# Assignment8

```
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
from sklearn import linear_model
from sklearn.model_selection import train_test_split
from sklearn import metrics
```

# How Much is Your Car Worth?

Data about the retail price of 2005 General Motors cars can be found in `car_data.csv`.

The columns are:

1. Price: suggested retail price of the used 2005 GM car in excellent condition.
2. Mileage: number of miles the car has been driven
3. Make: manufacturer of the car such as Saturn, Pontiac, and Chevrolet
4. Model: specific models for each car manufacturer such as Ion, Vibe, Cavalier
5. Trim (of car): specific type of car model such as SE Sedan 4D, Quad Coupe 2D
6. Type: body type such as sedan, coupe, etc.
7. Cylinder: number of cylinders in the engine
8. Liter: a more specific measure of engine size
9. Doors: number of doors
10. Cruise: indicator variable representing whether the car has cruise control (1 = cruise)
11. Sound: indicator variable representing whether the car has upgraded speakers (1 = upgraded)
12. Leather: indicator variable representing whether the car has leather seats (1 = leather)

# Tasks, Part 1

1. Find the linear regression equation for mileage vs price.
2. Chart the original data and the equation on the chart.
3. Find the equation's $R^2$ score (use the `.score` method) to determine whether the equation is a good fit for this data. (0.8 and greater is considered a strong correlation.)

# Tasks, Part 2

1. Use mileage, cylinders, liters, doors, cruise, sound, and leather to find the linear regression equation.
2. Find the equation's $R^2$ score (use the `.score` method) to determine whether the equation is a good fit for this data. (0.8 and greater is considered a strong correlation.)
3. Find the combination of the factors that is the best predictor for price.

# Tasks, Hard Mode

1. Research dummy variables in scikit-learn to see how to use the make, model, and body type.
2. Find the best combination of factors to predict price.

```
data = pd.read_csv("car_data.csv")
print(data.shape)
data.head()
```

```
(804, 12)
```

| | Price | Mileage | Make | Model | Trim | Type | Cylinder | Liter | Doors | Cruise | Sound | Leather |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 17314.103129 | 8221 | Buick | Century | Sedan 4D | Sedan | 6 | 3.1 | 4 | 1 | 1 | 1 |

```
data.describe().style.background_gradient()
```

| | Price | Mileage | Cylinder | Liter | Doors | Cruise | Sound | Leather |
|---|---|---|---|---|---|---|---|---|
| **count** | 804.000000 | 804.000000 | 804.000000 | 804.000000 | 804.000000 | 804.000000 | 804.000000 | 804.000000 |
| **mean** | 21343.143767 | 19831.934080 | 5.268657 | 3.037313 | 3.527363 | 0.752488 | 0.679104 | 0.723881 |
| **std** | 9884.852801 | 8196.319707 | 1.387531 | 1.105562 | 0.850169 | 0.431836 | 0.467111 | 0.447355 |
| **min** | 8638.930895 | 266.000000 | 4.000000 | 1.600000 | 2.000000 | 0.000000 | 0.000000 | 0.000000 |
| **25%** | 14273.073870 | 14623.500000 | 4.000000 | 2.200000 | 4.000000 | 1.000000 | 0.000000 | 0.000000 |
| **50%** | 18024.995019 | 20913.500000 | 6.000000 | 2.800000 | 4.000000 | 1.000000 | 1.000000 | 1.000000 |
| **75%** | 26717.316636 | 25213.000000 | 6.000000 | 3.800000 | 4.000000 | 1.000000 | 1.000000 | 1.000000 |
| **max** | 70755.466717 | 50387.000000 | 8.000000 | 6.000000 | 4.000000 | 1.000000 | 1.000000 | 1.000000 |

```
data.isnull().sum()
```

```
Price       0
Mileage     0
Make        0
Model       0
Trim        0
Type        0
Cylinder    0
Liter       0
Doors       0
Cruise      0
Sound       0
Leather     0
dtype: int64
```

```
data.duplicated().sum()
```

```
0
```

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 804 entries, 0 to 803
Data columns (total 12 columns):
 #   Column    Non-Null Count  Dtype  
---  ------    --------------  -----  
 0   Price     804 non-null    float64
 1   Mileage   804 non-null    int64  
 2   Make      804 non-null    object 
 3   Model     804 non-null    object 
 4   Trim      804 non-null    object 
 5   Type      804 non-null    object 
 6   Cylinder  804 non-null    int64  
 7   Liter     804 non-null    float64
 8   Doors     804 non-null    int64  
 9   Cruise    804 non-null    int64  
 10  Sound     804 non-null    int64  
 11  Leather   804 non-null    int64  
dtypes: float64(2), int64(6), object(4)
memory usage: 75.5+ KB
```

# Part 1

