In [83]:
```python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

# 1. LINEAR REGRESSION ON TIME SERIES DATA

In [84]:
```python
df = pd.read_csv('time_series.csv')
df.head()
```

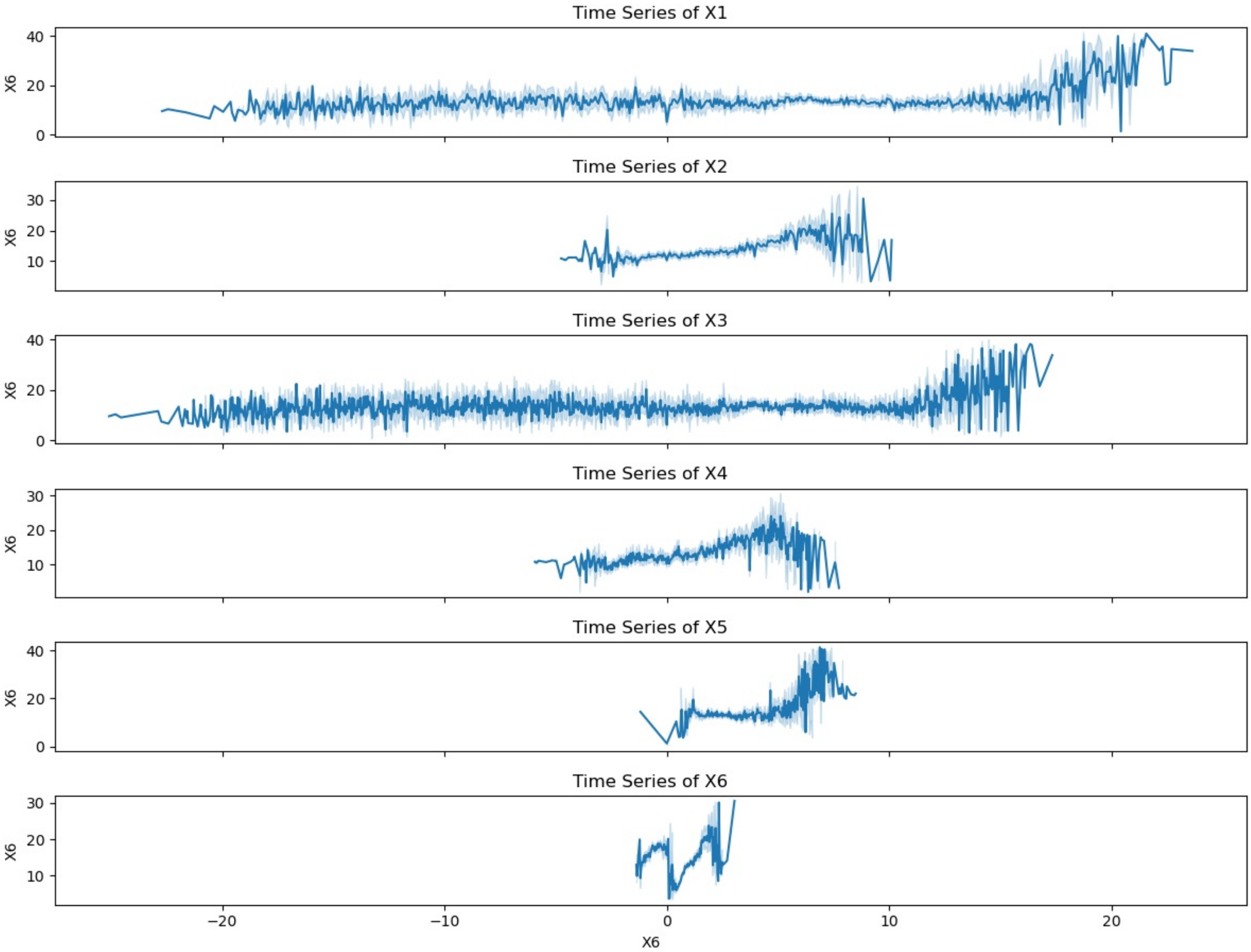
Out[84]:

| | date | X1 | X2 | X3 | X4 | X5 | X6 | Y |
|---|---|---|---|---|---|---|---|---|
| **0** | 2016-07-01 00:00:00 | 5.827 | 2.009 | 1.599 | 0.462 | 4.203 | 1.340 | 30.531000 |
| **1** | 2016-07-01 01:00:00 | 5.693 | 2.076 | 1.492 | 0.426 | 4.142 | 1.371 | 27.787001 |
| **2** | 2016-07-01 02:00:00 | 5.157 | 1.741 | 1.279 | 0.355 | 3.777 | 1.218 | 27.787001 |
| **3** | 2016-07-01 03:00:00 | 5.090 | 1.942 | 1.279 | 0.391 | 3.807 | 1.279 | 25.044001 |
| **4** | 2016-07-01 04:00:00 | 5.358 | 1.942 | 1.492 | 0.462 | 3.868 | 1.279 | 21.948000 |

In [85]:
```python
sample_df = df.head(300)
```

In [86]:
```python
fig, axes = plt.subplots(6, 1, figsize=(12, 10), sharex=True)

# Loop to plot each variable
lst = ['X1', 'X2', 'X3', 'X4', 'X5','X6']
for i in range(0,6):
    sns.lineplot(x=lst[i], y='Y', data=df, ax=axes[i])
    axes[i].set_title(f'Time Series of {lst[i]}')
    axes[i].set_ylabel(var)

plt.tight_layout(rect=[0, 0.03, 1, 0.95])
```

```
C:\Users\tegbe\anaconda3\Lib\site-packages\seaborn\_oldcore.py:1119: FutureWarning: use_inf_as_na option is depr
ecated and will be removed in a future version. Convert inf values to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
C:\Users\tegbe\anaconda3\Lib\site-packages\seaborn\_oldcore.py:1119: FutureWarning: use_inf_as_na option is depr
ecated and will be removed in a future version. Convert inf values to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
C:\Users\tegbe\anaconda3\Lib\site-packages\seaborn\_oldcore.py:1119: FutureWarning: use_inf_as_na option is depr
ecated and will be removed in a future version. Convert inf values to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
C:\Users\tegbe\anaconda3\Lib\site-packages\seaborn\_oldcore.py:1119: FutureWarning: use_inf_as_na option is depr
ecated and will be removed in a future version. Convert inf values to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
C:\Users\tegbe\anaconda3\Lib\site-packages\seaborn\_oldcore.py:1119: FutureWarning: use_inf_as_na option is depr
ecated and will be removed in a future version. Convert inf values to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
C:\Users\tegbe\anaconda3\Lib\site-packages\seaborn\_oldcore.py:1119: FutureWarning: use_inf_as_na option is depr
ecated and will be removed in a future version. Convert inf values to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
C:\Users\tegbe\anaconda3\Lib\site-packages\seaborn\_oldcore.py:1119: FutureWarning: use_inf_as_na option is depr
ecated and will be removed in a future version. Convert inf values to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
C:\Users\tegbe\anaconda3\Lib\site-packages\seaborn\_oldcore.py:1119: FutureWarning: use_inf_as_na option is depr
ecated and will be removed in a future version. Convert inf values to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
C:\Users\tegbe\anaconda3\Lib\site-packages\seaborn\_oldcore.py:1119: FutureWarning: use_inf_as_na option is depr
ecated and will be removed in a future version. Convert inf values to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
C:\Users\tegbe\anaconda3\Lib\site-packages\seaborn\_oldcore.py:1119: FutureWarning: use_inf_as_na option is depr
ecated and will be removed in a future version. Convert inf values to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
C:\Users\tegbe\anaconda3\Lib\site-packages\seaborn\_oldcore.py:1119: FutureWarning: use_inf_as_na option is depr
ecated and will be removed in a future version. Convert inf values to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
C:\Users\tegbe\anaconda3\Lib\site-packages\seaborn\_oldcore.py:1119: FutureWarning: use_inf_as_na option is depr
ecated and will be removed in a future version. Convert inf values to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
```




Exploring the Correlation

In [87]:
```
corr_df = df.drop('date',axis=1)
corr_df.corr()
```

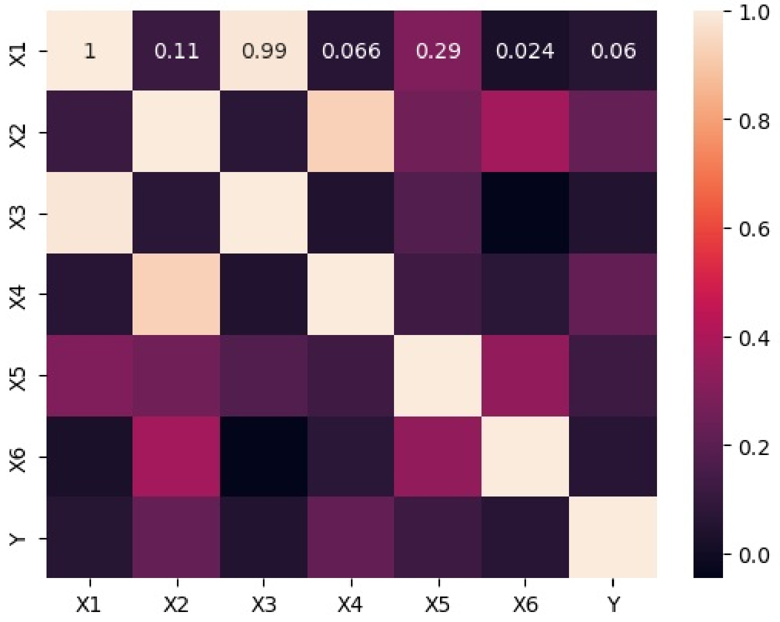
Out[87]:

| | X1 | X2 | X3 | X4 | X5 | X6 | Y |
|---|---|---|---|---|---|---|---|
| **X1** | 1.000000 | 0.114672 | 0.987355 | 0.066002 | 0.291418 | 0.023606 | 0.059916 |
| **X2** | 0.114672 | 1.000000 | 0.068817 | 0.930491 | 0.259487 | 0.377641 | 0.224354 |
| **X3** | 0.987355 | 0.068817 | 1.000000 | 0.046266 | 0.177491 | -0.046519 | 0.050854 |
| **X4** | 0.066002 | 0.930491 | 0.046266 | 1.000000 | 0.128607 | 0.069419 | 0.220004 |
| **X5** | 0.291418 | 0.259487 | 0.177491 | 0.128607 | 1.000000 | 0.334563 | 0.118836 |
| **X6** | 0.023606 | 0.377641 | -0.046519 | 0.069419 | 0.334563 | 1.000000 | 0.067455 |
| **Y** | 0.059916 | 0.224354 | 0.050854 | 0.220004 | 0.118836 | 0.067455 | 1.000000 |

Columns X1 and X3 look to be very closely correlated

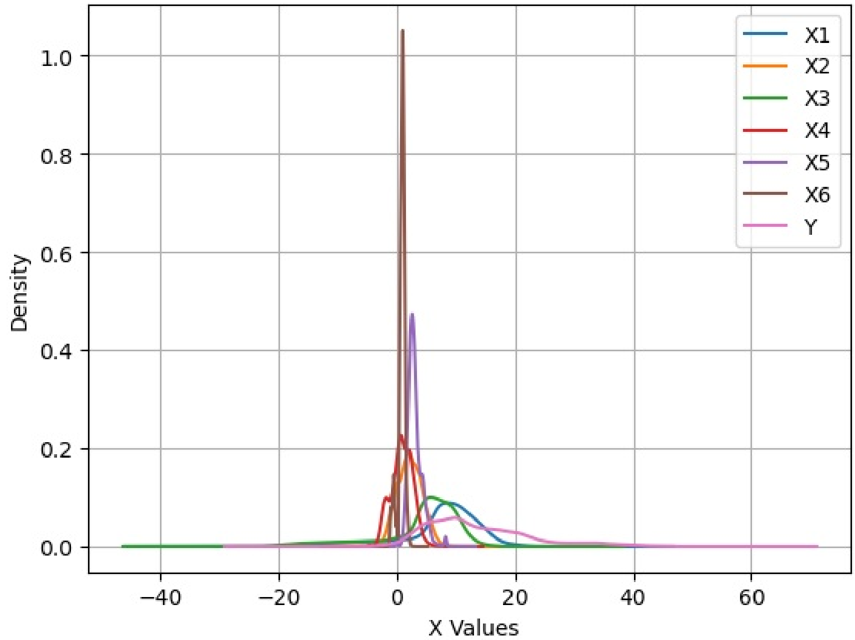
In [88]:
```
sns.heatmap(corr_df.corr(),annot=True)
```

Out[88]: <Axes: >




Plotting the Kernel Density

In [89]:
```
df.plot(kind='kde')
plt.xlabel(" X Values")
plt.ylabel("Density")
plt.grid(True)
plt.show()
```




In [ ]:

b.) Train/Test Split

In [90]:
```
df
```

Out[90]:

| | date | X1 | X2 | X3 | X4 | X5 | X6 | Y |
|---|---|---|---|---|---|---|---|---|
| **0** | 2016-07-01 00:00:00 | 5.827 | 2.009 | 1.599 | 0.462 | 4.203 | 1.340 | 30.531000 |
| **1** | 2016-07-01 01:00:00 | 5.693 | 2.076 | 1.492 | 0.426 | 4.142 | 1.371 | 27.787001 |
| **2** | 2016-07-01 02:00:00 | 5.157 | 1.741 | 1.279 | 0.355 | 3.777 | 1.218 | 27.787001 |
| **3** | 2016-07-01 03:00:00 | 5.090 | 1.942 | 1.279 | 0.391 | 3.807 | 1.279 | 25.044001 |
| **4** | 2016-07-01 04:00:00 | 5.358 | 1.942 | 1.492 | 0.462 | 3.868 | 1.279 | 21.948000 |
| **...** | ... | ... | ... | ... | ... | ... | ... | ... |
| **17415** | 2018-06-26 15:00:00 | -1.674 | 3.550 | -5.615 | 2.132 | 3.472 | 1.523 | 10.904000 |
| **17416** | 2018-06-26 16:00:00 | -5.492 | 4.287 | -9.132 | 2.274 | 3.533 | 1.675 | 11.044000 |
| **17417** | 2018-06-26 17:00:00 | 2.813 | 3.818 | -0.817 | 2.097 | 3.716 | 1.523 | 10.271000 |
| **17418** | 2018-06-26 18:00:00 | 9.243 | 3.818 | 5.472 | 2.097 | 3.655 | 1.432 | 9.778000 |
| **17419** | 2018-06-26 19:00:00 | 10.114 | 3.550 | 6.183 | 1.564 | 3.716 | 1.462 | 9.567000 |

17420 rows × 8 columns

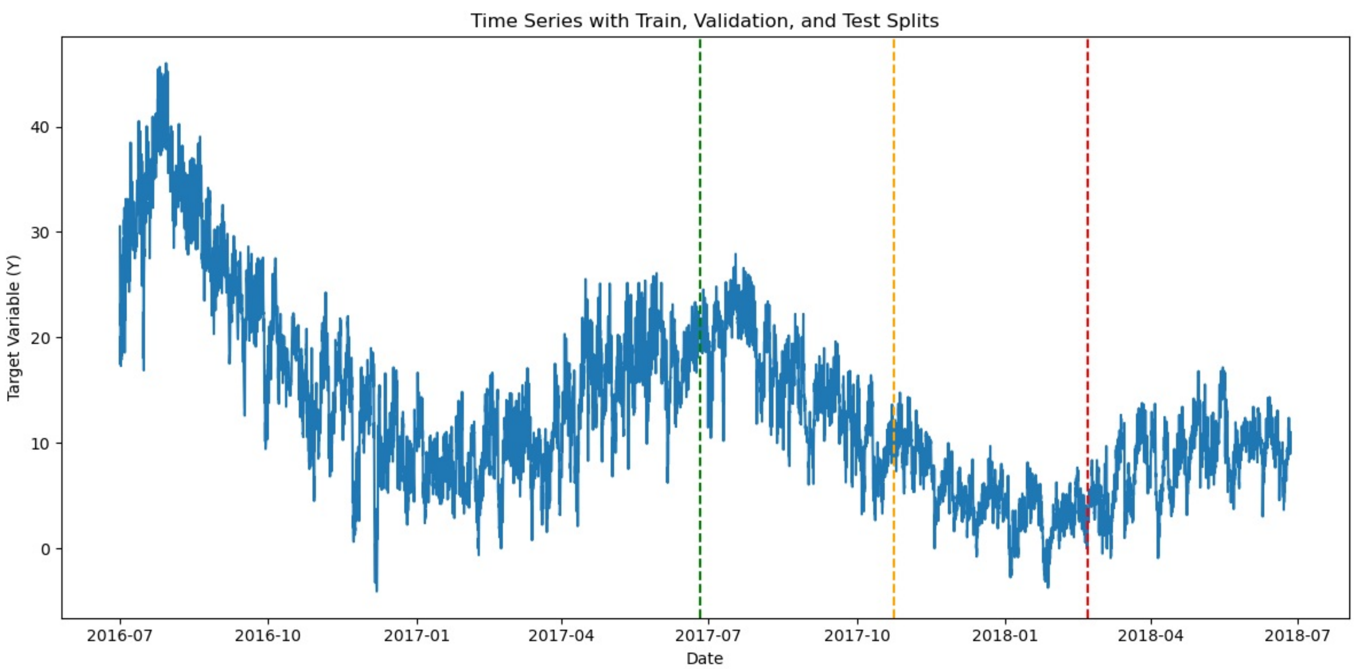
In [91]:

```
train = df.loc[df['date'] <= '2017-06-26 23:00:00']
val = df.loc[(df['date'] > '2017-06-26 23:00:00') & (df['date'] <= '2017-10-24 23:00')]
test = df.loc[(df['date'] > '2017-10-24 23:00') & (df['date'] <= '2018-02-21 23:00')]

print("Training set size:", len(train))
print("Validation set size:", len(val))
print("Test set size:", len(test))
```

```
Training set size: 8664
Validation set size: 2879
Test set size: 2880
```

In [92]:

```
df['date'] = pd.to_datetime(df['date'])

train_end_date = '2017-06-26'
val_end_date = '2017-10-24'
test_end_date = '2018-02-21'

plt.figure(figsize=(12, 6))
plt.plot(df['date'], df['Y'], label='Target Variable (Y)')

plt.axvline(pd.to_datetime(train_end_date), color='green', linestyle='--', label='Train End')
plt.axvline(pd.to_datetime(val_end_date), color='orange', linestyle='--', label='Validation End')
plt.axvline(pd.to_datetime(test_end_date), color='red', linestyle='--', label='Test End')

plt.title("Time Series with Train, Validation, and Test Splits")
plt.xlabel("Date")
plt.ylabel("Target Variable (Y)")

plt.tight_layout()
plt.show()
```




In [ ]:

In [ ]:

## c. Scaling

In [93]:
```
def scale_variables(data, metrics):
    scaled_data = data.copy()

    for variable in metrics:
        mean = train[variable].mean()
        std = train[variable].std()
        scaled_data[variable] = (data[variable] - mean) / std

    return scaled_data

variables = ['X1','X2','X3','X4','X5','X6']

scaled_train = scale_variables(train,variables)
scaled_val = scale_variables(val,variables)
scaled_test = scale_variables(test,variables)
```

In [ ]:

In [ ]:

## d. LSE

In [131…
```
X = scaled_train[['X1', 'X2', 'X3', 'X4', 'X5']]
Y = scaled_train['Y']

X = np.array(X)
Y1 = np.array(Y)

X_b = np.c_[np.ones(X.shape[0]), X] ### This will add an intercept column
```

In [133…
```
beta_hat = np.linalg.inv(X_b.T.dot(X_b)).dot(X_b.T).dot(Y1)
print("The estimated Parameters for Train are :", beta_hat)

Y_pred1 = X_b.dot(beta_hat)
print("Predictions for Train are :", Y_pred1)

# Calculate Mean Squared Error
mse = np.mean((Y1 - Y_pred1) ** 2)
mae = np.mean(np.abs(Y1 - Y_pred1))
print("Mean Squared Error and Mean Absolute Error for Train are :", mse,mae)
```

```
The estimated Parameters for Train are : [17.13662789 -1.91639749  5.26124611  1.78251227 -0.13756652  2.0145516
9]
Predictions for Train are : [19.49669908 19.55754778 18.1072748  ... 14.8454925  16.28669993
 17.33686072]
Mean Squared Error and Mean Absolute Error for Train are : 50.894201122160645 5.476539015669759
```

In [ ]:

## Now applying to Validation and Test Dataset

In [134…
```
X = scaled_val[['X1', 'X2', 'X3', 'X4', 'X5']]
Y = scaled_val['Y']

X = np.array(X)
Y2 = np.array(Y)
X_b = np.c_[np.ones(X.shape[0]), X]

beta_hat = np.linalg.inv(X_b.T.dot(X_b)).dot(X_b.T).dot(Y2)
print("The estimated Parameters for Validation are :", beta_hat)

Y_pred2 = X_b.dot(beta_hat)
print("Predictions for Validation:", Y_pred2)

# Calculate Mean Squared Error
mse = np.mean((Y2 - Y_pred2) ** 2)
mae = np.mean(np.abs(Y - Y_pred2))
print("Mean Squared Error and Mean Absolute Error for Validation:", mse,mae)
```

```
The estimated Parameters for Validation are : [13.81194261  7.68653594 -3.10614367 -7.71847754  2.83733242  1.98
827336]
Predictions for Validation: [14.97764    14.20710219 14.49630545 ... 15.54576055 12.2865472
 13.02299611]
Mean Squared Error and Mean Absolute Error for Validation: 11.195550354114475 2.7126178489513086
```

In [135…
```
X = scaled_test[['X1', 'X2', 'X3', 'X4', 'X5']]
```

```
Y = scaled_test['Y']

X = np.array(X)
Y3 = np.array(Y)
X_b = np.c_[np.ones(X.shape[0]), X]

beta_hat = np.linalg.inv(X_b.T.dot(X_b)).dot(X_b.T).dot(Y3)
print("The estimated Parameters for Test are :", beta_hat)

Y_pred3 = X_b.dot(beta_hat)
print("Predictions for Test:", Y_pred3)

# Calculate Mean Squared Error
mse = np.mean((Y3 - Y_pred3) ** 2)
mae = np.mean(np.abs(Y - Y_pred3))
print("Mean Squared Error and Mean Absolute Error for Test:", mse,mae)
```

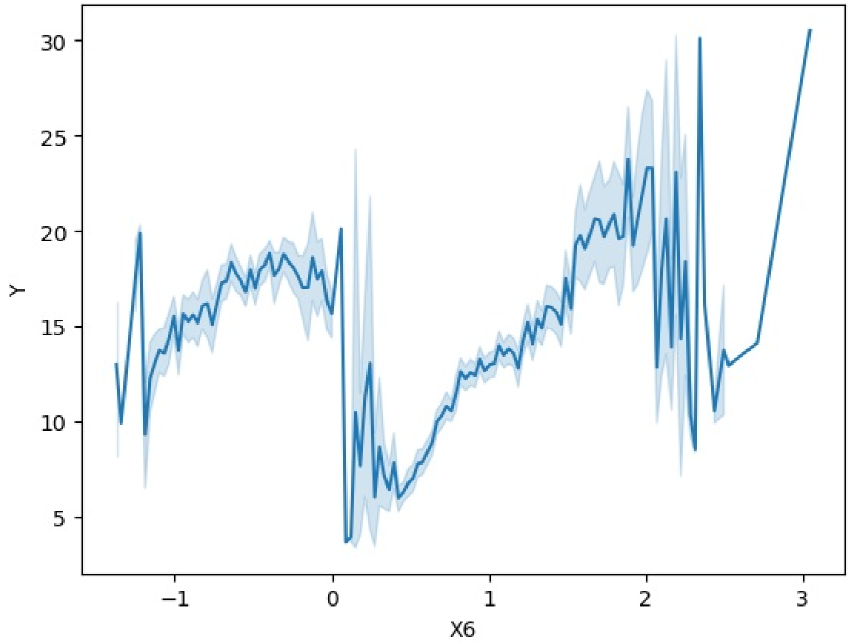
```
The estimated Parameters for Test are : [ 5.3615342  -4.56222556  3.66057755  3.88489226 -4.12782807 -0.67826609
 ]
Predictions for Test: [5.86041621 4.66510387 6.64783407 ... 1.88372612 1.9110887  2.60269018]
Mean Squared Error and Mean Absolute Error for Test: 7.285752203254583 2.1457949999478316
```

In [142…

```
sns.lineplot(x=lst[i], y='Y', data=df)
#plt.scatter(X, exponential_func(X, *popt), label='Fitted Curve', color='red')
```

```
C:\Users\tegbe\anaconda3\Lib\site-packages\seaborn\_oldcore.py:1119: FutureWarning: use_inf_as_na option is depr
ecated and will be removed in a future version. Convert inf values to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
C:\Users\tegbe\anaconda3\Lib\site-packages\seaborn\_oldcore.py:1119: FutureWarning: use_inf_as_na option is depr
ecated and will be removed in a future version. Convert inf values to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
```

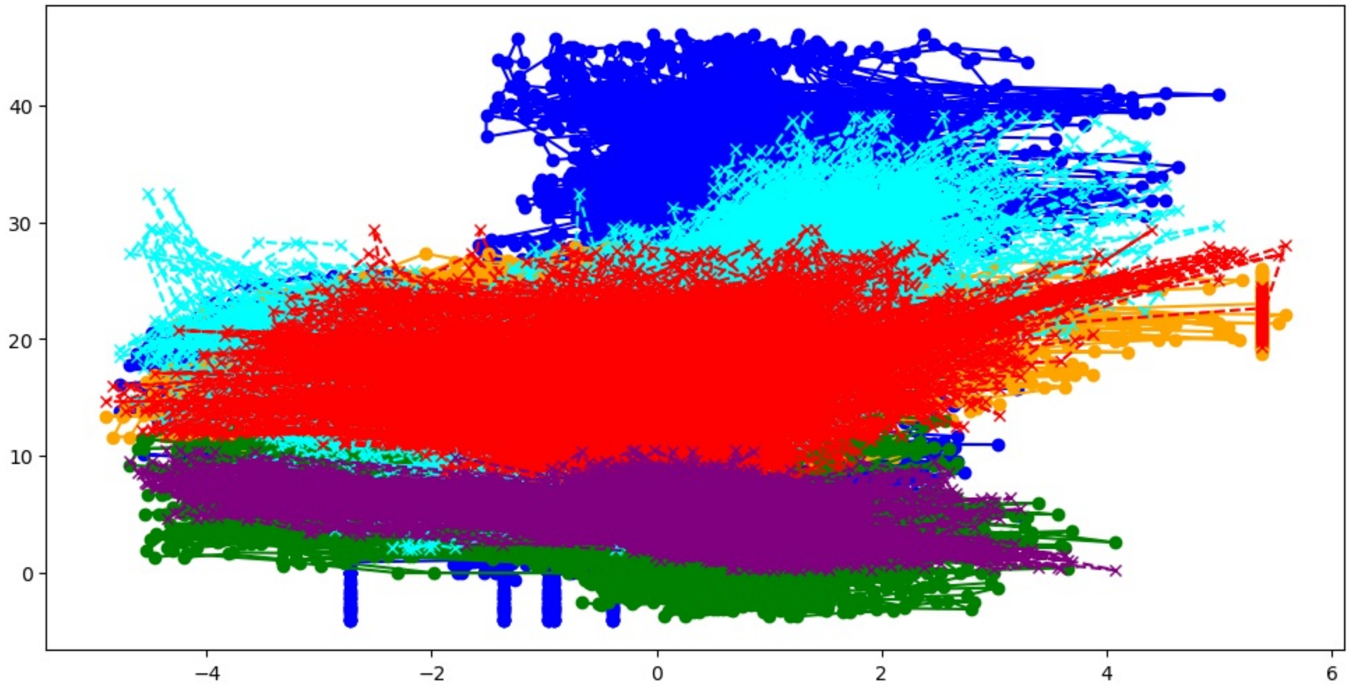
Out[142… <Axes: xlabel='X6', ylabel='Y'>




In [140…

```
X_train = scaled_train[['X1', 'X2', 'X3', 'X4', 'X5']]
X_val = scaled_val[['X1', 'X2', 'X3', 'X4', 'X5']]
X_test = scaled_test[['X1', 'X2', 'X3', 'X4', 'X5']]


Y_train = scaled_train['Y']
Y_val = scaled_val['Y']
Y_test = scaled_test['Y']

Y_pred1
Y_pred2
Y_pred2

plt.figure(figsize=(12, 6))

# Plot true values
plt.plot(X_train, Y_train, marker='o', label='Train True', color='blue')
plt.plot(X_val, Y_val, marker='o', label='Validation True', color='orange')
plt.plot(X_test, Y_test, marker='o', label='Test True', color='green')

# Plot predicted values
plt.plot(X_train, Y_pred1, marker='x', linestyle='--', label='Train Predicted', color='cyan')
plt.plot(X_val, Y_pred2, marker='x', linestyle='--', label='Validation Predicted', color='red')
plt.plot(X_test, Y_pred3, marker='x', linestyle='--', label='Test Predicted', color='purple')
```

Out[140… [<matplotlib.lines.Line2D at 0x13b78fc7b90>,
 <matplotlib.lines.Line2D at 0x13b78fc7bd0>,
 <matplotlib.lines.Line2D at 0x13b755fb8d0>,
 <matplotlib.lines.Line2D at 0x13b78fd0690>,
 <matplotlib.lines.Line2D at 0x13b78fd0ad0>]




In [ ]:

In [ ]:

## 2. VISUALIZING LSE ON A TOY DATASET

In [143…
```
df1 = pd.read_csv('toy_data.csv')
df1.head()
```

Out[143…

| | X1 | X2 | Y |
|---|---|---|---|
| **0** | 5.530492 | 8.136530 | 53.470131 |
| **1** | 5.111720 | 0.846906 | 15.925409 |
| **2** | 9.011047 | 6.510469 | 54.649639 |
| **3** | 7.806497 | 0.349096 | 24.003095 |
| **4** | 2.047190 | 1.057417 | 14.739897 |

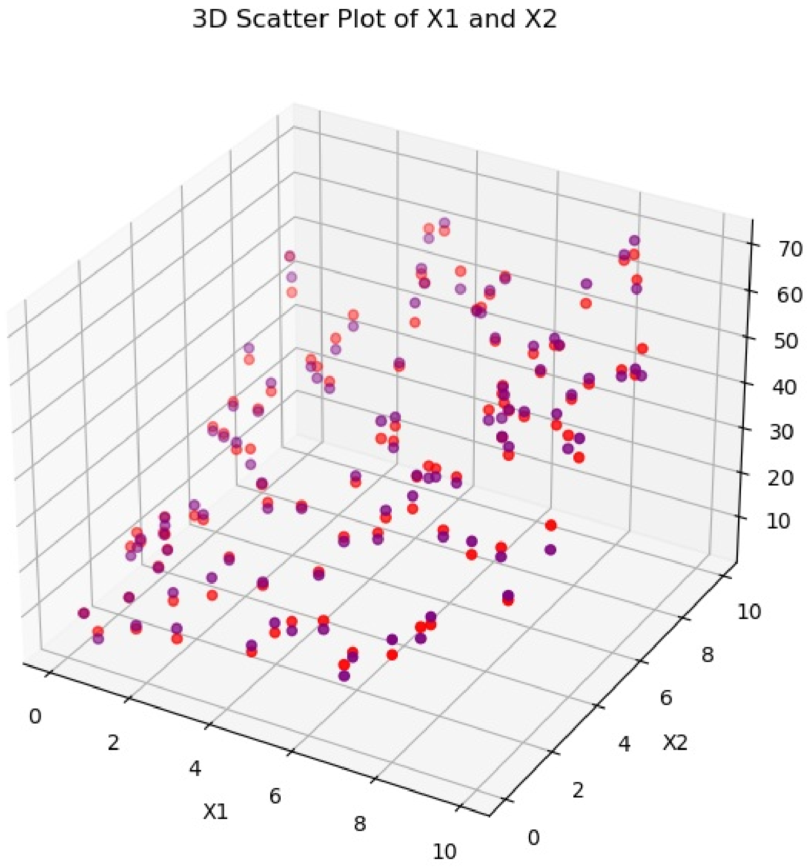
In [149…
```
X = df1[['X1', 'X2']]
Y = df1['Y']

#Converting it to Arrays
X = np.array(X)
Y = np.array(Y)

X_b = np.c_[np.ones(X.shape[0]), X] ### This will add an intercept column

### Now calculating the parameters using LSE

beta_hat = np.linalg.inv(X_b.T.dot(X_b)).dot(X_b.T).dot(Y)
print("The estimated Parameters for Test are :", beta_hat)
```

```
The estimated Parameters for Test are : [3.66345247 1.96836519 4.93212246]
```

In [157…
```
Y_pred = X_b.dot(beta_hat)
```

In [169…
```
from mpl_toolkits.mplot3d import Axes3D

fig = plt.figure(figsize=(10, 7))
ax = fig.add_subplot(111, projection='3d')

# Scatter plot
ax.scatter(df1['X1'], df1['X2'], df1['Y'], color='purple')
ax.scatter(df1['X1'], df1['X2'],Y_pred , color='red')

ax.set_xlabel('X1')
```

```python
ax.set_ylabel('X2')
ax.set_zlabel('Y')
ax.set_title('3D Scatter Plot of X1 and X2')

plt.show()
```




In [ ]:

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js