Importing Important Libraries

In [ ]:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
plt.style.use('seaborn')
```

Reading Data Files

In [ ]:

```
df = pd.read_csv('imports-85.data')
df.columns = ['symboling','normalized-losses','make','fuel-type','aspiration','num-o
```

In [ ]:

```
df.head()
```

Out[ ]:

	symboling	normalized-losses	make	fuel-type	aspiration	num-of-doors	body-style	drive-wheels	engine-location	wheel-base
0	3	?	alfa-romero	gas	std	two	convertible	rwd	front	88.6
1	1	?	alfa-romero	gas	std	two	hatchback	rwd	front	94.5
2	2	164	audi	gas	std	four	sedan	fwd	front	99.8
3	2	164	audi	gas	std	four	sedan	4wd	front	99.4
4	2	?	audi	gas	std	two	sedan	fwd	front	99.8

5 rows × 26 columns

In [ ]:

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 204 entries, 0 to 203
Data columns (total 26 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
 0   symboling          204 non-null    int64
 1   normalized-losses  204 non-null    object
 2   make               204 non-null    object
 3   fuel-type          204 non-null    object
 4   aspiration         204 non-null    object
 5   num-of-doors       204 non-null    object
 6   body-style         204 non-null    object
 7   drive-wheels       204 non-null    object
 8   engine-location    204 non-null    object
 9   wheel-base         204 non-null    float64
 10  length             204 non-null    float64
 11  width              204 non-null    float64
 12  height             204 non-null    float64
 13  curb-weight        204 non-null    int64
 14  engine-type        204 non-null    object
 15  num-of-cylinders   204 non-null    object 
 16  engine-size        204 non-null    int64  
 17  fuel-system        204 non-null    object 
 18  bore               204 non-null    object 
 19  stroke             204 non-null    object 
 20  compression-ratio  204 non-null    float64
 21  horsepower         204 non-null    object 
 22  peak-rpm           204 non-null    object 
 23  city-mpg           204 non-null    int64  
 24  highway-mpg        204 non-null    int64  
 25  price              204 non-null    object 
dtypes: float64(5), int64(5), object(16)
memory usage: 41.6+ KB
```

In [ ]:

```
columns_to_drop = ['bore','compression-ratio','peak-rpm','wheel-base','curb-weight',
df = df.drop(columns_to_drop,axis=1)
```

In [ ]:

```
df.head()
```

Out[ ]:

	symboling	make	fuel-type	aspiration	num-of-doors	body-style	drive-wheels	engine-location	length	width	height
0	3	alfa-romero	gas	std	two	convertible	rwd	front	168.8	64.1	48.8
1	1	alfa-romero	gas	std	two	hatchback	rwd	front	171.2	65.5	52.4
2	2	audi	gas	std	four	sedan	fwd	front	176.6	66.2	54.3
3	2	audi	gas	std	four	sedan	4wd	front	176.6	66.4	54.3
4	2	audi	gas	std	two	sedan	fwd	front	177.3	66.3	53.1

Cleaning the DataFrame

In [ ]:

```
df_clean = df.replace(to_replace='?',value = np.nan)
```

In [ ]:

```
df_clean = df_clean.fillna(value = df_clean.median())
```

In [ ]:

```
df_clean = df_clean.dropna()
```

In [ ]:

```
df_clean.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 202 entries, 0 to 203
Data columns (total 20 columns):
 #   Column            Non-Null Count  Dtype 
---  ------            --------------  ----- 
 0   symboling         202 non-null    int64 
 1   make              202 non-null    object
 2   fuel-type         202 non-null    object
 3   aspiration        202 non-null    object
 4   num-of-doors      202 non-null    object
 5   body-style         202 non-null    object 
 6   drive-wheels       202 non-null    object 
 7   engine-location    202 non-null    object 
 8   length             202 non-null    float64
 9   width              202 non-null    float64
 10  height             202 non-null    float64
 11  engine-type        202 non-null    object 
 12  num-of-cylinders   202 non-null    object 
 13  engine-size        202 non-null    int64  
 14  fuel-system        202 non-null    object 
 15  stroke             202 non-null    object 
 16  horsepower         202 non-null    object 
 17  city-mpg           202 non-null    int64  
 18  highway-mpg        202 non-null    int64  
 19  price              202 non-null    object 
dtypes: float64(3), int64(4), object(13)
memory usage: 33.1+ KB
```

In [ ]:

```
df_clean['price'] = df_clean['price'].astype('float64')
```

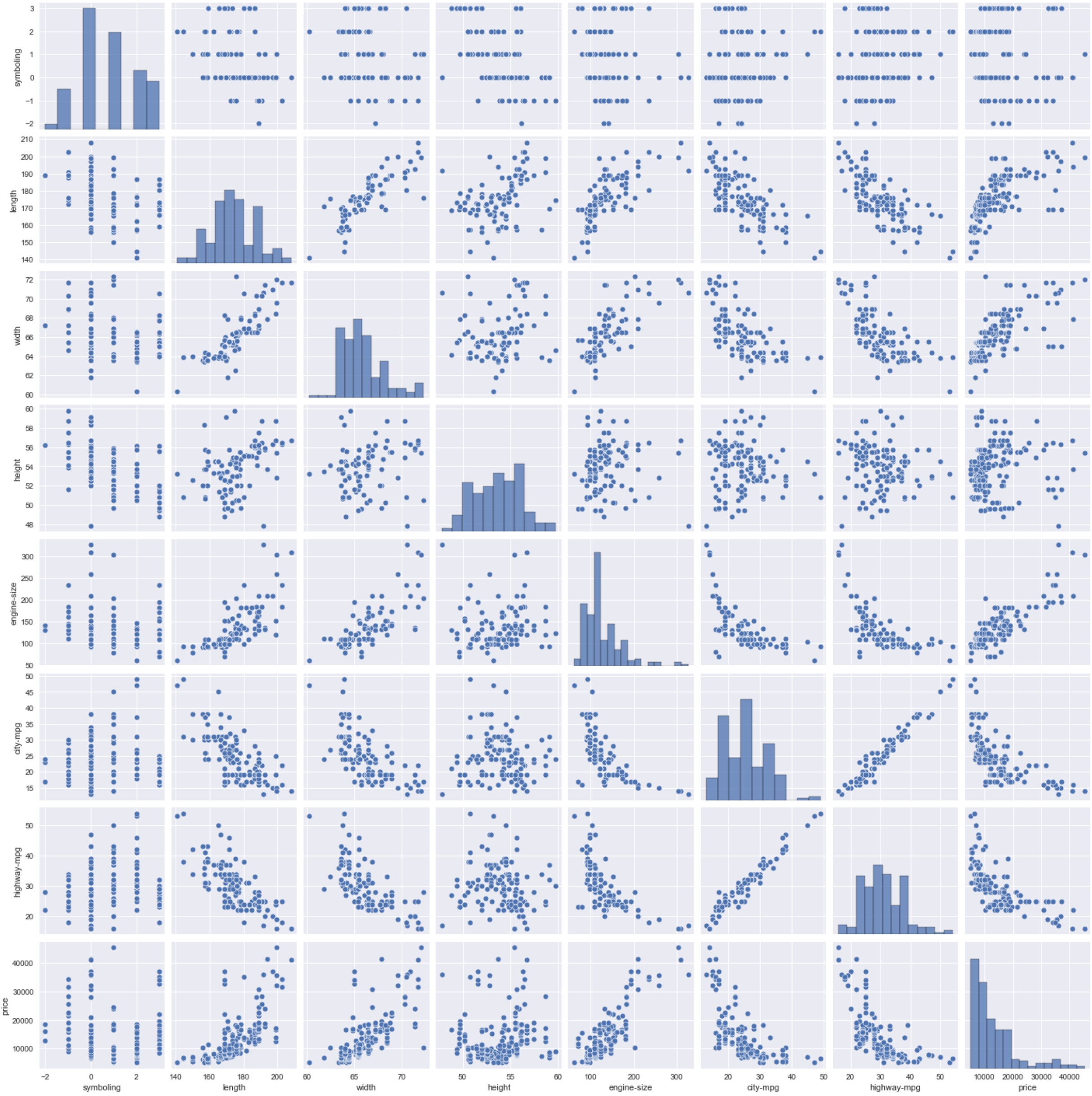
Visualizing Data to make a Co-relation Matrix