```
d1=data[['Mileage','Price']]
import seaborn as sns
plt.figure(figsize=(10,8))
sns.boxplot(d1.Mileage)
```

```
ı version 0.12, the only valid positional argument will be `data`, and passing other arguments without an explicit keyword will r
```

```
percentile25=d1.Mileage.quantile(0.25)
percentile75=d1.Mileage.quantile(0.75)
iqr=percentile75-percentile25
upper_limit=round(percentile75+1.5*iqr)
lower_limit=round(percentile25-1.5*iqr)
print(f"upper limit is {upper_limit} \n lower limit is {lower_limit}")
```

```
upper limit is 41097
```

```
lower limit is -1261
```

```
print("Number of outliers in mileage : ",d1[d1.Mileage>=upper_limit].shape[0])
```

```
Number of outliers in mileage :  5
```

```
notoutlier=d1.loc[(d1['Mileage']>lower_limit)& (d1['Mileage']<upper_limit)]
notoutlier
```

| | Mileage | Price |
|---|---|---|
| **0** | 8221 | 17314.103129 |
| **1** | 9135 | 17542.036083 |
| **2** | 13196 | 16218.847862 |
| **3** | 16342 | 16336.913140 |
| **4** | 19832 | 16339.170324 |
| **...** | ... | ... |
| **799** | 16229 | 16507.070267 |
| **800** | 19095 | 16175.957604 |
| **801** | 20484 | 15731.132897 |
| **802** | 25979 | 15118.893228 |
| **803** | 35662 | 13585.636802 |

799 rows × 2 columns

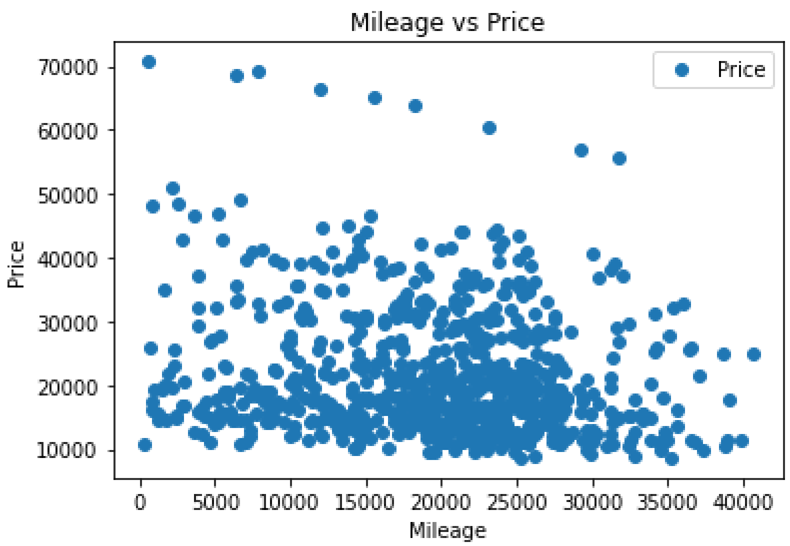
```
notoutlier.describe().style.background_gradient()
```

| | Mileage | Price |
|---|---|---|
| **count** | 799.000000 | 799.000000 |
| **mean** | 19673.856070 | 21287.275019 |
| **std** | 7967.875493 | 9842.539866 |
| **min** | 266.000000 | 8638.930895 |
| **25%** | 14596.000000 | 14261.330129 |
| **50%** | 20870.000000 | 18004.870415 |
| **75%** | 25158.000000 | 26495.534994 |

```
plt.figure(figsize=(10,10))
sns.boxplot(notoutlier.Mileage)
```

```
ı version 0.12, the only valid positional argument will be `data`, and passing other arguments without an explicit keyword will r
```

```
notoutlier.plot(x='Mileage',y='Price',style='o')
plt.title('Mileage vs Price')
plt.xlabel('Mileage')
plt.ylabel('Price')
plt.show()
```




```
notoutlier.corr()
```

```
notoutlier.corr()
```

| | Mileage | Price |
|---|---|---|
| **Mileage** | 1.000000 | -0.165933 |
| **Price** | -0.165933 | 1.000000 |

```
x=notoutlier['Mileage']
x= x.to_frame()
y=notoutlier['Price']
```

# Training and Testing and Modeling

```
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 = 30)
```

```
from sklearn import linear_model
linreg=linear_model.LinearRegression()
linreg.fit(X_train,y_train)
```

```
LinearRegression()
```

```
print('intercept :', linreg.intercept_)
print('coefficients :', linreg.coef_)
print('R2 score :', linreg.score(x, y))
```

```
intercept : 25560.767313927863
coefficients : [-0.21109352]
R2 score : 0.02735926386971299
```

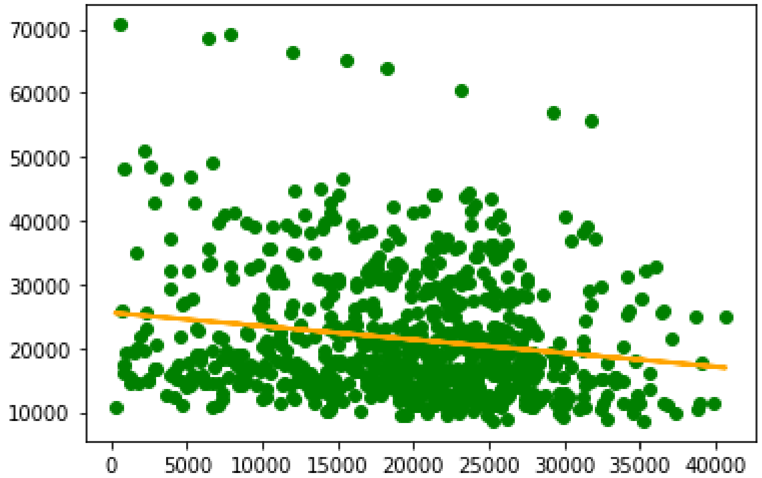
```
y_pred=linreg.predict(X_test)
y_pred
```

```
array([23667.8917245 , 24578.76026118, 24603.03601592, 20831.21700935,
       19844.9880862 , 21461.5422586 , 24022.52883727, 20019.1402398 ,
       24649.26549669, 22168.91664247, 19999.50854248, 19812.4796842 ,
       21719.92072648, 23493.10629034, 20488.8233207 , 23726.99790996,
       20374.62172665, 21089.59547722, 19214.66283695, 21879.08524018,
       21547.035134  , 22581.18228607, 25378.17141955, 20952.80687658,
       23764.99474347, 20249.02108254, 19858.07588441, 19746.40741259,
       21876.97430499, 19864.19759648, 20531.25311812, 23446.87680957,
       22554.37340909, 20543.70763578, 25248.34890506, 21046.32130572,
       21083.8959522 , 22833.43904188, 21830.11154366, 20280.47401695,
       21236.72766032, 20111.1770143 , 19840.97730933, 20782.24331282,
       21444.44368352, 21259.94794747, 24465.4030412 , 22049.64880395,
       21982.52106474, 21528.88109132, 21063.63097432, 18924.83143467,
       21045.26583813, 18944.25203846, 20744.87975987, 19760.76177192,
       22485.55692173, 20925.15362553, 20659.8090715 , 24474.90224958,
       21901.46115325, 19825.77857593, 21568.98886003, 25095.93938397,
       21436.21103626, 22490.20097916, 21815.75718433, 22526.08687748,
       18692.83965673, 22585.40415646, 20983.41543691, 18519.32078369,
       20496.84487444, 22363.96705449, 20065.15862705, 21521.28172462,
       21399.05857682, 19964.46701824, 19295.51165492, 20272.45246321,
       22076.03549389, 23170.76648606, 21440.01071961, 22541.28561088,
       20080.77954749, 19697.43371607, 20285.11807438, 21038.93303254,
       19564.02261174, 20260.84231963, 21030.48929176, 24165.86133702,
       24795.55330571, 20203.21378881, 23447.51009013, 18406.38575075,
       21361.27283683, 21309.97711159, 20032.01694449, 21981.46559715,
       21653.21517431, 21716.75432368, 20237.41093897, 19350.39597   ,
       24069.3915986 , 22024.10648809, 22166.80570727, 20739.18023484,
       20460.11460205, 24929.17550356, 20490.08988182, 21145.32416637,
       20273.08574377, 22473.10240408, 20164.37258122, 24676.70765423,
       20268.86387338, 21642.44940482, 22448.61555582, 20571.57198035,
       20420.85120742, 20448.50445848, 22133.66402471, 22375.99938511,
       20942.67438765, 20115.82107173, 23328.03115809, 19568.87776269,
       23888.06226534, 21621.55114638, 22546.77404239, 20982.57106283,
       23449.19883829, 20959.77296273, 19743.45210332, 22407.66341303,
       20835.86106677, 22641.76612617, 20025.47304538, 21833.70013349,
       20935.7083015 , 20115.39888469, 25040.63288186, 18846.93792597,
       18878.60195389, 20437.52759546, 22354.2567526 , 24399.96405015,
       20569.46104516, 20101.46671241, 21666.30297252, 24133.56402853,
```

```
       24953.02907126, 20627.3006695 , 20815.5960889 , 20537.16373667,
       24146.65182674, 17780.91565245, 18211.54643225, 21350.50706734])
```

