In [ ]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
```

In [ ]:
```
columns = np.array(["symboling","normailsed-losses","make","fuel-type","aspiration","nu
print(columns.shape)
```

```
(26,)
```

In [ ]:
```
data = pd.read_csv("imports-85.csv")
data.columns = columns
data.head()
```

Out[ ]:

	symboling	normailsed-losses	make	fuel-type	aspiration	num-of-doors	body-style	drive-wheels	engine-locations	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 [ ]:
```
data_clean = data.replace(to_replace='?',value = np.nan)
data_clean.head()
```

Out[ ]:

	symboling	normailsed-losses	make	fuel-type	aspiration	num-of-doors	body-style	drive-wheels	engine-locations	wheel-base	...
0	3	NaN	alfa-romero	gas	std	two	convertible	rwd	front	88.6	...
1	1	NaN	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	NaN	audi	gas	std	two	sedan	fwd	front	99.8	...

5 rows × 26 columns

In [ ]:
```
data_clean = data_clean.fillna(value = data_clean.median)
data_clean.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   normailsed-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-locations   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 [ ]:
```
from sklearn.preprocessing import LabelEncoder
label = LabelEncoder()
col = ['make','fuel-type','aspiration','num-of-doors','body-style','drive-wheels','engi
for i in col:
    data_clean[i] = label.fit_transform(data_clean[i].astype(str))
```

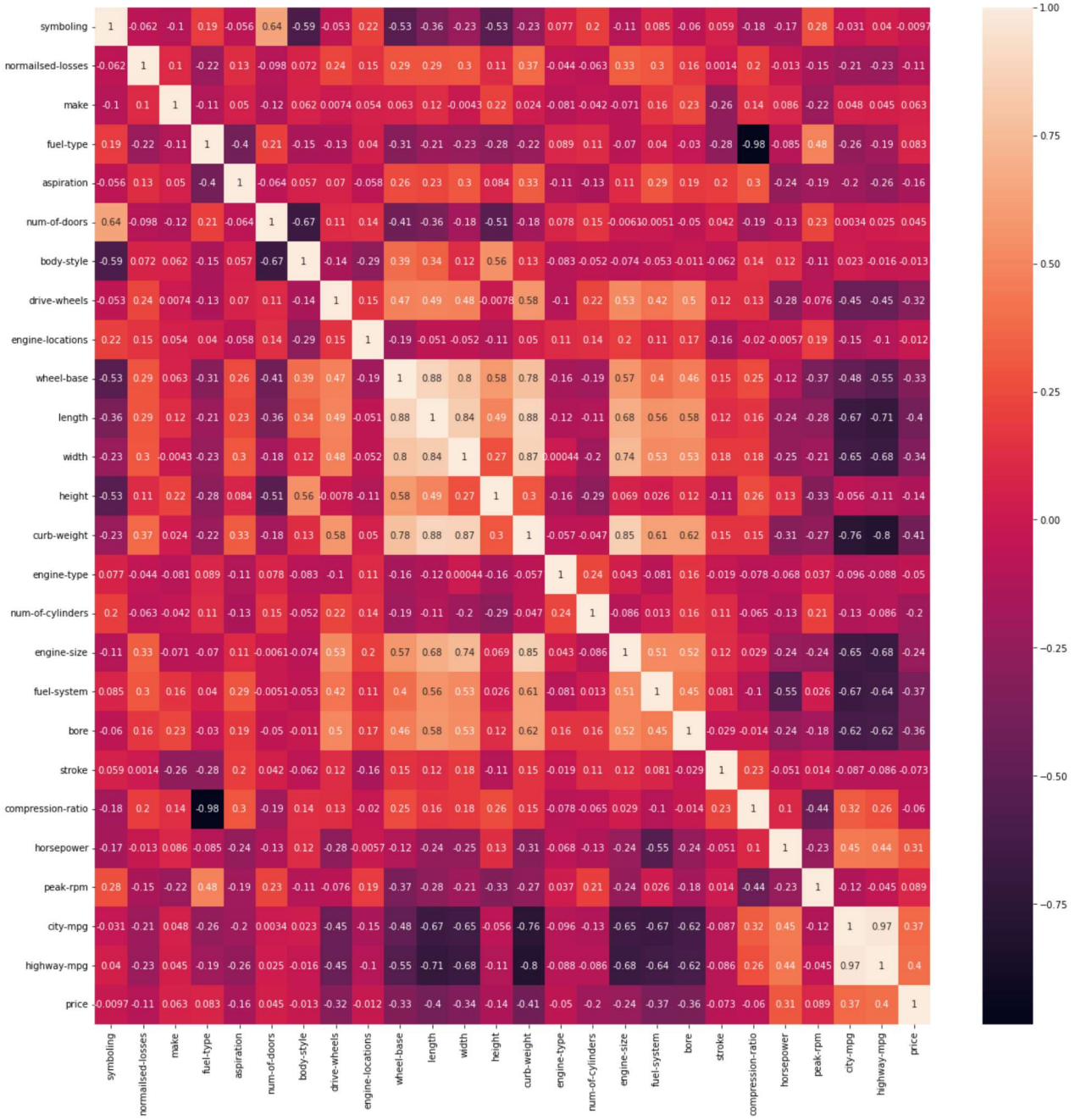
In [ ]:
```
data_clean.info()
```

```
<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   normailsed-losses  204 non-null    int32  
 2   make               204 non-null    int32  
 3   fuel-type          204 non-null    int32  
 4   aspiration         204 non-null    int32  
 5   num-of-doors       204 non-null    int32  
 6   body-style         204 non-null    int32  
 7   drive-wheels       204 non-null    int32  
 8   engine-locations   204 non-null    int32  
```

```
 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    int32
 15  num-of-cylinders   204 non-null    int32
 16  engine-size        204 non-null    int64
 17  fuel-system        204 non-null    int32
 18  bore               204 non-null    int32
 19  stroke             204 non-null    int32
 20  compression-ratio  204 non-null    float64
 21  horsepower         204 non-null    int32
 22  peak-rpm           204 non-null    int32
 23  city-mpg           204 non-null    int64
 24  highway-mpg        204 non-null    int64
 25  price              204 non-null    int32
dtypes: float64(5), int32(16), int64(5)
memory usage: 28.8 KB
```

In [ ]:

```
import seaborn as sns
plt.figure(figsize=(20,20))
sns.heatmap(data_clean.corr(),annot=True)
plt.show()
```



In [ ]:
```
drop_cols = ['symboling','stroke','num-of-doors','body-style','engine-type','compressio
data_clean = data_clean.drop(drop_cols,axis = 1)
```

In [ ]:
```
data_clean.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 204 entries, 0 to 203
Data columns (total 20 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
 0   normailsed-losses  204 non-null    int32
 1   make               204 non-null    int32
 2   fuel-type          204 non-null    int32
 3   aspiration         204 non-null    int32
 4   drive-wheels       204 non-null    int32
 5   engine-locations   204 non-null    int32
```

```
 6   wheel-base         204 non-null    float64
 7   length             204 non-null    float64
 8   width              204 non-null    float64
 9   height             204 non-null    float64
 10  curb-weight        204 non-null    int64
 11  num-of-cylinders   204 non-null    int32
 12  engine-size        204 non-null    int64
 13  fuel-system        204 non-null    int32
 14  bore               204 non-null    int32
 15  horsepower         204 non-null    int32
 16  peak-rpm           204 non-null    int32
 17  city-mpg           204 non-null    int64
 18  highway-mpg        204 non-null    int64
 19  price              204 non-null    int32
dtypes: float64(4), int32(12), int64(4)
memory usage: 22.4 KB
```

In [ ]:
```
X = data_clean.values[:,:-1]
Y = data_clean.values[:,-1]
print(X.shape,Y.shape)
```

```
(204, 19) (204,)
```

In [ ]:
```
u = np.mean(X,axis=0)
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.shape)
```

```
(204, 20)
```

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_state = 7)
```

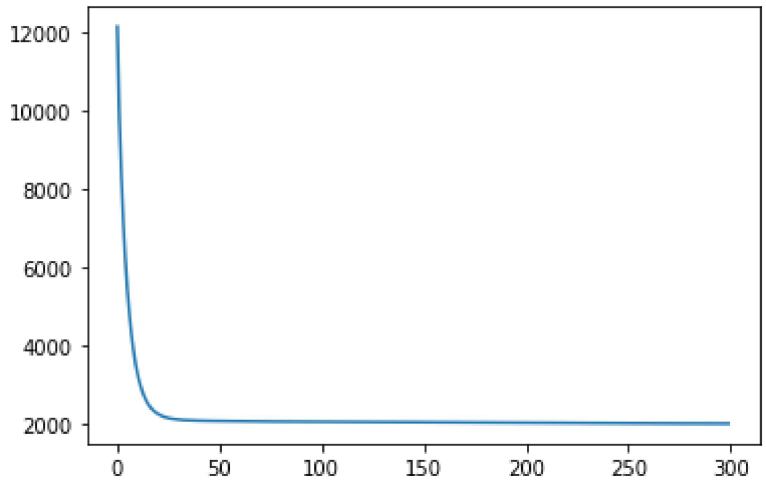
In [ ]:
```
def r2(Y,Y_):
    num = np.sum((Y-Y_)**2)
    denom = np.sum((Y-Y.mean())**2)
    score = 1-(num/denom)
    return score*100

def train_val_split(X,Y):
    X_train_new,X_val,Y_train_new,Y_val = train_test_split(X,Y,test_size = 0.05, shuffl
    return X_train_new,X_val,Y_train_new,Y_val
```

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

def error(X,y,theta):
    e = 0.0
    y_ = hypothesis(X,theta)
    e = np.sum((y_-y)**2)

    return e/X.shape[0]
```

```
def gradient(X,y,theta):
    m = X.shape[0]
    y_ = hypothesis(X,theta)
    grad = np.dot(X.T,(y_-y))

    return grad/m

def gradient_descent(X,y,lr=0.1):
    errors = []
    n = X.shape[1]
    theta = np.zeros((n,))
    val_scores = []

    for i in range(300):
        X_train_new,X_val,Y_train_new,Y_val = train_val_split(X,y)

        e = error(X,y,theta)
        errors.append(e)

        y_ = hypothesis(X_val,theta)
        score = r2(Y_val,y_)
        val_scores.append(score)

        grad = gradient(X_train_new,Y_train_new,theta)
        theta = theta - lr*grad

    return theta,errors,val_scores
```

In [ ]:

```
theta,errors,val_scores = gradient_descent(X_train,Y_train)
plt.plot(errors)
plt.show()
```



In [ ]:

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

```
        Val Score
count  300.000000
mean    19.133166
std     39.319306
min   -286.155864
```

```
25%     11.087503
50%     25.716911
75%     40.188743
max     80.213723
```

In [ ]:
```
y_ = hypothesis(X_test,theta)
print("R2-score: ",r2(Y_test,y_))
```

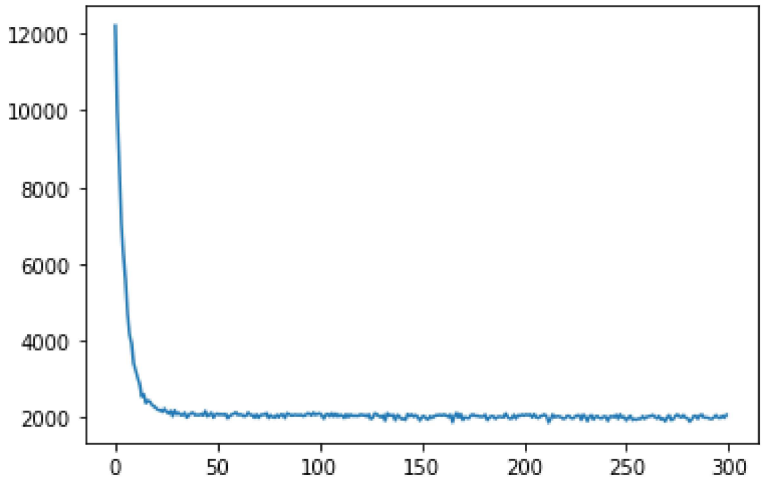
```
R2-score:  25.30631229492608
```

Lasso Regression

In [ ]:
```
def hypothesis_lasso(X,theta):
    return np.dot(X,theta)

def error_lasso(X,y,theta,lamda):
    e = 0.0
    y_ = hypothesis_lasso(X,theta)
    e = np.sum((y_-y)**2) + lamda*np.sum(np.abs(theta))

    return e/X.shape[0]

def gradient_lasso(X,y,theta,lamda):
    m = X.shape[0]
    y_ = hypothesis_lasso(X,theta)
    grad = np.dot(X.T,(y_-y)) + lamda*(2*np.sinc(theta)-1)

    return grad/m

def gradient_descent_lasso(X,y,lamda,lr=0.1):
    errors = []
    n = X.shape[1]
    theta = np.zeros((n,))
    val_scores = []

    for i in range(300):
        X_train_new,X_val,Y_train_new,Y_val = train_val_split(X,y)

        e = error_lasso(X_train_new,Y_train_new,theta,lamda)
        errors.append(e)

        y_ = hypothesis_lasso(X_val,theta)
        score = r2(Y_val,y_)
        val_scores.append(score)

        grad = gradient_lasso(X_train_new,Y_train_new,theta,lamda)
        theta = theta - lr*grad

    return theta,errors,val_scores
```

In [ ]:
```
theta_lasso,error_lasso,val_score_lasso = gradient_descent_lasso(X_train,Y_train,10)
```

In [ ]:
```
plt.plot(error_lasso)
plt.show()
```



In [ ]:
```
val_score_lasso = pd.DataFrame(np.array(val_score_lasso),columns=['Val Score Lasso'])
print(val_score_lasso.describe())
```

```
       Val Score Lasso
count       300.000000
mean         16.406488
std          52.492885
min        -428.959032
25%          11.039810
50%          27.642138
75%          39.491201
max          69.706456
```

In [ ]:
```
y_ = hypothesis_lasso(X_test,theta_lasso)
print("R2-score: ",r2(Y_test,y_))
```

```
R2-score:  25.196130078133372
```

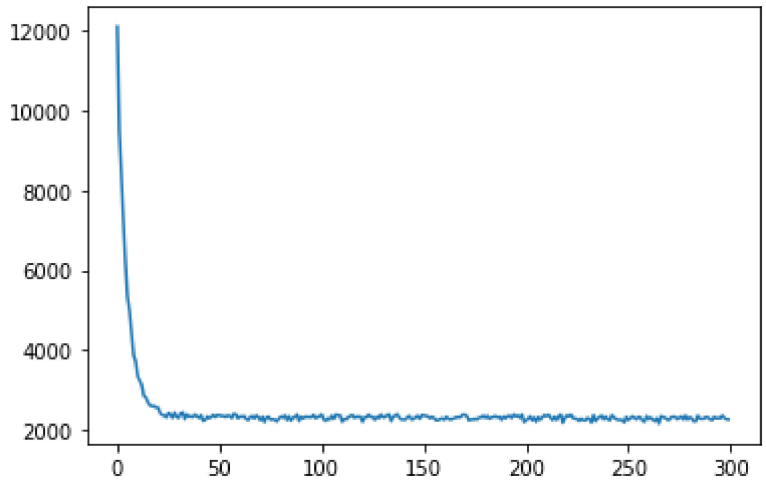
Ridge Regression

In [ ]:
```
def hypothesis_lasso(X,theta):
    return np.dot(X,theta)

def error_lasso(X,y,theta,lamda):
    e = 0.0
    y_ = hypothesis_lasso(X,theta)
    e = np.sum((y_-y)**2) + lamda*np.sum(np.square(theta))

    return e/X.shape[0]

def gradient_lasso(X,y,theta,lamda):
    m = X.shape[0]
    y_ = hypothesis_lasso(X,theta)
    grad = np.dot(X.T,(y_-y)) + lamda*theta

    return grad/m

def gradient_descent_lasso(X,y,lamda,lr=0.1):
    errors = []
    n = X.shape[1]
    theta = np.zeros((n,))
```

```
    val_scores = []

    for i in range(300):
        X_train_new,X_val,Y_train_new,Y_val = train_val_split(X,y)

        e = error_lasso(X_train_new,Y_train_new,theta,lamda)
        errors.append(e)

        y_ = hypothesis_lasso(X_val,theta)
        score = r2(Y_val,y_)
        val_scores.append(score)

        grad = gradient_lasso(X_train_new,Y_train_new,theta,lamda)
        theta = theta - lr*grad

    return theta,errors,val_scores
```

In [ ]:
```
theta_ridge,error_ridge,val_score_ridge = gradient_descent_lasso(X_train,Y_train,5)
```

In [ ]:
```
plt.plot(error_ridge)
plt.show()
```



In [ ]:
```
val_score_ridge = pd.DataFrame(np.array(val_score_ridge),columns=['Val Score Ridge'])
print(val_score_ridge.describe())
```

```
       Val Score Ridge
count       300.000000
mean          9.630181
std          79.710641
min       -1038.155673
25%           1.371537
50%          21.629123
75%          39.950043
max          72.980039
```

In [ ]:
```
y_ = hypothesis_lasso(X_test,theta_ridge)
print("R2-score: ",r2(Y_test,y_))
```

```
R2-score:  24.232403338944618
```

In [ ]: