Daniel Lupercio HW3

8. We will now perform cross-validation on a simulated data set.

(a) Generate a simulated data set as follows

In [6]:
```python
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import numpy as np
import statsmodels.api as sm
```

In [7]:
```python
np.random.seed(1)
x = np.random.normal(loc=0, scale=1, size=100)
y = x - 2*(x**2) + np.random.normal(loc=0, scale=1, size=100)
```

In this data set, what is n and what is p? Write out the model used to generate the data in equation form.

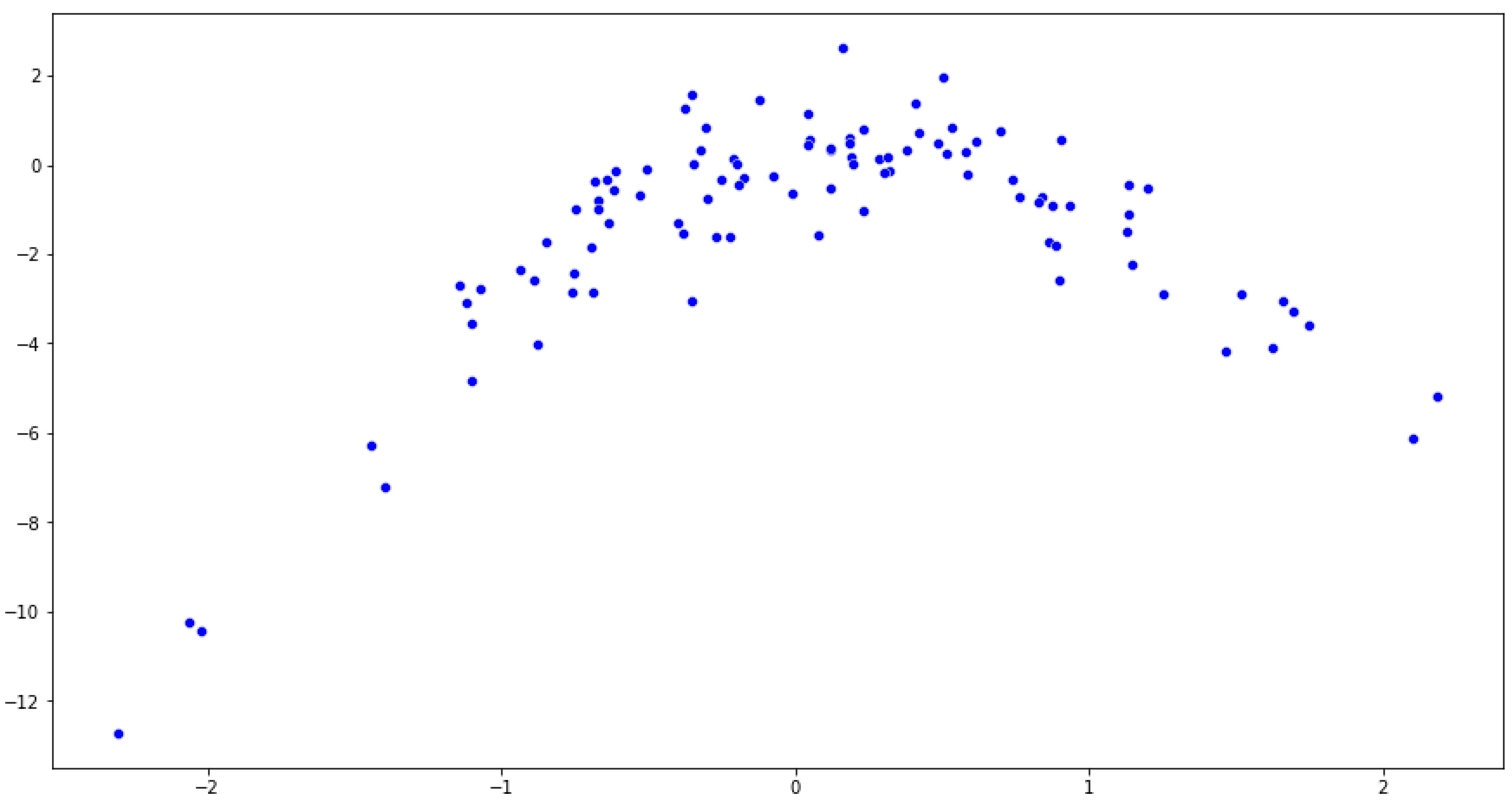
n is the number of observations, where n = 100. p is the number of parameters or variables used, here we are using two parameters.

$$Y = X - 2X^2 + \epsilon$$

(b) Create a scatterplot of X against Y. Comment on what you find.

In [8]:
```python
fig = plt.figure(figsize=(15, 8))
sns.scatterplot(x=x, y=y, color='b')
```

Out[8]:
```
<AxesSubplot:>
```



We see a negative quadratic function. With most of the data points in the domain of (-1.5, 2.5). This function has a range of (-12,3).

(c) Set a random seed, and then compute the LOOCV errors that result from fitting the following four models using least squares:

$$i. Y = \beta_0 + \beta_1 X + \epsilon$$
$$ii. Y = \beta_0 + \beta_1 X + \beta_2 X^2 + \epsilon$$
$$iii. Y = \beta_0 + \beta_1 X + \beta_2 X^2 + \beta_3 X^3 + \epsilon$$
$$iv. Y = \beta_0 + \beta_1 X + \beta_2 X^2 + \beta_3 X^3 + \beta_4 X^4 + \epsilon$$

Note you may find it helpful to use the $data.frame()$ function to create a single data set containing both X and Y.

In [13]:
```python
import random
from sklearn.linear_model import LinearRegression

def LOOCV(df): #df is defined/redefined for each model
    n = len(df)
    error = 0.0

    for i in range(n):
        test = df.iloc[[i]]
        train = df.drop(df.index[i])

        X_ = train.loc[:, train.columns != 'y'] #each dataframe will have a set number of x columns, and the last y column
        y_ = train['y']

        model = LinearRegression(fit_intercept=True)
        model.fit(X_, y_)

        X_ = test.loc[:, df.columns != 'y']
        predictions = model.predict(X_)
        error += (predictions - test.iloc[0]['y'])**2

    return (error/n)

random.seed(1)

# Model 1
df = pd.DataFrame({'x':x, 'y':y})
print("MSE for model 1: " +str(LOOCV(df)))

# Model 1
df = pd.DataFrame({'x':x, 'x2':x**2, 'y':y})
print("MSE for model 2: " +str(LOOCV(df)))

# Model 3
df = pd.DataFrame({'x':x, 'x2':x**2, 'x3':x**3, 'y':y})
print("MSE for model 3: " +str(LOOCV(df)))

# Model 4
df = pd.DataFrame({'x':x, 'x2':x**2, 'x3':x**3, 'x4':x**4, 'y':y})
print("MSE for model 4: " +str(LOOCV(df)))
```

```
MSE for model 1: [6.26076433]
MSE for model 2: [0.91428971]
MSE for model 3: [0.92687688]
MSE for model 4: [0.86691169]
```

(d) Repeat c) using another random seed, and report your results. Are your results the same as in part c)? Why?

In [15]:
```python
random.seed(5)
# Model 1
df = pd.DataFrame({'x':x, 'y':y})
print("MSE for model 1: " +str(LOOCV(df)))

# Model 1
df = pd.DataFrame({'x':x, 'x2':x**2, 'y':y})
print("MSE for model 2: " +str(LOOCV(df)))

# Model 3
df = pd.DataFrame({'x':x, 'x2':x**2, 'x3':x**3, 'y':y})
print("MSE for model 3: " +str(LOOCV(df)))

# Model 4
df = pd.DataFrame({'x':x, 'x2':x**2, 'x3':x**3, 'x4':x**4, 'y':y})
print("MSE for model 4: " +str(LOOCV(df)))
```

```
MSE for model 1: [6.26076433]
MSE for model 2: [0.91428971]
MSE for model 3: [0.92687688]
MSE for model 4: [0.86691169]
```

Yes, the results are the same as in part c). It apppears that the different random seed had no effect on the results.

(e) Which of the models in c) had the smallest LOOCV error? Is this what you expected? Explain your answer?

Model 4 has the lowest LOOCV error. I did not expect this, the relationship between x and y is a fourth degree polynomial. Interpreting this model is complex as is.

(f) Comment on the statistical significance of the coefficient estimates that results from fitting each of the models in c) using least squares. Do these results agree with the conclusions drawn based on the cross-validation results?

In [16]:
```python
#begin by using the model that has all the variables
df = pd.DataFrame({'x':x, 'x2':x**2, 'x3':x**3, 'x4':x**4, 'y':y})

X_ = df.loc[:, df.columns != 'y']
X_ = sm.add_constant(X_, prepend=True)
y_ = df['y']

model = sm.OLS(y_, X_)
result = model.fit()
print(result.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                      y   R-squared:                       0.873
Model:                            OLS   Adj. R-squared:                  0.867
Method:                 Least Squares   F-statistic:                     163.0
Date:                Sun, 24 Oct 2021   Prob (F-statistic):           1.24e-41
Time:                        15:25:13   Log-Likelihood:                -130.63
No. Observations:                 100   AIC:                             271.3
Df Residuals:                      95   BIC:                             284.3
Df Model:                           4
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const          0.3140      0.136      2.311      0.023       0.044       0.584
x              0.9127      0.183      4.999      0.000       0.550       1.275
x2            -2.5445      0.248    -10.264      0.000      -3.037      -2.052
x3             0.0992      0.064      1.556      0.123      -0.027       0.226
x4             0.1394      0.057      2.437      0.017       0.026       0.253
==============================================================================
Omnibus:                        1.537   Durbin-Watson:                   2.100
Prob(Omnibus):                  0.464   Jarque-Bera (JB):                1.088
Skew:                          -0.238   Prob(JB):                        0.581
Kurtosis:                       3.184   Cond. No.                         15.9
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
/Users/daniel421/Desktop/STAT_724/ds_724/lib/python3.8/site-packages/statsmodels/tsa/tsatools.py:142: FutureWarning: In a future version of pandas all arguments of conc
at except for the argument 'objs' will be keyword-only
  x = pd.concat(x[::order], 1)
```

Using the fourth model, we are able to see that the cubic term, is statistically insignificant.