```
plt.scatter(x, y, color='green')
plt.plot(x, linreg.predict(x), color='orange', linewidth=2)
```

```
[<matplotlib.lines.Line2D at 0x7fbe529fb990>]
```




```
linreg.predict([[8000]])
print('R2 SCORE :', metrics.r2_score(y_test,y_pred))
```

```
R2 SCORE : 0.01932680455941127
/usr/local/lib/python3.7/dist-packages/sklearn/base.py:451: UserWarning: X does not have valid feature names, but LinearRegress
  "X does not have valid feature names, but"
```

## Part 2

```
features = ['Mileage', 'Cylinder', 'Liter', 'Doors', 'Cruise', 'Sound', 'Leather']
d2=data[features]
```

```
X = d2[features]
y = data.Price
d2.head()
```

| | Mileage | Cylinder | Liter | Doors | Cruise | Sound | Leather |
|---|---|---|---|---|---|---|---|
| **0** | 8221 | 6 | 3.1 | 4 | 1 | 1 | 1 |
| **1** | 9135 | 6 | 3.1 | 4 | 1 | 1 | 0 |
| **2** | 13196 | 6 | 3.1 | 4 | 1 | 1 | 0 |
| **3** | 16342 | 6 | 3.1 | 4 | 1 | 0 | 0 |
| **4** | 19832 | 6 | 3.1 | 4 | 1 | 0 | 1 |

```
d2.describe().style.background_gradient()
```

| | Mileage | Cylinder | Liter | Doors | Cruise | Sound | Leather |
|---|---|---|---|---|---|---|---|
| **count** | 804.000000 | 804.000000 | 804.000000 | 804.000000 | 804.000000 | 804.000000 | 804.000000 |
| **mean** | 19831.934080 | 5.268657 | 3.037313 | 3.527363 | 0.752488 | 0.679104 | 0.723881 |
| **std** | 8196.319707 | 1.387531 | 1.105562 | 0.850169 | 0.431836 | 0.467111 | 0.447355 |
| **min** | 266.000000 | 4.000000 | 1.600000 | 2.000000 | 0.000000 | 0.000000 | 0.000000 |
| **25%** | 14623.500000 | 4.000000 | 2.200000 | 4.000000 | 1.000000 | 0.000000 | 0.000000 |
| **50%** | 20913.500000 | 6.000000 | 2.800000 | 4.000000 | 1.000000 | 1.000000 | 1.000000 |
| **75%** | 25213.000000 | 6.000000 | 3.800000 | 4.000000 | 1.000000 | 1.000000 | 1.000000 |
| **max** | 50387.000000 | 8.000000 | 6.000000 | 4.000000 | 1.000000 | 1.000000 | 1.000000 |

```
for i in X:
  percentile25=d2[i].quantile(0.25)
  percentile75=d2[i].quantile(0.75)
  iqr=percentile75-percentile25
```

```
  iqr=percentile75-percentile25
  upper_limit=round(percentile75+1.5*iqr)
  lower_limit=round(percentile25-1.5*iqr)
  print(f"{i} \n upper limit is {upper_limit} \n lower limit is {lower_limit}")
```

```
Mileage
 upper limit is 41097
 lower limit is -1261
Cylinder
 upper limit is 9
 lower limit is 1
Liter
 upper limit is 6
 lower limit is 0
Doors
 upper limit is 4
 lower limit is 4
Cruise
 upper limit is 1
 lower limit is 1
Sound
 upper limit is 2
 lower limit is -2
Leather
 upper limit is 2
 lower limit is -2
```

```
X = d2[features]
y = data.Price
```

```
from sklearn.model_selection import train_test_split #import the required function
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state = 30)

from sklearn.preprocessing import StandardScaler ## standrard scaling
scaler = StandardScaler() #initialise to a variable
scaler.fit(X_train,y_train) # we are finding the values of mean and sd from the td
X_train_scaled = scaler.transform(X_train) # fit (mean, sd) and then transform the training data
X_test_scaled = scaler.transform(X_test) # transform the test data
```

```
regressor = linear_model.LinearRegression()
regressor.fit(X_train_scaled, y_train)
```

```
LinearRegression()
```

```
coeff_df = pd.DataFrame(regressor.coef_,['Mileage', 'Cylinder', 'Liter', 'Doors', 'Cruise', 'Sound', 'Leather'],columns=['Coefficient
y_pred = regressor.predict(X_test_scaled)
coeff_df
```

| | Coefficients |
|---|---|
| **Mileage** | -1661.044065 |
| **Cylinder** | 5220.545797 |
| **Liter** | -946.726507 |
| **Doors** | -1185.252532 |
| **Cruise** | 2816.539386 |
| **Sound** | -918.341751 |
| **Leather** | 1685.916881 |

```
regressor.intercept_
```

```
21451.07890200263
```

```
df = pd.DataFrame({'Actual': y_test, 'Predicted': y_pred})
df
```

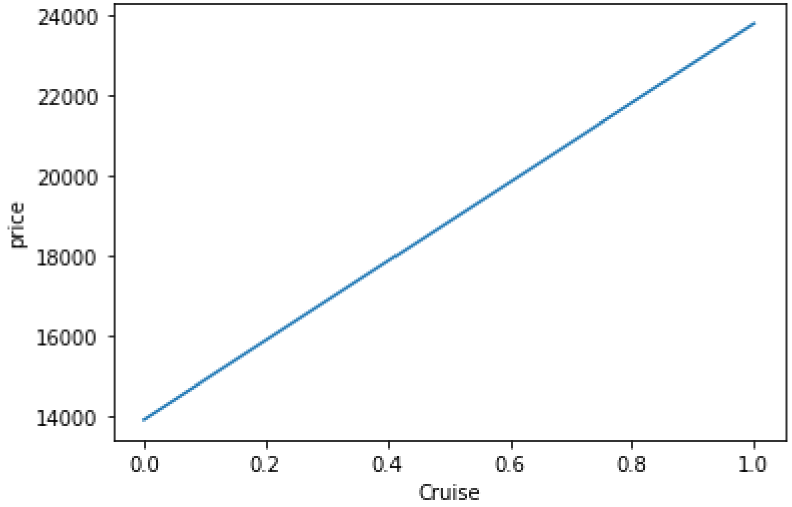
| | Actual | Predicted |
|---|---|---|
| **200** | 10813.343521 | 25184.625951 |
| **128** | 31181.715159 | 32309.408796 |
| **599** | 12209.559623 | 13636.692269 |
| **398** | 16143.957292 | 23718.519746 |
| **71** | 26060.335350 | 23144.302034 |
| **...** | ... | ... |
| **85** | 43892.467880 | 31952.166644 |
| **165** | 10386.040218 | 10363.695370 |
| **67** | 23077.565910 | 25998.350865 |
| **767** | 15194.975354 | 15582.447578 |

```
from sklearn import metrics
print('R2 SCORE :', metrics.r2_score(y_test,y_pred))
```

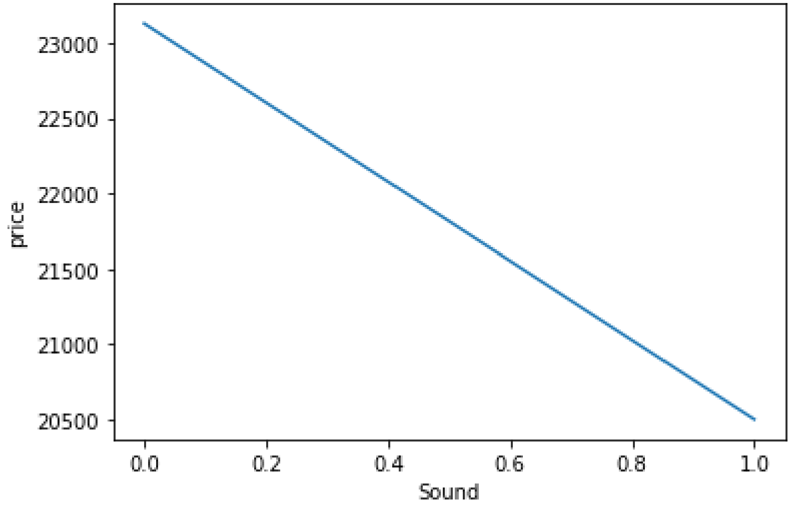
```
R2 SCORE : 0.45249864349981517
```

```
def plotting_with_one_feature():
    for i in ('Mileage', 'Cylinder', 'Liter', 'Doors', 'Cruise', 'Sound', 'Leather'):
      data.groupby(data[i])['Price'].mean().plot()
      #plt.title(i,' vs Price')
      plt.xlabel(i)
      plt.ylabel('price')
      plt.show()
      print(data[[i,'Price']].corr())
plotting_with_one_feature()
```

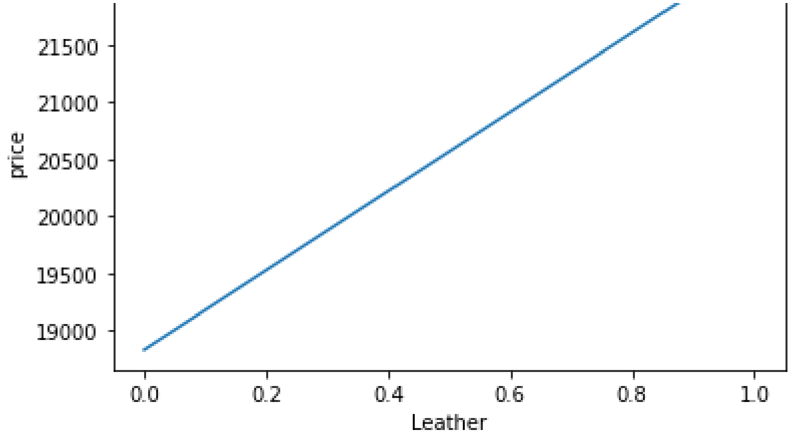
```
        Doors    Price
Doors  1.00000 -0.13875
Price -0.13875  1.00000
```




```
          Cruise     Price
Cruise  1.000000  0.430851
Price   0.430851  1.000000
```



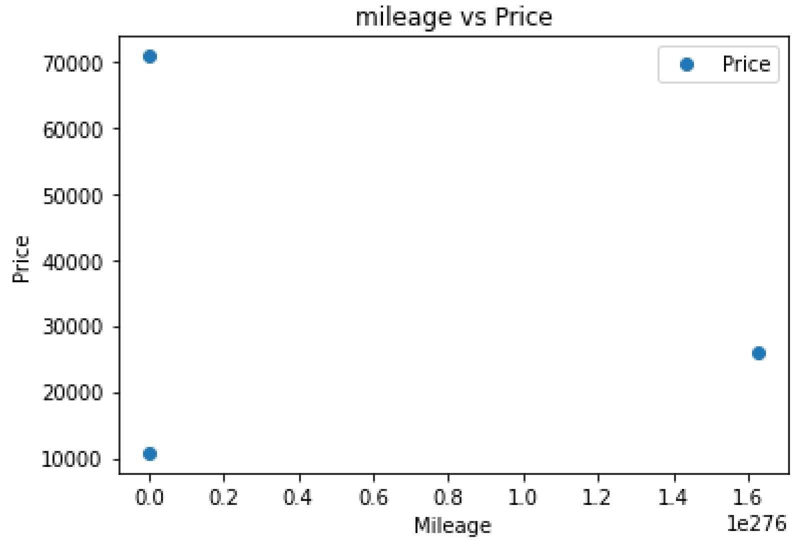

```
          Sound     Price
Sound  1.000000 -0.124348
Price -0.124348  1.000000
```




| | Leather | Price |
|---|---|---|
| Leather | 1.000000 | 0.157197 |
| Price | 0.157197 | 1.000000 |

```
data['transformed'] = np.exp(data['Mileage'])
data.plot(x='transformed', y='Price', style='o')
plt.title('mileage vs Price')
plt.xlabel('Mileage')
plt.ylabel('Price')
plt.show()
data[['Mileage','Price']].corr()
```

```
/usr/local/lib/python3.7/dist-packages/pandas/core/arraylike.py:364: RuntimeWarning: overflow encountered in exp
  result = getattr(ufunc, method)(*inputs, **kwargs)
```




| | Mileage | Price |
|---|---|---|
| **Mileage** | 1.000000 | -0.143051 |
| **Price** | -0.143051 | 1.000000 |

# Part 3

```
feature = ['Cylinder', 'Liter', 'Cruise']
```

```
x=data[feature]
y=data['Price']
```

```
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)
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()
scaler.fit(X_train,y_train)
X_train_scaled = scaler.transform(X_train)
X_test_scaled = scaler.transform(X_test)

regressor = linear_model.LinearRegression()
regressor.fit(X_train_scaled, y_train)
```

```
LinearRegression()
```

```
#coeff_df = pd.DataFrame(regressor.coef_,['Cylinder', 'Liter', 'Cruise'],columns=['Coefficients'])
y_pred = regressor.predict(X_test_scaled)
#coeff_df
y_pred
```

```
array([18511.34228319, 19019.83874158, 15435.45821194, 15508.6721542 ,
       14305.62384485, 17135.84247124, 20720.3607655 , 26847.46246052,
       25354.39339314, 18284.03717246, 21307.15050394, 24017.44653223,
       14579.3722882 , 23465.64906618, 30255.60540774, 24825.73045749,
       18810.22103015, 12382.70094987, 32791.53441979,  9862.77487276,
       21987.23958571, 25525.49839028, 10216.98560329, 22690.39012571,
       17355.09156061, 14218.71756973, 17940.52744322, 32726.64200238,
       14565.14062025, 21276.77761722, 19805.74420572, 17286.8403185 ,
       11302.75190782, 24597.02096077, 14679.61941158, 17918.47114211,
       17640.3975293 , 16786.63726   , 23683.41439564, 35239.37715663,
       17125.71817566, 20755.04892574, 16404.46661411, 22469.95074631,
       18472.91337511, 22180.13584381, 12257.75513286, 25383.89449048,
```

```
       23132.70353374, 33976.74089007, 25416.74138653, 20698.76465266,
       23545.31007135,  8879.25559675, 33469.53027906, 17276.08886073,
       19845.93250025, 26429.39441652, 20656.70600639, 30388.09303957,
       10820.80752394, 20843.06957181, 16617.60983523, 37060.917842  ,
       18526.26560617, 10123.00348906, 22349.4184495 , 21646.99686563,
       27472.77893622,  8543.48933669, 21038.98158637, 19503.53544459,
       15187.70181196, 33928.44302102, 21133.57221628,  9770.52076573,
       33198.56702478, 24664.79718938, 23763.53901645, 23017.03658046,
       16959.45353725, 19915.931953  , 17026.25846429, 20198.39421283,
       22872.23932807, 22812.35110315, 20833.91834035, 15808.47314701,
       17958.39999353, 26210.46507602, 15950.08578719, 21911.45476921,
       33730.39847167, 35305.34721896, 20554.12638012, 33278.16449553,
       16615.85201172, 20291.54442117,  9630.653035  , 33067.40716942,
       26050.77022183, 11218.29503267, 12323.97798423, 16442.00061127,
       16997.04738198, 17399.81324462, 29114.97272477, 37702.55077832,
       29952.37445672, 12517.83334865, 11884.78477414, 23343.28014698,
       32536.39343667, 33868.52710788, 17855.82926133, 12878.03464484,
       10511.75881049, 28086.12750522, 24077.51284589, 31959.37770638,
       21466.98159635, 14535.96537392, 16527.77328063, 34847.00151054,
       29071.33764491, 21170.01175897, 22166.92769644, 23005.34891904,
       19106.8622246 , 14299.17156985, 16752.37040641, 18623.50753288,
       21513.23147317, 10783.13186664, 12379.48226911, 22948.94610736,
       35462.68873049, 17168.82846261, 18507.34479149, 20002.87912783,
       15771.67567352, 27650.2848794 , 24652.47280876, 17222.85926339,
       22848.30410004, 17594.97361727, 23909.21717849, 22018.30419264,
       24449.62220121, 32772.07375985, 16230.46489171, 23097.46341155,
       34035.99291498, 18532.21783204, 18982.81485583, 27335.55992726,
       26419.92056571, 20334.72021275, 23229.37447208, 13334.91379301,
       18623.31937696])
```

```
regressor.intercept_

print('R2 SCORE :', metrics.r2_score(y_test,y_pred))
```

```
R2 SCORE : 0.4357811899013233
```

# Task Dummie Variables

```
dummies = pd.get_dummies(data[['Make','Model','Trim','Type']])
dummies.head()
```

| | Make_Buick | Make_Cadillac | Make_Chevrolet | Make_Pontiac | Make_SAAB | Make_Saturn | Model_9-2X AWD | Model_9_3 | Model_9_3 HO | Model_9_5 |
|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| **1** | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| **2** | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| **3** | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| **4** | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

5 rows × 90 columns

```
merged = pd.concat([data,dummies],axis='columns')
merged.head()
```

| | Price | Mileage | Make | Model | Trim | Type | Cylinder | Liter | Doors | Cruise | ... | Trim_SVM Hatchback 4D | Trim_SVM Sedan 4D | Trim_Sedan 4D | Tri Ed |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 17314.103129 | 8221 | Buick | Century | Sedan 4D | Sedan | 6 | 3.1 | 4 | 1 | ... | 0 | 0 | 1 | |
| **1** | 17542.036083 | 9135 | Buick | Century | Sedan 4D | Sedan | 6 | 3.1 | 4 | 1 | ... | 0 | 0 | 1 | |
| | | | | | Sedan | | | | | | | | | | |

```
final = merged.drop(['Make','Model','Trim','Type'],axis='columns')
final.head()
```

| | Price | Mileage | Cylinder | Liter | Doors | Cruise | Sound | Leather | transformed | Make_Buick | ... | Trim_SVM Hatchback 4D | Trim_SVM Sedan 4D | Tri |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 17314.103129 | 8221 | 6 | 3.1 | 4 | 1 | 1 | 1 | inf | 1 | ... | 0 | 0 | |
| **1** | 17542.036083 | 9135 | 6 | 3.1 | 4 | 1 | 1 | 0 | inf | 1 | ... | 0 | 0 | |
| **2** | 16218.847862 | 13196 | 6 | 3.1 | 4 | 1 | 1 | 0 | inf | 1 | ... | 0 | 0 | |
| **3** | 16336.913140 | 16342 | 6 | 3.1 | 4 | 1 | 0 | 0 | inf | 1 | ... | 0 | 0 | |
| **4** | 16339.170324 | 19832 | 6 | 3.1 | 4 | 1 | 0 | 1 | inf | 1 | ... | 0 | 0 | |

5 rows × 99 columns