In [ ]:

```
import seaborn as sns
```

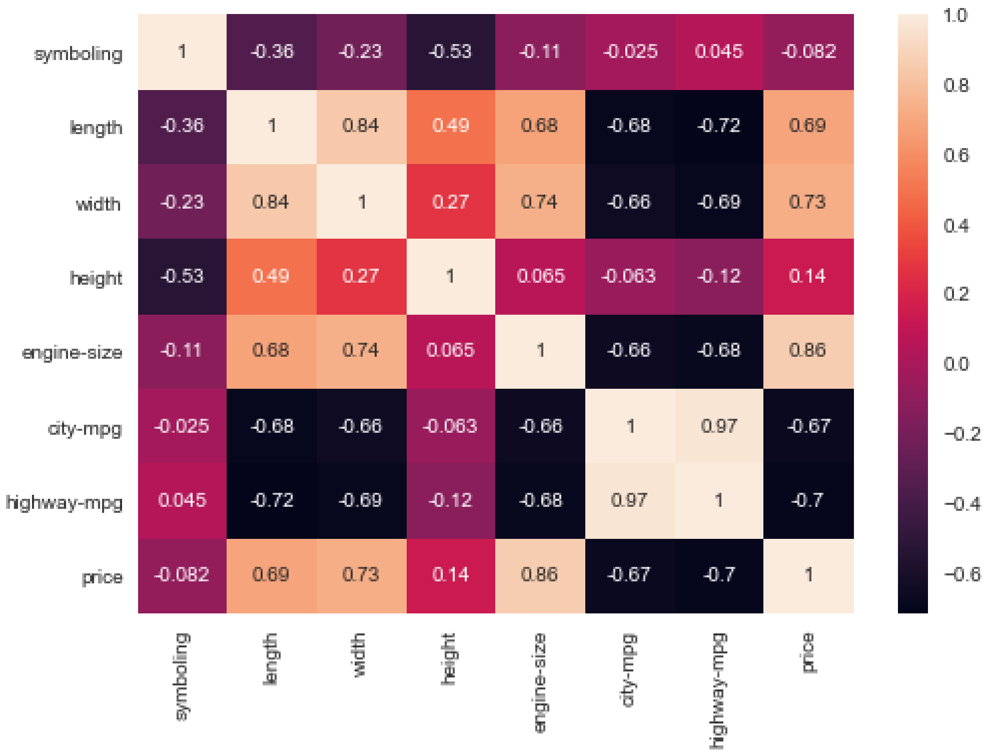
In [ ]:

```
sns.pairplot(df_clean)
plt.show()
```



In [ ]:

```
sns.heatmap(df_clean.corr(),annot=True)
plt.show()
```



Encoding Class Data to create Continuous Regression Data

In [ ]:

```
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
columns_to_encode = ['make','fuel-type','aspiration','num-of-doors','body-style','dr
for i in columns_to_encode:
    df_clean[i] = le.fit_transform(df_clean[i].astype(str))
```

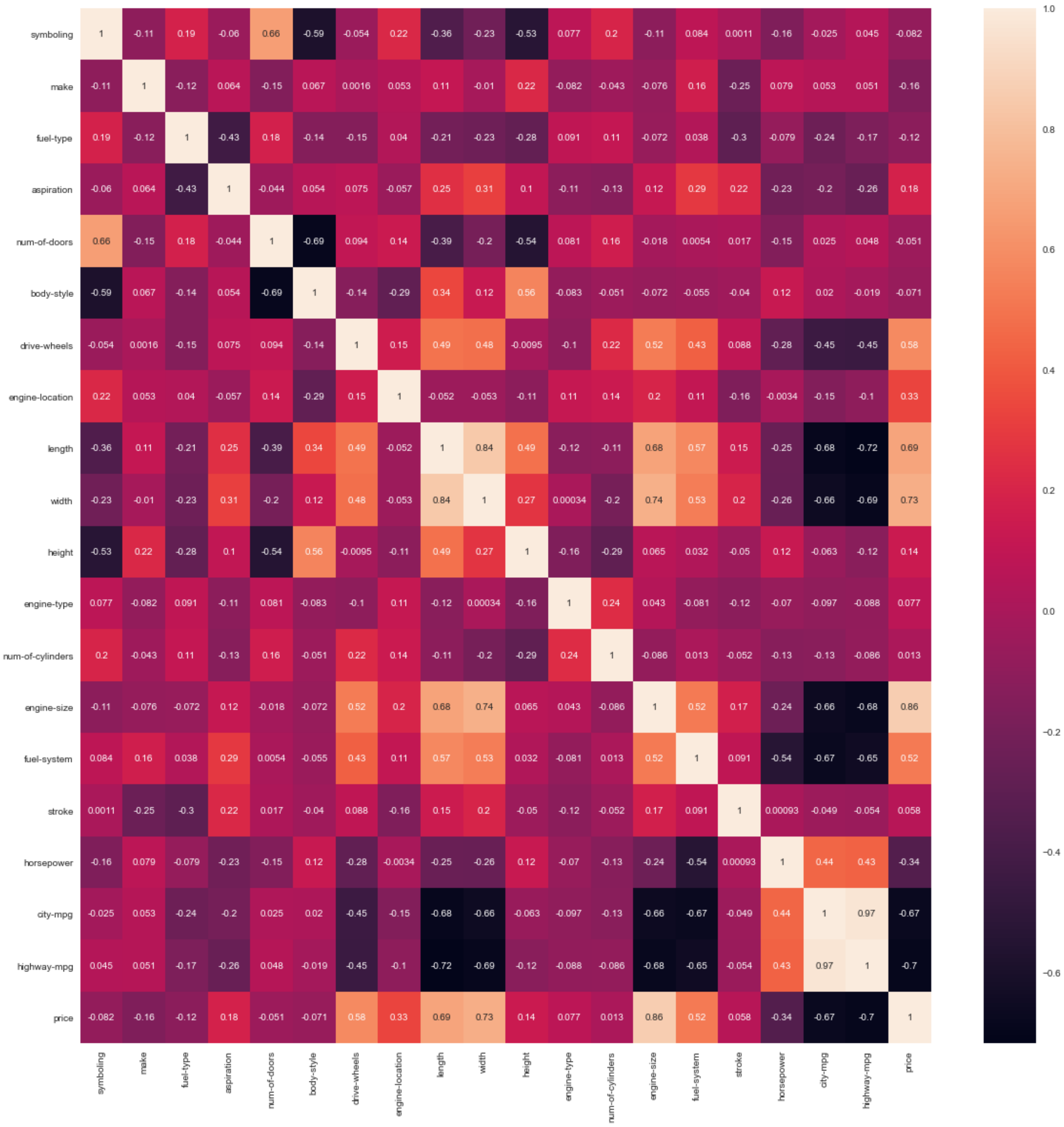
In [ ]:

```
df_clean.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 202 entries, 0 to 203
Data columns (total 20 columns):
 #   Column            Non-Null Count  Dtype
---  ------            --------------  -----
 0   symboling         202 non-null    int64
 1   make              202 non-null    int32
 2   fuel-type         202 non-null    int32
 3   aspiration        202 non-null    int32
 4   num-of-doors      202 non-null    int32
 5   body-style        202 non-null    int32
 6   drive-wheels      202 non-null    int32
 7   engine-location   202 non-null    int32
 8   length            202 non-null    float64
 9   width             202 non-null    float64
 10  height            202 non-null    float64
 11  engine-type       202 non-null    int32
 12  num-of-cylinders  202 non-null    int32
 13  engine-size       202 non-null    int64
 14  fuel-system       202 non-null    int32
 15  stroke            202 non-null    int32
 16  horsepower        202 non-null    int32
 17  city-mpg          202 non-null    int64
 18  highway-mpg       202 non-null    int64
 19  price              202 non-null    float64
dtypes: float64(4), int32(12), int64(4)
memory usage: 23.7 KB
```

In [ ]:

```
plt.figure(figsize=(20,20))
sns.heatmap(df_clean.corr(),annot=True)
plt.show()
```



Purging Features with very low correlation

In [ ]:

```
df_clean = df_clean.drop(['symboling', 'body-style', 'num-of-doors', 'stroke','num-o
```

In [ ]:

```
df_clean.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 202 entries, 0 to 203
Data columns (total 14 columns):
 #   Column             Non-Null Count  Dtype  
---  ------             --------------  -----  
```

```
 0   make              202 non-null    int32  
 1   fuel-type         202 non-null    int32  
 2   aspiration        202 non-null    int32  
 3   drive-wheels      202 non-null    int32  
 4   engine-location   202 non-null    int32  
 5   length            202 non-null    float64
 6   width             202 non-null    float64
 7   height            202 non-null    float64
 8   engine-size       202 non-null    int64  
 9   fuel-system       202 non-null    int32  
 10  horsepower        202 non-null    int32  
 11  city-mpg          202 non-null    int64  
 12  highway-mpg       202 non-null    int64  
 13  price             202 non-null    float64
dtypes: float64(4), int32(7), int64(3)
memory usage: 18.1 KB
```

In [ ]:

```
X,Y = df_clean.values[:,:-1],df_clean.values[:,-1]
```

In [ ]:

```
X.shape,Y.shape
```

Out[ ]:

```
((202, 13), (202,))
```

Data Normalization

In [ ]:

```
u = np.mean(X, axis=0)
# print(u.shape)
std = np.std(X, axis=0)
X = (X-u)/std
```

In [ ]:

```
ones = np.ones((X.shape[0],1))
X = np.hstack((ones,X))
print(X)
```

```
[[ 1.         -1.98265203  0.32221908 ... -1.46925631 -0.64279561
  -0.54119201]
 [ 1.         -1.98265203  0.32221908 ... -0.63760179 -0.9493947
  -0.68675896]
 [ 1.         -1.82168108  0.32221908 ... -1.63558721 -0.18289698
  -0.10449113]
 ...
 [ 1.          1.3977378   0.32221908 ... -1.02570723 -1.10269424
  -1.12345984]
 [ 1.          1.3977378  -3.10347852 ... -1.58014358  0.12370211
  -0.54119201]
 [ 1.          1.3977378   0.32221908 ... -1.35836904 -0.9493947
  -0.83232592]]
```

Splitting the Data into Test Train Split

In [ ]:

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=0.1, random_stat
```

Multiple Linear Regression

In [ ]:

```
def r2_score(y,y_):
    num = np.sum((y-y_)**2)
    denom = np.sum((y-y.mean())**2)
    score = 1-(num/denom)
    return score*100

def train_val_data(X,Y):
    train_X,val_X,train_Y,val_Y = train_test_split(X, Y, test_size=0.05,shuffle=True
    return train_X,val_X,train_Y,val_Y
```

In [ ]:

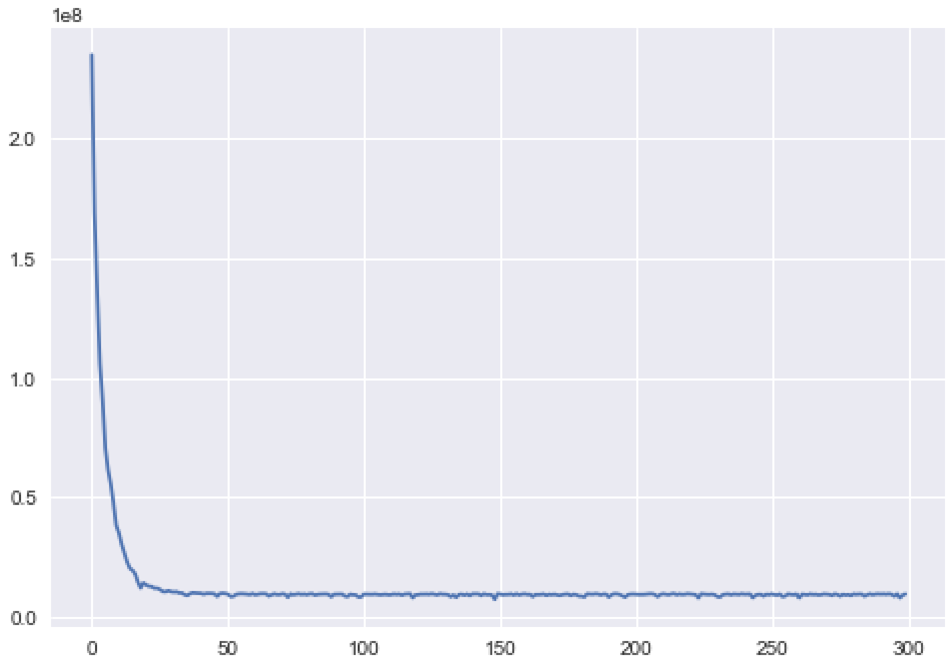
```
def hypothesis(X,theta):
    return np.dot(X,theta)

def error(X,y,theta):
    e = 0.0
    m = X.shape[0]
    y_ = hypothesis(X,theta)
    e = (np.sum((y-y_)**2))
    return e/m
def gradient(X,y,theta):
    y_ = hypothesis(X,theta)
    grad = np.dot(X.T,(y_-y))
    m = X.shape[0]
    return grad/m
def gradient_Descent(X,y,learning_rate = 0.1,max_iters = 300):
    n = X.shape[1]
    theta = np.zeros((n,))
    error_list = []
    val_score = []

    for i in range(max_iters):
        train_X,val_X,train_Y,val_Y = train_val_data(X,y)

        e = error(train_X,train_Y,theta)
        error_list.append(e)

        grad = gradient(train_X,train_Y,theta)
        theta = theta - learning_rate*grad

        y_ = hypothesis(val_X,theta)
        val_score.append(r2_score(val_Y,y_))
    return theta,error_list,val_score
```

In [ ]:

```
theta,error_list,val_score = gradient_Descent(X_train,y_train)
```

In [ ]:

```
plt.plot(error_list)
plt.show()
```



In [ ]:

```
val_score = pd.DataFrame(np.array(val_score),columns=["Val Score"])
print(val_score.describe())
```

```
          Val Score
count   300.000000
mean     53.485459
std     111.013255
min   -1539.427710
25%      55.857035
50%      78.884615
75%      89.711316
max      98.182104
```

In [ ]:

```
y_ = hypothesis(X_test,theta)
print("Overall R2-Score: ",r2_score(y_test,y_))
```

```
Overall R2-Score:  93.73951859962185
```

Ridge Regression

In [ ]:

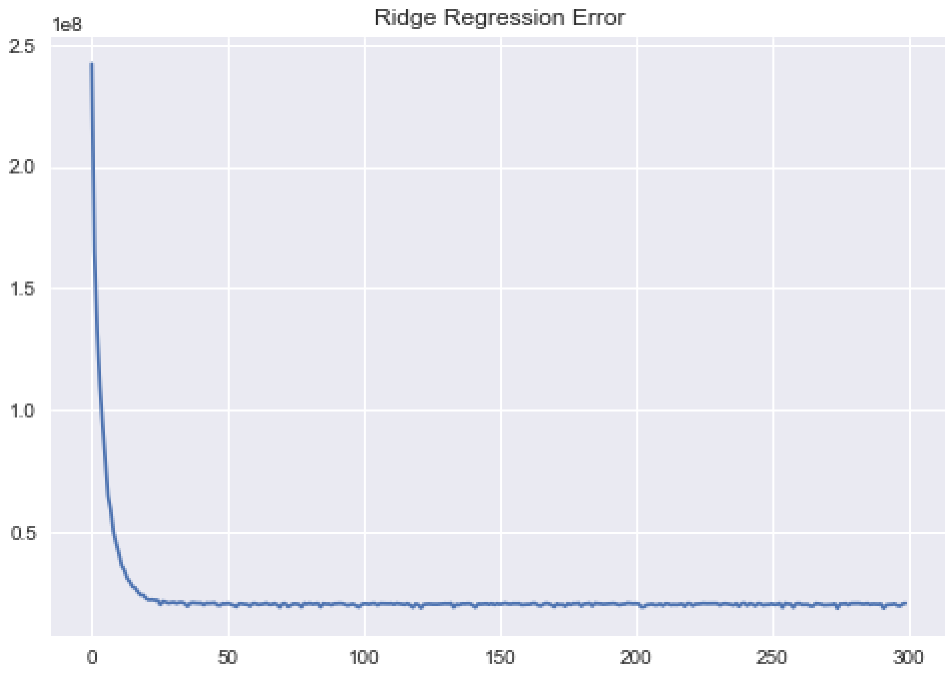
```
def hypothesis_r(X,theta):
    return np.dot(X,theta)

def error_r(X,y,theta, lam):
    e = 0.0
    m = X.shape[0]
    y_ = hypothesis_r(X,theta)
    e = np.sum((y-y_)**2) + lam*np.sum(np.square(theta))
    return e/m
def gradient_r(X,y,theta,lam):
    y_ = hypothesis_r(X,theta)
    grad = np.dot(X.T,(y_-y)) + lam*theta
    m = X.shape[0]
    return grad/m
def gradient_Descent_r(X,y,lam,learning_rate = 0.1,max_iters = 300):
    n = X.shape[1]
    theta = np.zeros((n,))
    error_list = []
    val_score = []
    for i in range(max_iters):
        train_X,val_X,train_Y,val_Y = train_val_data(X,y)

        e = error_r(train_X,train_Y,theta,lam)
        error_list.append(e)

        grad = gradient_r(train_X,train_Y,theta,lam)
        theta = theta - learning_rate*grad

        y_ = hypothesis_r(val_X,theta)
        val_score.append(r2_score(val_Y,y_))
    return theta,error_list,val_score
```

In [ ]:

```
theta,error_list,val_score = gradient_Descent_r(X_train,y_train,lam=10)
```

In [ ]:

```
plt.plot(error_list)
plt.title("Ridge Regression Error")
plt.show()
```



In [ ]:

```
val_score = pd.DataFrame(np.array(val_score),columns=["Val Score Ridge Regression"])
print(val_score.describe())
```

```
       Val Score Ridge Regression
count                  300.000000
mean                    63.269598
std                     65.456889
min                   -643.204399
25%                     65.353973
50%                     82.525613
75%                     89.126761
max                     97.744985
```

In [ ]:

```
y_ = hypothesis_r(X_test,theta)
print("Overall R2-Score: ",r2_score(y_test,y_))
```

```
Overall R2-Score:  94.26890687581202
```

Lasso Regression

In [ ]:

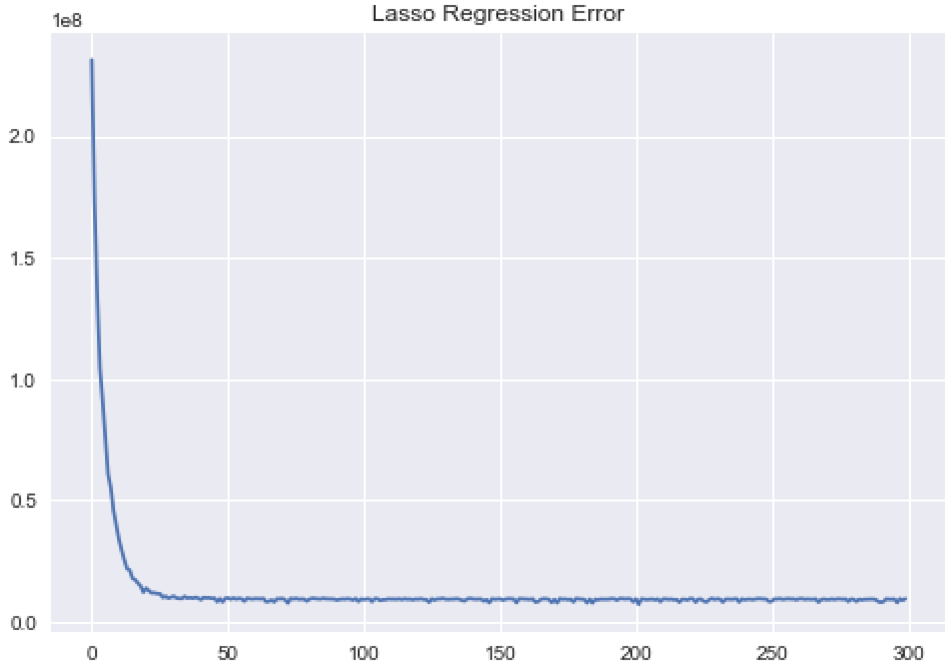
```
def hypothesis_l(X,theta):
    return np.dot(X,theta)

def error_l(X,y,theta, lam):
    e = 0.0
    m = X.shape[0]
    y_ = hypothesis_l(X,theta)
    e = np.sum((y-y_)**2) + lam*np.sum(np.abs(theta))
    return e/m
def gradient_l(X,y,theta,lam):
    y_ = hypothesis_l(X,theta)
    grad = np.dot(X.T,(y_-y)) + lam*(2*np.sinc(theta)-1)
    m = X.shape[0]
    return grad/m
def gradient_Descent_l(X,y,lam,learning_rate = 0.1,max_iters = 300):
    n = X.shape[1]
    theta = np.zeros((n,))
    error_list = []
    val_score = []

    for i in range(max_iters):
        train_X,val_X,train_Y,val_Y = train_val_data(X,y)

        e = error_l(train_X,train_Y,theta,lam)
        error_list.append(e)

        grad = gradient_l(train_X,train_Y,theta,lam)
        theta = theta - learning_rate*grad

        y_ = hypothesis_l(val_X,theta)
        val_score.append(r2_score(val_Y,y_))
    return theta,error_list,val_score
```

In [ ]:

```
theta,error_list,val_score = gradient_Descent_l(X_train,y_train,lam=10)
```

In [ ]:

```
plt.plot(error_list)
plt.title("Lasso Regression Error")
plt.show()
```



In [ ]:

```
val_score = pd.DataFrame(np.array(val_score),columns=["Val Score Lasso Regression"])
print(val_score.describe())
```

```
       Val Score Lasso Regression
count                  300.000000
mean                    65.963418
std                     47.818503
min                   -281.041242
25%                     63.649129
50%                     81.367128
75%                     90.137017
max                     99.064463
```

In [ ]:

```
y_ = hypothesis_r(X_test,theta)
print("Overall R2-Score: ",r2_score(y_test,y_))
```

```
Overall R2-Score:  93.77253605140649
```

In [ ]: