Linear Regression model

when model learns too well from the training data and not perform well on testing dats it causes overfitting to reduce overfitting we have some techniques

cross validaaion

ensamble learning

Regularization

Regularization:

Regularization is a technique in machine learning it used to reduce overfitting by controlling size of the coefficients in a model

it adds penalty to the model if it uses more coefficients

it keeps the model simple better at predicting the model

Types of Regularization

i) Ridge regularization(L2):

it is a type of regularization it is used to prevent overfitting by reducing high coefficients to low

ii) Lasso regularization(L1):

it is a type of regularization it is used to prevent overfitting by reducing high coefficiets to zero

In [1]:
```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline
from sklearn import preprocessing
from sklearn.preprocessing import PolynomialFeatures
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression, Ridge, Lasso
from sklearn.metrics import r2_score
```

In [2]:
```
data=pd.read_csv(r"C:\Users\akshi\Downloads\car-mpg.csv")
```

In [3]:
```
data
```

Out[3]:

	mpg	cyl	disp	hp	wt	acc	yr	origin	car_type	car_name
0	18.0	8	307.0	130	3504	12.0	70	1	0	chevrolet chevelle malibu
1	15.0	8	350.0	165	3693	11.5	70	1	0	buick skylark 320
2	18.0	8	318.0	150	3436	11.0	70	1	0	plymouth satellite
3	16.0	8	304.0	150	3433	12.0	70	1	0	amc rebel sst
4	17.0	8	302.0	140	3449	10.5	70	1	0	ford torino
...	...	...	...	...	...	...	...	...	...	...
393	27.0	4	140.0	86	2790	15.6	82	1	1	ford mustang gl
394	44.0	4	97.0	52	2130	24.6	82	2	1	vw pickup
395	32.0	4	135.0	84	2295	11.6	82	1	1	dodge rampage
396	28.0	4	120.0	79	2625	18.6	82	1	1	ford ranger
397	31.0	4	119.0	82	2720	19.4	82	1	1	chevy s-10

398 rows × 10 columns

In [4]:

```
# 1. Drop unnecessary column
data = data.drop(['car_name'], axis=1)

# 2. Replace origin numbers with names
data['origin'] = data['origin'].replace({1: 'america', 2: 'europe', 3: 'asia'})

# 3. Create dummy variables
data = pd.get_dummies(data, columns=['origin'], dtype=int)

# 4. Replace '?' with NaN
data = data.replace('?', np.nan)

#  5. Convert all columns to numeric (errors='ignore' skips dummy columns which are
data = data.apply(pd.to_numeric, errors='ignore')

#  6. Fill missing numeric values with median (only numeric columns)
data = data.fillna(data.median(numeric_only=True))
```

```
C:\Users\akshi\AppData\Local\Temp\ipykernel_18292\1530094145.py:14: FutureWarning: e
rrors='ignore' is deprecated and will raise in a future version. Use to_numeric with
out passing `errors` and catch exceptions explicitly instead
  data = data.apply(pd.to_numeric, errors='ignore')
```

In [5]:

```
data.head(5)
```

Out[5]:

	mpg	cyl	disp	hp	wt	acc	yr	car_type	origin_america	origin_asia	origin_eur
0	18.0	8	307.0	130.0	3504	12.0	70	0	1	0	
1	15.0	8	350.0	165.0	3693	11.5	70	0	1	0	
2	18.0	8	318.0	150.0	3436	11.0	70	0	1	0	
3	16.0	8	304.0	150.0	3433	12.0	70	0	1	0	
4	17.0	8	302.0	140.0	3449	10.5	70	0	1	0	

In [6]:

```python
x=data.drop(['mpg'],axis=1)
y=data[['mpg']]
```

In [7]:

```python
x
```

Out[7]:

	cyl	disp	hp	wt	acc	yr	car_type	origin_america	origin_asia	origin_europe
0	8	307.0	130.0	3504	12.0	70	0	1	0	0
1	8	350.0	165.0	3693	11.5	70	0	1	0	0
2	8	318.0	150.0	3436	11.0	70	0	1	0	0
3	8	304.0	150.0	3433	12.0	70	0	1	0	0
4	8	302.0	140.0	3449	10.5	70	0	1	0	0
...	...	...	...	...	...	...	...	...	...	...
393	4	140.0	86.0	2790	15.6	82	1	1	0	0
394	4	97.0	52.0	2130	24.6	82	1	0	0	1
395	4	135.0	84.0	2295	11.6	82	1	1	0	0
396	4	120.0	79.0	2625	18.6	82	1	1	0	0
397	4	119.0	82.0	2720	19.4	82	1	1	0	0

398 rows × 10 columns

In [8]:

```python
y
```

Out[8]:

	mpg
0	18.0
1	15.0
2	18.0
3	16.0
4	17.0
...	...
393	27.0
394	44.0
395	32.0
396	28.0
397	31.0

398 rows × 1 columns

In [18]:
```
x_s=preprocessing.scale(x) # it replaces every value by the z-score
x_s=pd.DataFrame(x_s,columns=x.columns)

y_s=preprocessing.scale(y)
y_s=pd.DataFrame(y_s,columns=y.columns)
```