```
X = final.drop('Price',axis='columns')
X.head()
```

| | Mileage | Cylinder | Liter | Doors | Cruise | Sound | Leather | transformed | Make_Buick | Make_Cadillac | ... | Trim_SVM Hatchback 4D | Trim_SVM Sedan 4D | Tr |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 8221 | 6 | 3.1 | 4 | 1 | 1 | 1 | inf | 1 | 0 | ... | 0 | 0 | |
| **1** | 9135 | 6 | 3.1 | 4 | 1 | 1 | 0 | inf | 1 | 0 | ... | 0 | 0 | |
| **2** | 13196 | 6 | 3.1 | 4 | 1 | 1 | 0 | inf | 1 | 0 | ... | 0 | 0 | |
| **3** | 16342 | 6 | 3.1 | 4 | 1 | 0 | 0 | inf | 1 | 0 | ... | 0 | 0 | |
| **4** | 19832 | 6 | 3.1 | 4 | 1 | 0 | 1 | inf | 1 | 0 | ... | 0 | 0 | |

5 rows × 98 columns

```
y = final['Price']
y.head()
```

```
0    17314.103129
1    17542.036083
2    16218.847862
3    16336.913140
4    16339.170324
Name: Price, dtype: float64
```

```
X_train,X_test,y_train,y_test = train_test_split(X,y,test_size=0.30,random_state=42)

ss = StandardScaler()
X_train_scaled = ss.fit_transform(X_train == True)
X_test_scaled = ss.transform(X_test == True)
```

```
regressor = linear_model.LinearRegression()
regressor.fit(X_train_scaled,y_train)
```

```
y_predict = regressor.predict(X_test_scaled)
print("R2 Score :", metrics.r2_score(y_test,y_predict))
```

```
R2 Score : 0.9561037411357705
```

# Find the best combination of factors to predict price.

```
dff = pd.read_csv('car_data.csv')
to_encode = ['Make','Model','Type','Trim']
def combinations(Ft):
  if Ft:
    result = combinations(Ft[:-1])
    return result + [i+[Ft[-1]] for i in result]
  else:
    return [[]]
comb = combinations(['Mileage','Cylinder','Liter','Doors','Cruise','Sound','Leather','Make','Model','Trim','Type'])
comb = comb[1:]
print(comb)
```

```
[['Mileage'], ['Cylinder'], ['Mileage', 'Cylinder'], ['Liter'], ['Mileage', 'Liter'], ['Cylinder', 'Liter'], ['Mileage', 'Cylin
```

```
R2_Score = []
for i in comb:
  X = dff[i]
  y = dff['Price'].values
  X = pd.get_dummies(X,columns=[j for j in to_encode if j in X.columns])
  X_train,X_test,y_train,y_test = train_test_split(X,y,test_size=0.30,random_state=20)

  if 'Mileage' in X_train:
    scaler = StandardScaler()
    scaler.fit(X_train)
    X_train_scaled = scaler.transform(X_train)
    X_test_scaled = scaler.transform(X_test)
  regressor = linear_model.LinearRegression()
```

```
  regressor.fit(X_train_scaled,y_train)
  y_predict = regressor.predict(X_test_scaled)
  R2_Score.append(metrics.r2_score(y_test,y_predict))
```

```
dff_snew = pd.DataFrame({'Feature Combination':comb,'R2 Score':R2_Score})
print(dff_snew.shape)
dff_snew.head()
```

(2047, 2)

| | Feature Combination | R2 Score |
|---|---|---|
| **0** | [Mileage] | 0.039431 |
| **1** | [Cylinder] | 0.039431 |
| **2** | [Mileage, Cylinder] | 0.298457 |
| **3** | [Liter] | 0.298457 |
| **4** | [Mileage, Liter] | 0.309513 |

```
dff_snew['R2 Score'].max()
```

0.9921621787561907

```
dff_snew['Feature Combination'][dff_snew['R2 Score'].argmax()]
```

['Mileage', 'Cylinder', 'Leather', 'Make', 'Model', 'Trim', 'Type']

```
# Vizulaizing Actual & Predicted Price for the  best combination of factors ['Mileage', 'Cylinder', 'Leather', 'Make', 'Model', 'Trim

X = data[['Mileage', 'Cylinder', 'Leather', 'Make', 'Model', 'Trim', 'Type']]
y = data['Price'].values
X = pd.get_dummies(X,columns=[j for j in to_encode if j in X.columns])
X_train,X_test,y_train,y_test = train_test_split(X,y,test_size=0.30,random_state=200)
regressor = linear_model.LinearRegression()
```