In [2]:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
```

In [42]:

```
df = pd.read_csv('cars24-car-price-clean.csv')
df.head()
```

Out[42]:

	selling_price	year	km_driven	mileage	engine	max_power	make	model	transmissio
0	1.20	2012.0	120000	19.70	796.0	46.30	Maruti	Alto Std	
1	5.50	2016.0	20000	18.90	1197.0	82.00	Hyundai	Grand i10 Asta	
2	2.15	2010.0	60000	17.00	1197.0	80.00	Hyundai	i20 Asta	
3	2.26	2012.0	37000	20.92	998.0	67.10	Maruti	Alto K10 2010-2014 VXI	
4	5.70	2015.0	30000	22.77	1498.0	98.59	Ford	Ecosport 2015-2021 1.5 TDCi Titanium BSIV	

In [4]:

```
X = df['max_power'].values
Y = df['selling_price'].values
```

In [5]:

```
X.shape
```

Out[5]:

```
(19820,)
```

In [6]:

```
Y.shape
```

Out[6]:

```
(19820,)
```

In [7]:

```
X[:5]
```

Out[7]:

```
array([46.3 , 82.  , 80.  , 67.1 , 98.59])
```

In [8]:

```
Y[:5]
```

Out[8]:

```
array([1.2 , 5.5 , 2.15, 2.26, 5.7 ])
```

Sklearn / Scikit-learn

In [9]:

```
!pip install scikit-learn
```

```
Requirement already satisfied: scikit-learn in /Users/mohit/opt/anacon
da3/lib/python3.8/site-packages (0.24.1)
Requirement already satisfied: threadpoolctl>=2.0.0 in /Users/mohit/op
t/anaconda3/lib/python3.8/site-packages (from scikit-learn) (2.1.0)
Requirement already satisfied: numpy>=1.13.3 in /Users/mohit/opt/anaco
nda3/lib/python3.8/site-packages (from scikit-learn) (1.20.1)
Requirement already satisfied: joblib>=0.11 in /Users/mohit/opt/anacon
da3/lib/python3.8/site-packages (from scikit-learn) (1.0.1)
Requirement already satisfied: scipy>=0.19.1 in /Users/mohit/opt/anaco
nda3/lib/python3.8/site-packages (from scikit-learn) (1.6.2)
```

In [10]:

```
from sklearn.linear_model import LinearRegression
```

In [11]:

```
# create the model
model = LinearRegression()
type(model)
```

Out[11]:

```
sklearn.linear_model._base.LinearRegression
```

In [12]:

```
# train the model
model.fit(X, Y)
```

```
---------------------------------------------------------------------
-----
ValueError                                Traceback (most recent call
 last)
<ipython-input-12-2241c6f73b39> in <module>
      1 # train the model
----> 2 model.fit(X, Y)

~/opt/anaconda3/lib/python3.8/site-packages/sklearn/linear_model/_bas
e.py in fit(self, X, y, sample_weight)
    516         accept_sparse = False if self.positive else ['csr', 'c
sc', 'coo']
    517
--> 518         X, y = self._validate_data(X, y, accept_sparse=accept_
sparse,
    519                                    y_numeric=True, multi_outpu
t=True)
    520

~/opt/anaconda3/lib/python3.8/site-packages/sklearn/base.py in _valida
te_data(self, X, y, reset, validate_separately, **check_params)
    431                 y = check_array(y, **check_y_params)
    432             else:
--> 433                 X, y = check_X_y(X, y, **check_params)
    434             out = X, y
    435

~/opt/anaconda3/lib/python3.8/site-packages/sklearn/utils/validation.p
y in inner_f(*args, **kwargs)
     61             extra_args = len(args) - len(all_args)
     62             if extra_args <= 0:
---> 63                 return f(*args, **kwargs)
     64
     65             # extra_args > 0

~/opt/anaconda3/lib/python3.8/site-packages/sklearn/utils/validation.p
y in check_X_y(X, y, accept_sparse, accept_large_sparse, dtype, order,
copy, force_all_finite, ensure_2d, allow_nd, multi_output, ensure_min_
samples, ensure_min_features, y_numeric, estimator)
    812         raise ValueError("y cannot be None")
    813
--> 814     X = check_array(X, accept_sparse=accept_sparse,
    815                     accept_large_sparse=accept_large_sparse,
    816                     dtype=dtype, order=order, copy=copy,

~/opt/anaconda3/lib/python3.8/site-packages/sklearn/utils/validation.p
y in inner_f(*args, **kwargs)
     61             extra_args = len(args) - len(all_args)
     62             if extra_args <= 0:
---> 63                 return f(*args, **kwargs)
     64
     65             # extra_args > 0

~/opt/anaconda3/lib/python3.8/site-packages/sklearn/utils/validation.p
y in check_array(array, accept_sparse, accept_large_sparse, dtype, ord
```

```
er, copy, force_all_finite, ensure_2d, allow_nd, ensure_min_samples, e
nsure_min_features, estimator)
    635             # If input is 1D raise error
    636             if array.ndim == 1:
--> 637                 raise ValueError(
    638                     "Expected 2D array, got 1D array instea
d:\narray={}.\n"
    639                     "Reshape your data either using array.resh
ape(-1, 1) if "

ValueError: Expected 2D array, got 1D array instead:
array=[ 46.3   82.    80.   ... 103.52 140.   117.6 ].
Reshape your data either using array.reshape(-1, 1) if your data has a
single feature or array.reshape(1, -1) if it contains a single sample.
```

In [15]:

```
X = X.reshape(-1, 1)
Y = Y.reshape(-1, 1)
```

In [17]:

```
# train the model
model = model.fit(X, Y)
```

In [32]:

```
x_query = np.array([120, 85])
model.predict(x_query.reshape(-1, 1))
```

Out[32]:

```
array([[10.64248787],
       [ 5.43648191]])
```

In [25]:

```
# w1
print(model.coef_)
```

```
[[0.14874303]]
```

In [26]:

```
# w0
print(model.intercept_)
```

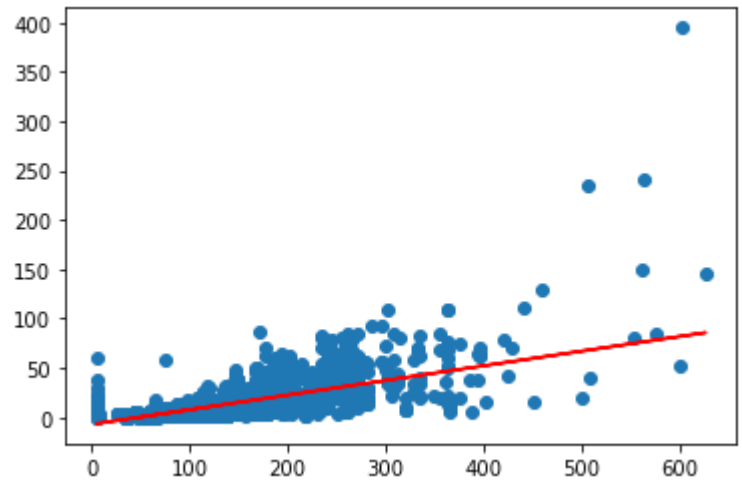
```
[-7.20667543]
```

In [28]:

```
Y_hat = model.predict(X)
```

In [29]:

```
plt.scatter(X[:, 0], Y[:, 0])
plt.plot(X, Y_hat, c='red')
plt.show()
```



In [30]:

```
model.score(X, Y)
```

Out[30]:

```
0.5402545880839582
```

In [ ]:

```

```

Multiple Linear Regression

In [33]:

```
def predict(X, weights):
    return np.dot(X, weights)
```

In [76]:

```
def error(X, Y, weights):

    Y_hat = predict(X, weights)
    err = np.mean((Y - Y_hat)**2)

    return err
```

In [77]:

```
def gradient(X, Y, weights):

    n = X.shape[0]

    Y_hat = predict(X, weights)
    grad = np.dot(X.T, Y - Y_hat)

    grad = (grad*-2)/n

    return grad
```

In [78]:

```
def gradient_descent(X, Y, epochs=800, learning_rate =0.1):

    weights = np.zeros((X.shape[1],1))
    error_list = []


    for i in range(epochs):
        # Compute grad
        grad = gradient(X,Y,weights)

        e = error(X,Y,weights)
        error_list.append(e)

        #Update weights
        weights = weights - learning_rate*grad



    return weights, error_list
```

In [43]:

```
df.drop(columns=['make', 'model'], inplace=True)
```

In [44]:

```
df.head()
```

Out[44]:

	selling_price	year	km_driven	mileage	engine	max_power	transmission_type	seats_coup
0	1.20	2012.0	120000	19.70	796.0	46.30	1	
1	5.50	2016.0	20000	18.90	1197.0	82.00	1	
2	2.15	2010.0	60000	17.00	1197.0	80.00	1	
3	2.26	2012.0	37000	20.92	998.0	67.10	1	
4	5.70	2015.0	30000	22.77	1498.0	98.59	1	

In [47]:

```
Y = df['selling_price'].values
```

In [50]:

```
X = df.drop(columns=['selling_price']).values
```

In [52]:

```
X.shape
```

Out[52]:

```
(19820, 17)
```

In [53]:

```
Y.shape
```

Out[53]:

```
(19820,)
```

In [55]:

```
Y = Y.reshape(-1, 1)
```

In [56]:

```
Y.shape
```

Out[56]:

```
(19820, 1)
```

Standardisation

In [62]:

```
mu = X.mean(axis=0)
sig = X.std(axis=0)
```

In [64]:

```
X_new = (X-mu)/sig
```

In [65]:

```
X_new.shape
```

Out[65]:

```
(19820, 17)
```

In [69]:

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

In [71]:

```
X_new = np.hstack((ones, X_new))
```

In [72]:

```
X_new.shape
```

Out[72]:

```
(19820, 18)
```

call gradient descent

In [79]:

```
weights, error_list = gradient_descent(X_new, Y, epochs=100)
```

In [80]:

```
weights
```

Out[80]:

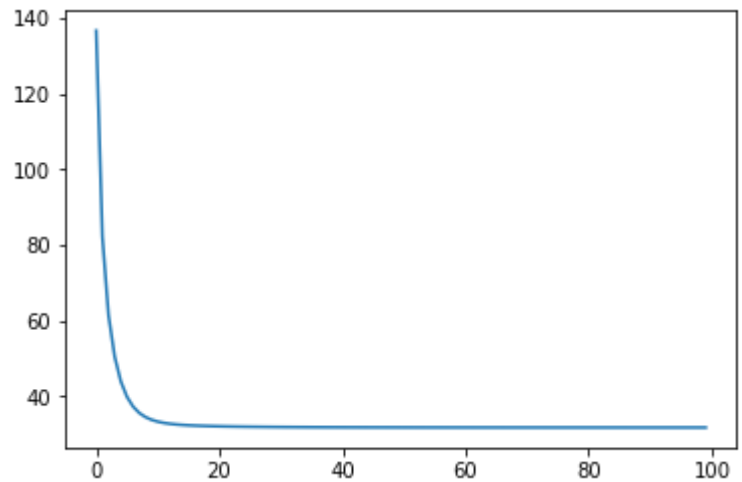
```
array([[ 7.38842289],
       [ 1.88406033],
       [-0.47746363],
       [ 0.36740462],
       [ 1.26013856],
       [ 4.9954916 ],
       [-0.98036804],
       [ 0.82812398],
       [-0.27796105],
       [-0.22819305],
       [ 0.06523244],
       [ 0.0632498 ],
       [-0.06181939],
       [ 0.20760246],
       [-0.1006866 ],
       [ 0.03415015],
       [-0.01677573],
       [-0.08771605]])
```

In [81]:

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

Out[81]:

```
[<matplotlib.lines.Line2D at 0x7f8320bf6b50>]
```



In [82]:

```
def r2_score(X, Y, weights):
    Y_hat = predict(X, weights)
    numerator = np.sum((Y-Y_hat)**2)
    denominator = np.sum((Y - Y.mean())**2)

    return (1- numerator/denominator)
```

In [84]:

```
Y_hat = predict(X_new, weights)
```

In [88]:

```
r2_score(X_new, Y, weights)
```

Out[88]:

```
0.6137962361083591
```

Multiple linear regression using sklearn?

In [ ]:

```

```

Adjusted R_squared

In [90]:

```
X_new.shape
```

Out[90]:

```
(19820, 18)
```

In [94]:

```
adj_R2 = 1 - (1-r2_score(X_new, Y, weights))*(len(Y)-1)/(len(Y)-X_new.shape[1]-2)
print("Adjusted R-squared:", adj_R2 )
```

```
Adjusted R-squared: 0.6134256365369479
```

In [ ]:

```

```

Suppose that we have N independent variables (X1,X2… Xn) and dependent variable is Y. Now Imagine that you are applying linear regression by fitting the best fit line using least square error on this data.

You found that correlation coefficient for one of it's variable(Say X1) with Y is -0.95.

Which of the following is true for X1?

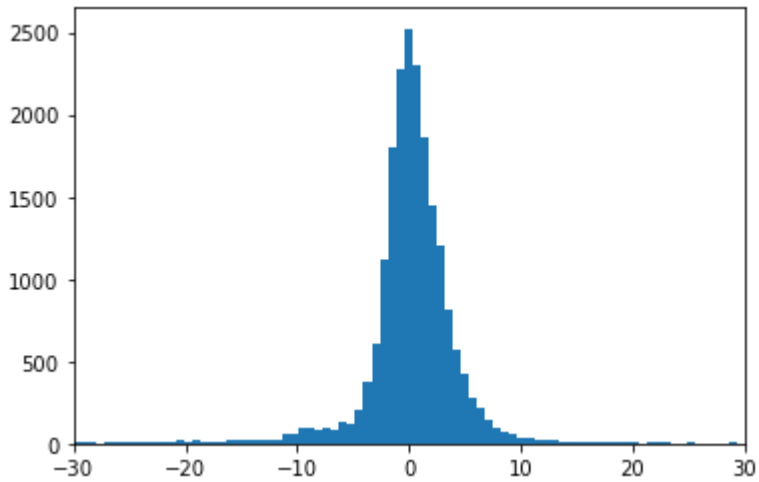
1. Relation between the X1 and Y is weak
2. Relation between the X1 and Y is strong
3. Relation between the X1 and Y is neutral
4. Correlation can't judge the relationship

In [95]:

```
errs = Y_hat - Y
```

In [97]:

```
plt.hist(errs, bins=500)
plt.xlim(-30, 30)
plt.show()
```



In [ ]:

```

```