Multi-Variate Linear Regression

68_Adnan Shaikh

In [9]:
```
import numpy as np
import pandas as pd
from matplotlib import pyplot as plt
from sklearn.model_selection import train_test_split
```

In [10]:
```
#Lambda function to create linearly dependent feature using linearly independent features
dep_feat = lambda X: np.sum([np.random.uniform(10,10)+np.random.randint(0,2)*np.random.randint(-10
,10)*np.random.rand()*x for x in X],axis=0)
```

In [11]:
```
#Initializing constants N <- No. of records, NO_INDEP <- No. of Independent Features, NO_DEP <- N
o. of Dependent Features
N = 1000
NO_INDEP = 10
NO_DEP = 4
```

In [12]:

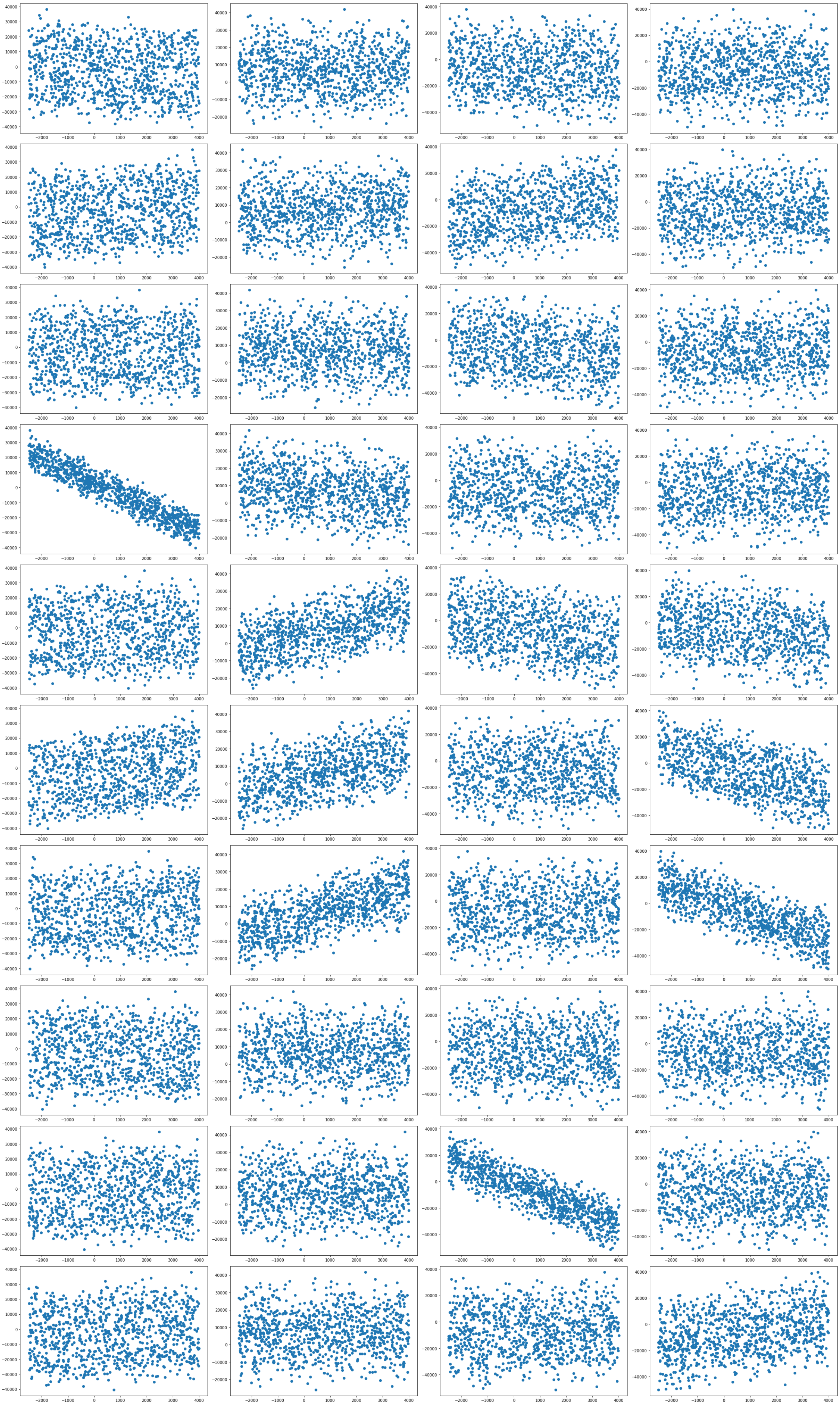
```
#Initializing Independent Features X and Dependent Features Y
X = np.array([np.random.uniform(low=-2500,high=4001,size=N) for _ in range(NO_INDEP)])
Y = np.transpose([dep_feat(X) for _ in range(NO_DEP)])
X = np.transpose(X)
print(f"Shape of Independent Array: {X.shape},Shape of Dependent Array: {Y.shape}")
print(f"Sample of X data from each features:\n {X[:10,:]}")
print(f"Sample of Y data from each features:\n {Y[:10,:]}")
```

```
Shape of Independent Array: (1000, 10),Shape of Dependent Array: (1000, 4)
Sample of X data from each features:
 [[ 3517.36411745  3242.55487782  1504.67607976  3226.63227622
   -920.1981244  -1707.33163588  3706.33546492  2932.22514714
   3569.83653402  -365.01638012]
 [ 2970.65588953  -398.40570039  2685.21420735  -755.71512461
   1410.56656655  2561.16992899  3298.43661622  3678.38940818
   -825.84615784  3853.50737534]
 [ 2264.11644431  1723.10709214  2575.55049952 -1545.96619944
  -1310.78850035  3785.76949599  3492.19283486 -1782.50668535
    724.47400094  1112.58514722]
 [ 1826.44220667 -2028.40216093  1547.78294526 -1125.21689097
   -518.83851474  -545.89148162 -1966.03802717  1491.70816542
   3274.73803414    62.60049293]
 [  684.06862355  -790.12721043  2459.11760112  3316.94509863
   3648.94615902  1570.81760048   103.78725663 -1978.66985567
   -564.87640845 -1221.29854422]
 [  875.10627978  2961.91361651  1300.35312827  -434.08739244
  -1097.70175763  -291.25786239 -2119.14918139  1786.16295187
   -353.02453889  1235.46064586]
 [  125.29622962  2506.00256549  1463.22622442 -1898.50438102
    196.46293069  3240.44440033   957.4538656  -1301.082368
    977.84967921   867.99298017]
 [-1170.80852259  -402.22742058  1805.49815175  2384.89282736
   -322.50930537  -640.78597159 -1037.2555477  -2273.88239981
   2052.57617831  -990.12846267]
 [-2309.07704444  1409.13580929  1517.52149903  -438.0430452
  -1118.0855794   2396.95264855  -961.24801975   -45.24891748
  -1948.04437598  1837.55807605]
 [ 2295.18705355 -1644.35723336  3257.59362967 -1348.72718286
  -2079.70400979  -372.83631433  -279.54406207  1099.88192433
  -2333.76306898  -678.09158362]]
Sample of Y data from each features:
 [[-30006.32284031   1797.11074228 -21365.7146995  -10003.18125475]
 [ 13145.29345795  27641.28417799  -3758.05432889 -25619.70098495]
 [ 23497.50496002  22732.26919346  -1048.03173066 -33701.56351273]
 [  4205.92218009  -9402.66895759 -33746.2089082   14332.16639925]
 [-25192.2736726   13613.87713678 -15578.31542618 -17673.47704061]
 [  6457.05888182 -12372.42860218  13302.0848313   24417.43081554]
 [ 27305.11625887  17149.78818647  -2069.22452284 -17498.41658928]
 [-21270.96301299 -10930.79251929 -22242.86025724   8213.42668203]
 [ 14689.71168783    128.67904649  20720.707765     3595.33946789]
 [  5949.04474399  -7617.30647249  13666.99309792   3514.0923618 ]]
```

In [13]:

```
#Scatter plot between each dependent and independent variable
fig,axes = plt.subplots(NO_INDEP,NO_DEP,figsize=(30,50),tight_layout=True)
for i in range(NO_INDEP):
  for j in range(NO_DEP):
    axes[i,j].scatter(x=X[:,i],y=Y[:,j])

plt.show()
```


40000
30000
20000
10000
0
−10000
−20000
−30000
−40000
−2000
−1000
0
1000
2000
3000
4000

In [14]:
```
class MultiVariateRegression:

  def fit_model(self,X,Y):
    # Estimator Beta[i,j] calculation using matrices (Generalization of Multiple Linear Regression
to Multivariate Regression)
    self.X,self.Y =  np.concatenate((np.ones((len(X),1)),X),axis=1),np.array(Y)
    self.X_transpose = self.X.T
    self.compose_mat = np.matmul(self.X_transpose,self.X) #Making square matrix that contains all
 the required summation of (XiXj)
    self.compose_inverse = np.linalg.inv(self.compose_mat)
    self.Beta = np.matmul(np.matmul(self.compose_inverse,self.X_transpose),Y) #Calculating Beta Es
timators

  def predict(self,X):
    return np.matmul(np.concatenate(([1],X)),self.Beta) #Predicting Single Sample


  def predict_many(self,indep_feat):
    #Predicting Multiple Sample
    Y = []
    for x in indep_feat:
      Y.append(self.predict(x))
    return np.array(Y)

  def residual(self,Y,predicted_Y):
    #Error calculation using SSR and averaging errors of all the dependent variables
    sq_of_res = np.square(np.subtract(Y,predicted_Y))
    np.round(sq_of_res,2)
    ssr = np.sum(np.transpose(sq_of_res),axis=1)
    return np.round(np.average(ssr),2)
```

In [15]:
```
#Model Demonstration
mvr = MultiVariateRegression()
X_train, X_test, Y_train,Y_test = train_test_split(X,Y,test_size=0.33,random_state=69,shuffle=True
) # Splitting in training and test set
mvr.fit_model(X_train,Y_train)
Y_predicted = mvr.predict_many(X_test)
print(f"Residual: {mvr.residual(Y_test,Y_predicted)}")
#Residual is zero since each dependent vector is a linear combination of independent vectors (refe
r block 2) it shows model working as required
#It can also be used to check relation between independent variables
```

```
Residual: 0.0
```