In [19]:
```
x_s
```

Out[19]:

	cyl	disp	hp	wt	acc	yr	car_type	origin_ame
0	1.498191	1.090604	0.673118	0.630870	-1.295498	-1.627426	-1.062235	0.773
1	1.498191	1.503514	1.589958	0.854333	-1.477038	-1.627426	-1.062235	0.773
2	1.498191	1.196232	1.197027	0.550470	-1.658577	-1.627426	-1.062235	0.773
3	1.498191	1.061796	1.197027	0.546923	-1.295498	-1.627426	-1.062235	0.773
4	1.498191	1.042591	0.935072	0.565841	-1.840117	-1.627426	-1.062235	0.773
...	...	...	...	...	...	...	...	
393	-0.856321	-0.513026	-0.479482	-0.213324	0.011586	1.621983	0.941412	0.773
394	-0.856321	-0.925936	-1.370127	-0.993671	3.279296	1.621983	0.941412	-1.292
395	-0.856321	-0.561039	-0.531873	-0.798585	-1.440730	1.621983	0.941412	0.773
396	-0.856321	-0.705077	-0.662850	-0.408411	1.100822	1.621983	0.941412	0.773
397	-0.856321	-0.714680	-0.584264	-0.296088	1.391285	1.621983	0.941412	0.773

398 rows × 10 columns

In [20]:

```
y_s
```

Out[20]:

	mpg
0	-0.706439
1	-1.090751
2	-0.706439
3	-0.962647
4	-0.834543
...	...
393	0.446497
394	2.624265
395	1.087017
396	0.574601
397	0.958913

398 rows × 1 columns

In [21]:

```
x_train,x_test,y_train,y_test=train_test_split(x_s,y_s,test_size=0.30,random_state=
```

In [22]:
```
x_train
```

Out[22]:

	cyl	disp	hp	wt	acc	yr	car_type	origin_ame
230	1.498191	1.503514	1.720935	1.412400	-1.513346	0.268063	-1.062235	0.773
357	-0.856321	-0.714680	-0.112746	-0.420234	-0.278877	1.351199	0.941412	-1.292
140	1.498191	1.061796	1.197027	1.521175	-0.024722	-0.544290	-1.062235	0.773
22	-0.856321	-0.858718	-0.243723	-0.703997	0.701436	-1.627426	0.941412	-1.292
250	1.498191	1.196232	0.935072	0.903991	-0.859804	0.538847	-1.062235	0.773
...	...	...	...	...	...	...	...	...
323	-0.856321	-0.359385	0.018232	-0.201501	-0.424109	1.080415	0.941412	0.773
192	0.320935	0.543257	0.018232	0.452336	-0.387801	-0.002721	-1.062235	0.773
117	-0.856321	-1.204411	-1.448713	-1.304628	1.427593	-0.815074	0.941412	-1.292
47	0.320935	0.543257	-0.112746	0.368389	-0.206262	-1.356642	-1.062235	0.773
172	-0.856321	-0.993154	-0.872414	-0.883713	0.338357	-0.273506	0.941412	-1.292

278 rows × 10 columns

In [23]:
```
y_test
```

Out[23]:

	mpg
65	-1.218855
132	0.190289
74	-1.346959
78	-0.322127
37	-0.706439
...	...
236	0.254341
352	0.817999
92	-1.346959
221	-0.770491
322	2.957335

120 rows × 1 columns

simple linear model

In [25]:
```
# Fit simple linear model and find coefficients

regressor=LinearRegression()
regressor.fit(x_train,y_train)

for idx, col_name in enumerate(x_train.columns):
    print('The coefficient for {} is {}'.format(col_name,regressor.coef_[0][idx]))

intercept=regressor.intercept_[0]
print('The intercept is {}'.format(intercept))
```

```
The coefficient for cyl is 0.2474447975894671
The coefficient for disp is 0.28838215446098686
The coefficient for hp is -0.18990342687152878
The coefficient for wt is -0.673222906511177
The coefficient for acc is 0.0675450154068818
The coefficient for yr is 0.34463640721172734
The coefficient for car_type is 0.31491491540037686
The coefficient for origin_america is -0.07682943694882902
The coefficient for origin_asia is 0.0633604889661997
The coefficient for origin_europe is 0.031283357351475284
The intercept is -0.019500467624017432
```

Regularized ridge regression

In [27]:
```
ridge_model=Ridge(alpha=0.3)
ridge_model.fit(x_train,y_train)

print("Ridge model coef: {}".format(ridge_model.coef_))
```

```
Ridge model coef: [ 0.24424435  0.27853222 -0.18980689 -0.66458446  0.06588077  0.34
396213
 0.31169746 -0.07642734  0.06333336  0.03080065]
```

Regularized Lasso Regression

In [28]:
```
lasso_model=Lasso(alpha=0.1)
lasso_model.fit(x_train,y_train)

print("LAsso model coef: {}".format(lasso_model.coef_))
```

```
LAsso model coef: [-0.         -0.         -0.06203044 -0.48363379  0.          0.27
163751
 0.09620861 -0.03490256  0.          0.        ]
```

In [30]:
```
# simple linear model
print(regressor.score(x_train,y_train))
print(regressor.score(x_test,y_test))

# Ridge
print(ridge_model.score(x_train,y_train))
print(ridge_model.score(x_test,y_test))

#Lasso
```

```
print(lasso_model.score(x_train,y_train))
print(lasso_model.score(x_test,y_test))
```

```
0.836163800114943
0.8439452810748138
0.8361520170844985
0.8437853815947187
0.7994535676270829
0.81026554865651
```

In [31]:

```
data_train_test=pd.concat([x_train,y_train],axis=1)
data_train_test.head()
```

Out[31]:

	cyl	disp	hp	wt	acc	yr	car_type	origin_ame
230	1.498191	1.503514	1.720935	1.412400	-1.513346	0.268063	-1.062235	0.773
357	-0.856321	-0.714680	-0.112746	-0.420234	-0.278877	1.351199	0.941412	-1.292
140	1.498191	1.061796	1.197027	1.521175	-0.024722	-0.544290	-1.062235	0.773
22	-0.856321	-0.858718	-0.243723	-0.703997	0.701436	-1.627426	0.941412	-1.292
250	1.498191	1.196232	0.935072	0.903991	-0.859804	0.538847	-1.062235	0.773