X_train,y_train,value)
    print('>%s %.3f' % (key, results[key]))
```

```
>LR 1717.376
>CART 1581.283
>RF 1268.607
>GBM 1297.524
```

1.5.1 Using the best fit Model and Tuning it using HyperParameters

From the above analysis we see that RandomForest Regressor gave us the lowest RSME amongst all the ones compared on our training dataset. We will try to do hyperparameter tuning inorder to achieve the best possible results and values that fit our dataset.

[35]:
```
# Number of trees in random forest
n_estimators = [int(x) for x in np.linspace(start = 500, stop = 2000, num = 4)]
# Maximum number of levels in tree
max_depth = [int(x) for x in np.linspace(1, 10, num = 3)]
max_depth.append(None)
```

[36]:
```
# Create the params grid
param_grid = {'n_estimators': n_estimators,
              'max_depth': max_depth
             }
```

[37]:
```
gm_cv = GridSearchCV(estimator=RandomForestRegressor(),param_grid=␣
 ↪param_grid,cv = 5,n_jobs=-1)
gm_cv.fit(scaled_X_train,y_train)
print(gm_cv.best_params_)
```

```
{'max_depth': 10, 'n_estimators': 500}
```

[38]:
```
y_gm_cv_preds = gm_cv.predict(scaled_X_test)
rsme_hyp_gm_cv = MSE(y_test,y_gm_cv_preds)**(1/2)
print("Tuned RandomForest Regressor RMSE: {}".format(rsme_hyp_gm_cv))
```

```
Tuned RandomForest Regressor RMSE: 1168.2610910121873
```