Linear_Reg

In [1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

import warnings
warnings.filterwarnings("ignore")
```

In [2]:
```
#simple_linera_regression

class Simple_linear_regression:

    def __init__(self,learning_rate=1e-3,n_steps=1000):
        self.learning_rate =learning_rate
        self.n_steps =n_steps

    def fit(self,X,y):

        # adding the bias term
        X_train = np.c_[np.ones(X.shape[0]),X]

        # random initialization of the model weights
        self.W =np.random.rand((X_train.shape[1]))

        # random initialization of the model weights
        for i in range(self.n_steps):
            self.W =self.W -self.learning_rate*self.cal_gradiant_descent(X_train,y)

    def cal_gradiant_descent(self,X,y):

        #calculating gradiant descent

        return 2/X.shape[0] * np.dot(X.T,np.dot(X,self.W)-y)

    def predict(self,X):

        #Predicting Y for the X

        #adding bias term

        X_pred =np.c_[np.ones(X.shape[0]),X]

        return np.dot(X_pred,self.W)
```

In [3]:
```
#creating dataset
from sklearn.datasets import make_regression
X , y = make_regression (n_samples=1000,n_features = 1,n_targets=1,bias =2.5,noise=40,random_state = 44)
print("X_shape =",X.shape)
print("y_shape =",y.shape)
```

```
X_shape = (1000, 1)
y_shape = (1000,)
```

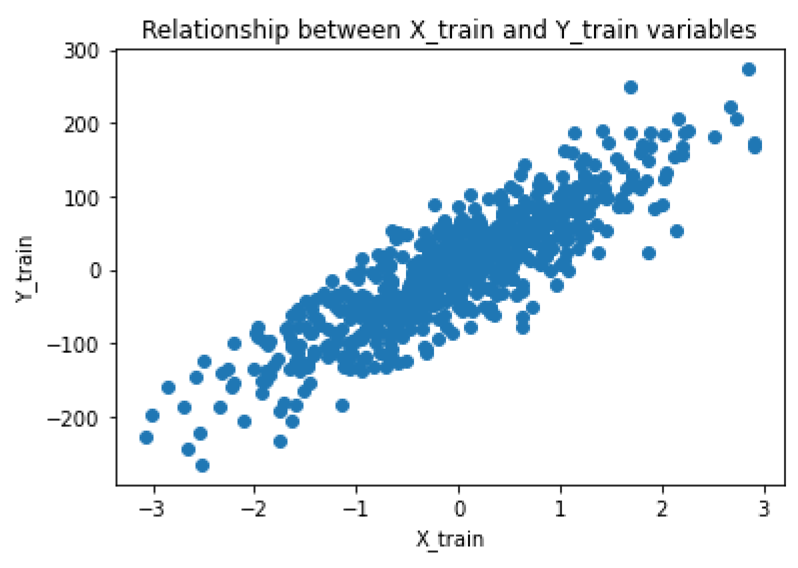
In [4]:
```
#train_test_split
from sklearn.model_selection import train_test_split
X_train,X_test,y_train,y_test =train_test_split(X,y,test_size=.33,random_state=12)
```

In [5]:
```
print("Shape X_train :",X_train.shape)
print("Shape y_train :",y_train.shape)
print("Shape X_test :",X_test.shape)
print("Shape y_test :",y_test.shape)
```

```
Shape X_train : (670, 1)
Shape y_train : (670,)
Shape X_test : (330, 1)
Shape y_test : (330,)
```

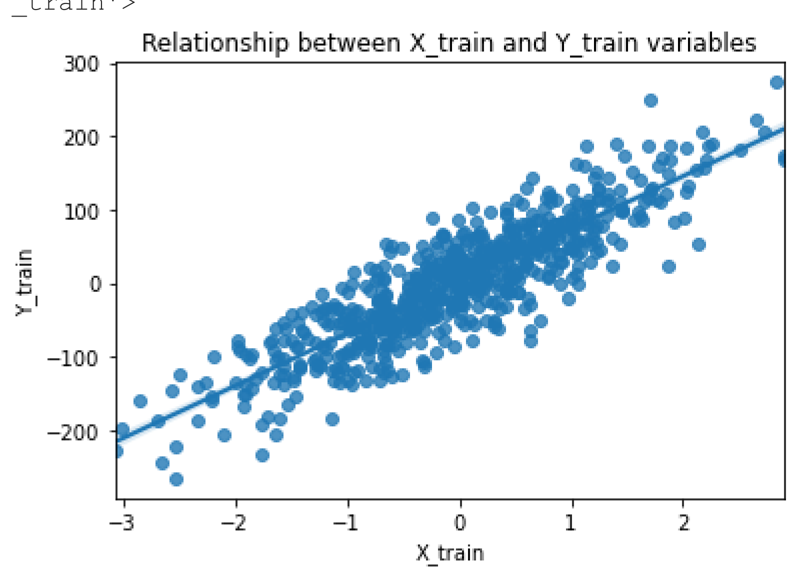
In [6]:
```
%matplotlib inline
import matplotlib.pyplot as plt

plt.xlabel('X_train')
plt.ylabel('Y_train')
plt.title('Relationship between X_train and Y_train variables')
plt.scatter(X_train, y_train)
```

Out[6]:
```
<matplotlib.collections.PathCollection at 0x25c7efed0a0>
```



In [7]:
```
plt.xlabel('X_train')
plt.ylabel('Y_train')
plt.title('Relationship between X_train and Y_train variables')
sns.regplot(X_train, y_train)
```

Out[7]:
```
<AxesSubplot:title={'center':'Relationship between X_train and Y_train variables'}, xlabel='X_train', ylabel='Y
_train'>
```



In [8]:
```
#model
model = Simple_linear_regression()
model.fit(X_train,y_train)
```

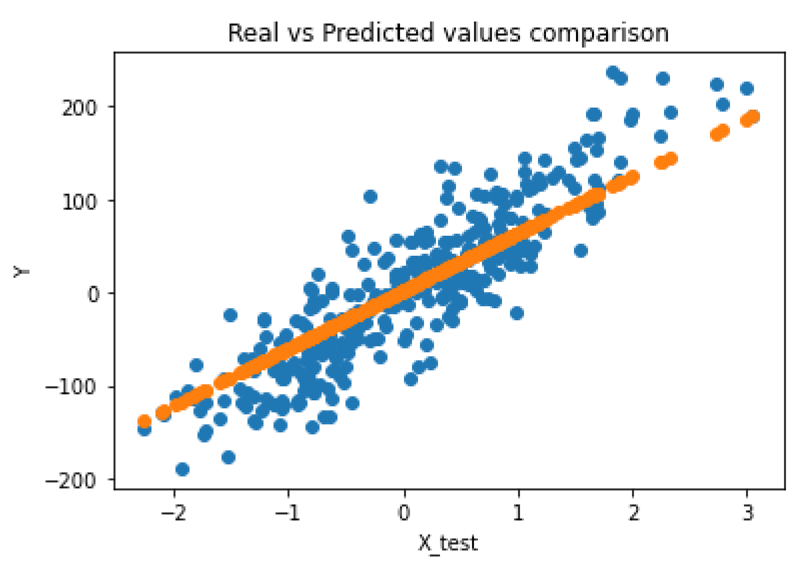
In [9]:
```
#prediction
y_pred =model.predict(X_test)
```

In [10]:
```
#error
print("Mean squared error: %.2f" % np.mean((model.predict(X_test) - y_test) ** 2))
```

```
Mean squared error: 1716.39
```

In [11]:
```
plt.xlabel('X_test')
plt.ylabel('Y')
plt.title('Real vs Predicted values comparison')

plt.scatter(X_test, y_test)
plt.scatter(X_test, y_pred)
```

Out[11]:
```
<matplotlib.collections.PathCollection at 0x25c7f113b20>
```



In [12]:
```
#Same with Sklearn lib
from sklearn.linear_model import LinearRegression
modelSk =LinearRegression()
modelSk.fit(X_train,y_train)
```

Out[12]:
```
LinearRegression()
```

In [13]:
```
y_predict=modelSk.predict(X_test)
```

In [14]:
```
#error
print("Mean squared error: %.2f" % np.mean((modelSk.predict(X_test) - y_test) ** 2))
```

```
Mean squared error: 1534.17
```

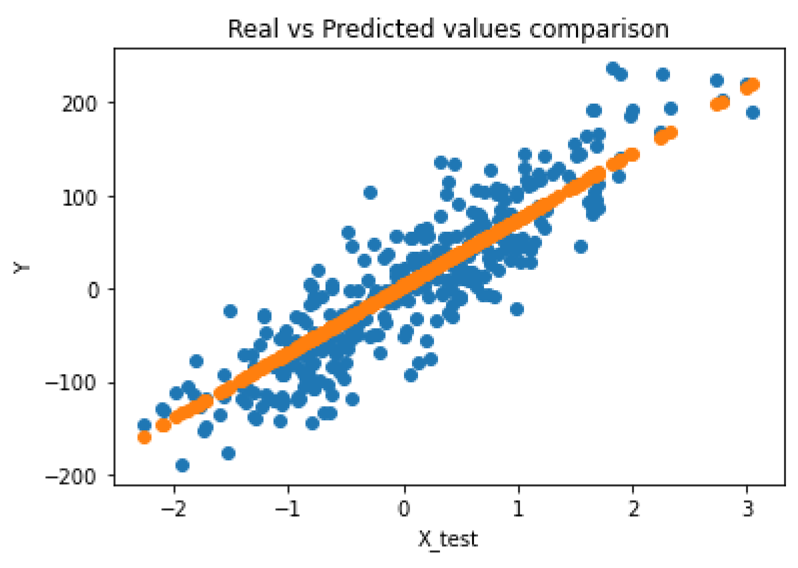
In [15]:
```
def accuracy(X_test,y_test, y_pred):
    print('accuracy (R^2):\n', modelSk.score(X_test, y_test)*100, '%')
```

In [16]:
```
accuracy(X_test,y_test,y_predict)
```

```
accuracy (R^2):
 79.03370956723134 %
```

In [17]:
```
plt.xlabel('X_test')
plt.ylabel('Y')
plt.title('Real vs Predicted values comparison')

plt.scatter(X_test, y_test)
plt.scatter(X_test, y_predict)
```

Out[17]:
```
<matplotlib.collections.PathCollection at 0x25c1006d550>
```



In [ ]:

In [ ]:

In [ ]: