Regularization is a technique used to reduce the errors by fitting the function appropriately on the given training set and avoid overfitting.

L1 regularization

Minimization objective = LS Obj + α * (sum of absolute value of coefficients)

L2 regularization

βridge=(X′X+λI)−1(X′Y)

- I denotes the identity matrix.
- The λ parameter is the regularization penalty.

A regression model which uses L1 Regularization technique is called

LASSO(Least Absolute Shrinkage and Selection Operator) regression.

A regression model that uses L2 regularization technique is called Ridge regression.

Lasso Regression adds “absolute value of magnitude” of coefficient as penalty term to the loss function(L).

Advantages

1. Avoids overfitting a model.

1. They does not require unbiased estimators.

1. The ridge estimator is preferably good at improving the least-squares estimate when there is multicollinearity.

1. They add just enough bias to make the estimates reasonably reliable approximations to true population values.

Disadvantages

1.They include all the predictors in the final model.

2.They shrink the coefficients towards zero.

3.They trade the variance for bias.

In [30]:
```
import pandas as pd
import numpy  as np
import seaborn as sns
import matplotlib.pyplot as plt
import warnings
warnings.filterwarnings('ignore')
```

In [2]:
```
df=pd.read_csv('cars.csv')
```

In [3]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 205 entries, 0 to 204
Data columns (total 15 columns):
 #   Column             Non-Null Count  Dtype  
---  ------             --------------  -----  
 0   symboling          205 non-null    int64  
 1   normalized-losses  205 non-null    object 
 2   make               205 non-null    object 
 3   fuel-type          205 non-null    object 
 4   body-style         205 non-null    object 
 5   drive-wheels       205 non-null    object 
 6   engine-location    205 non-null    object 
 7   width              205 non-null    float64
 8   height             205 non-null    float64
 9   engine-type        205 non-null    object 
 10  engine-size        205 non-null    int64  
 11  horsepower         205 non-null    object 
 12  city-mpg           205 non-null    int64  
 13  highway-mpg        205 non-null    int64  
 14  price              205 non-null    int64  
dtypes: float64(2), int64(5), object(8)
memory usage: 24.1+ KB
```

In [4]:
```
df.head()
```

Out[4]:

	symboling	normalized-losses	make	fuel-type	body-style	drive-wheels	engine-location	width	height	engine-type	engine-size	horsepower	city-mpg	highway-mpg	price
0	3	?	alfa-romero	gas	convertible	rwd	front	64.1	48.8	dohc	130	111	21	27	13495
1	3	?	alfa-romero	gas	convertible	rwd	front	64.1	48.8	dohc	130	111	21	27	16500
2	1	?	alfa-romero	gas	hatchback	rwd	front	65.5	52.4	ohcv	152	154	19	26	16500
3	2	164	audi	gas	sedan	fwd	front	66.2	54.3	ohc	109	102	24	30	13950
4	2	164	audi	gas	sedan	4wd	front	66.4	54.3	ohc	136	115	18	22	17450

In [ ]:

In [5]:
```
df['normalized-losses'].replace('?',np.nan,inplace=True)
df['normalized-losses']=df['normalized-losses'].astype(float)
nlm=df['normalized-losses'].mean()
df['normalized-losses'].fillna(nlm,inplace=True)
```

In [6]:
```
df['horsepower'].replace('?',np.nan,inplace=True)
df['horsepower']=df['horsepower'].astype(float)
nlm=df['horsepower'].mean()
df['horsepower'].fillna(nlm,inplace=True)
```

In [7]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 205 entries, 0 to 204
Data columns (total 15 columns):
 #   Column             Non-Null Count  Dtype  
---  ------             --------------  -----  
 0   symboling          205 non-null    int64  
 1   normalized-losses  205 non-null    float64
 2   make               205 non-null    object 
 3   fuel-type          205 non-null    object 
 4   body-style         205 non-null    object 
 5   drive-wheels       205 non-null    object 
 6   engine-location    205 non-null    object 
 7   width              205 non-null    float64
 8   height             205 non-null    float64
 9   engine-type        205 non-null    object 
 10  engine-size        205 non-null    int64  
 11  horsepower         205 non-null    float64
 12  city-mpg           205 non-null    int64  
 13  highway-mpg        205 non-null    int64  
 14  price              205 non-null    int64  
dtypes: float64(4), int64(5), object(6)
memory usage: 24.1+ KB
```

In [8]:
```
df_cat=df.select_dtypes(object)
df_num=df.select_dtypes(['int64','float'])
```

In [9]:
```
from sklearn.preprocessing import LabelEncoder
```

In [10]:
```
for col in df_cat:
    lb=LabelEncoder()
    df_cat[col]=lb.fit_transform(df_cat[col])
```

In [15]:
```
df=pd.concat([df_cat,df_num],axis=1)
df.describe()
```

Out[15]:

	make	fuel-type	body-style	drive-wheels	engine-location	engine-type	symboling	normalized-losses	width	height	engine-size	horsepower	city-mpg	highway-mpg	price
count	205.000000	205.000000	205.000000	205.000000	205.000000	205.000000	205.000000	205.000000	205.000000	205.000000	205.000000	205.000000	205.000000	205.000000	205.000000
mean	12.195122	0.902439	2.614634	1.326829	0.014634	3.014634	0.834146	122.000000	65.907805	53.724878	126.907317	104.256158	25.219512	30.751220	13227.478049
std	6.274831	0.297446	0.859081	0.556171	0.120377	1.054765	1.245307	31.681008	2.145204	2.443522	41.642693	39.519211	6.542142	6.886443	7902.651615
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	-2.000000	65.000000	60.300000	47.800000	61.000000	48.000000	13.000000	16.000000	5118.000000
25%	8.000000	1.000000	2.000000	1.000000	0.000000	3.000000	0.000000	101.000000	64.100000	52.000000	97.000000	70.000000	19.000000	25.000000	7788.000000
50%	12.000000	1.000000	3.000000	1.000000	0.000000	3.000000	1.000000	122.000000	65.500000	54.100000	120.000000	95.000000	24.000000	30.000000	10345.000000
75%	19.000000	1.000000	3.000000	2.000000	0.000000	3.000000	2.000000	137.000000	66.900000	55.500000	141.000000	116.000000	30.000000	34.000000	16500.000000
max	21.000000	1.000000	4.000000	2.000000	1.000000	6.000000	3.000000	256.000000	72.300000	59.800000	326.000000	288.000000	49.000000	54.000000	45400.000000

In [ ]:

In [34]:
```
x=df.iloc[:,:-1]
y=df.iloc[:,-1]
```

In [35]:
```
from sklearn.model_selection import train_test_split as t
xtrain,xtest,ytrain,ytest=t(x,y,random_state=1)
```

In [36]:
```
from sklearn.linear_model import LinearRegression as linreg
from sklearn.metrics import r2_score  as r2,mean_squared_error as mse ,accuracy_score as ac
```

In [37]:
```
lr=linreg()
lr.fit(xtrain,ytrain)
ypred=lr.predict(xtest)
print(f' accuracy {r2(ytest,ypred)*100}')
print(f'  rmse     {np.sqrt(mse(ytest,ypred))*100}')
```

```
accuracy 81.74935371483143
 rmse     349685.6374302878
```

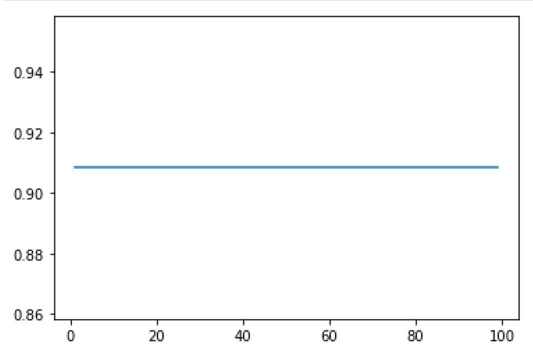
In [38]:
```
xtrain,xtest,ytrain,ytest=t(x,y,random_state=156)
lr=linreg()
lr.fit(xtrain,ytrain)
ypred=lr.predict(xtest)
print(f'  accuracy       {r2(ytest,ypred)*100}')
print(f'  rmse     {np.sqrt(mse(ytest,ypred))*100}')
```

```
 accuracy       90.83497619577983
 rmse     260284.19337298637
```

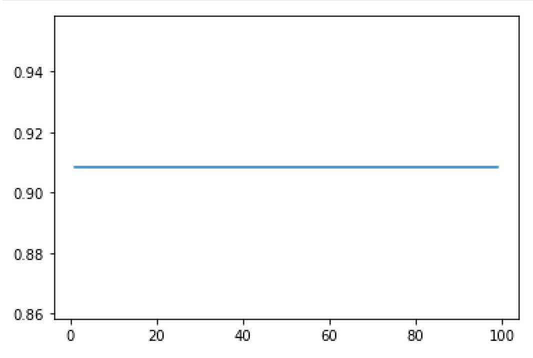
In [ ]:

In [39]:
```
from sklearn.linear_model import Lasso as l,Ridge as r
```

In [40]:
```
ac=[]
for i in range(1,100):
    l1=l(alpha=i,random_state=1)
    l1.fit(xtrain,ytrain)
    lypred=l1.predict(xtest)
    ac.append(r2(ytest,ypred))
plt.plot(range(1,100),ac)
plt.show()
```



In [41]:
```
ac=[]
for i in range(1,100):
    l1=l(alpha=1,random_state=i)
    l1.fit(xtrain,ytrain)
    lypred=l1.predict(xtest)
    ac.append(r2(ytest,ypred))
plt.plot(range(1,100),ac)
plt.show()
```

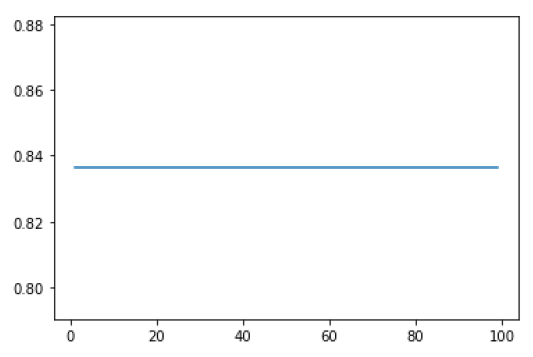


In [42]:
```
l1=l(alpha=1,random_state=i)
l1.fit(xtrain,ytrain)
lypred=l1.predict(xtest)
print(r2(ytest,lypred)*100)
```

```
90.83325932329973
```

In [43]:
```
ac=[]
for i in range(1,100):
    l2=r(alpha=10)
    l2.fit(xtrain,ytrain)
    l2ypred=l2.predict(xtest)
    ac.append(r2(ytest,l2ypred))
plt.plot(range(1,100),ac)
```

Out[43]:
```
[<matplotlib.lines.Line2D at 0x1ff0b867430>]
```



In [51]:
```
l2=r(alpha=10)
l2.fit(xtrain,ytrain)
l2ypred=l2.predict(xtest)
print(r2(ytest,l2ypred)*100)
```

```
83.63523911863774
```

From both l1 and l2 reguralization we can say l1 is giving more accuracy

that is 90.83325932329973

In [ ]:

In [ ]:

In [ ]: