# Module 4 Homework

ISE-529 Predictive Analytics

In [10]:
```
import numpy as np
import pandas as pd
import seaborn as sns
import statsmodels.stats.outliers_influence as smo
from sklearn.linear_model import LinearRegression
from sklearn import metrics
import statsmodels.api as sm
import statsmodels.formula.api as smf
```

## Linear Model Diagnosis

1) For this problem, you are to load the file "Problem 1 Dataset.csv" into a dataframe and perform model diagnosis on it to improve it. Use the steps identified in the slide in Module 4 at the end of the Model Diagnosis section (titled "Initial Steps for Model Diagnosis and Improvement"). Add comments to each step in your analysis describing your results and decisions and, at the end, write out the final equation of your model along with its $R^2$

In [11]:
```
prob1_dataset = pd.read_csv('Problem 1 Dataset.csv')
```

Looking at correlation matrix, we see no pairwise correlations greater than 0.9

In [12]:
```
prob1_dataset.drop('Y',1).corr()
```

Out[12]:

| | X1 | X2 | X3 | X4 | X5 |
|---|---|---|---|---|---|
| **X1** | 1.000000 | 0.057226 | 0.040811 | 0.567937 | 0.024574 |
| **X2** | 0.057226 | 1.000000 | -0.000972 | 0.025559 | 0.016519 |
| **X3** | 0.040811 | -0.000972 | 1.000000 | 0.823847 | 0.002501 |
| **X4** | 0.567937 | 0.025559 | 0.823847 | 1.000000 | 0.017179 |
| **X5** | 0.024574 | 0.016519 | 0.002501 | 0.017179 | 1.000000 |

However, the dataset has high multicollinearity:

In [13]:
```
X1 = prob1_dataset.drop('Y',1)
for i in range(X1.shape[1]):
    print(X1.columns[i], smo.variance_inflation_factor(exog = np.array(X1), exog_idx =
```

```
X1 35.17592699244897
X2 3.1569780596492016
X3 76.30954773017044
X4 179.06398316196183
X5 3.200667559799278
```

After dropping X4, the multicollinearity problem goes away:

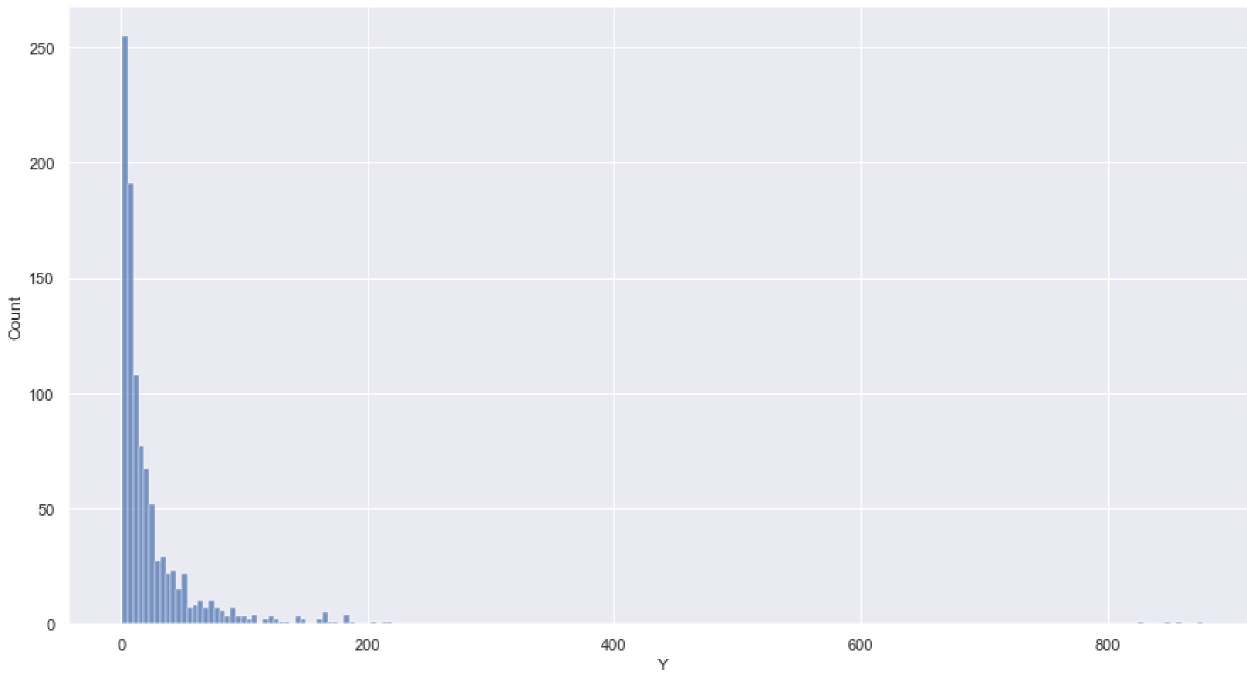
In [14]:
```
prob1_dataset = prob1_dataset.drop('X4', 1)
X2 = prob1_dataset.drop('Y',1)
```

In [15]:
```
for i in range(prob1_dataset.shape[1]-1):
    print(X2.columns[i], smo.variance_inflation_factor(exog = np.array(X2), exog_idx =
```

```
X1 3.2279750799646205
X2 3.154967129310902
X3 3.1128713376633863
X5 3.200086157698921
```

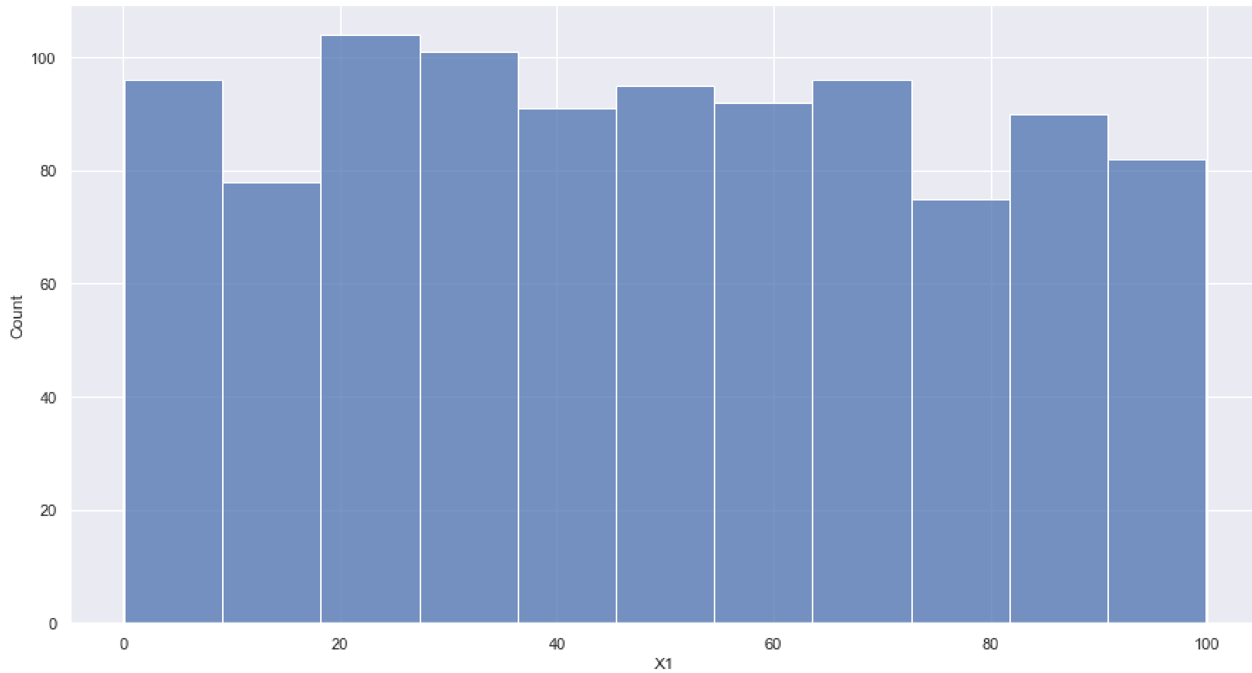
In [16]:
```
sns.set(rc = {'figure.figsize':(15,8)})
sns.histplot(prob1_dataset['Y'])
```

Out[16]: <AxesSubplot:xlabel='Y', ylabel='Count'>



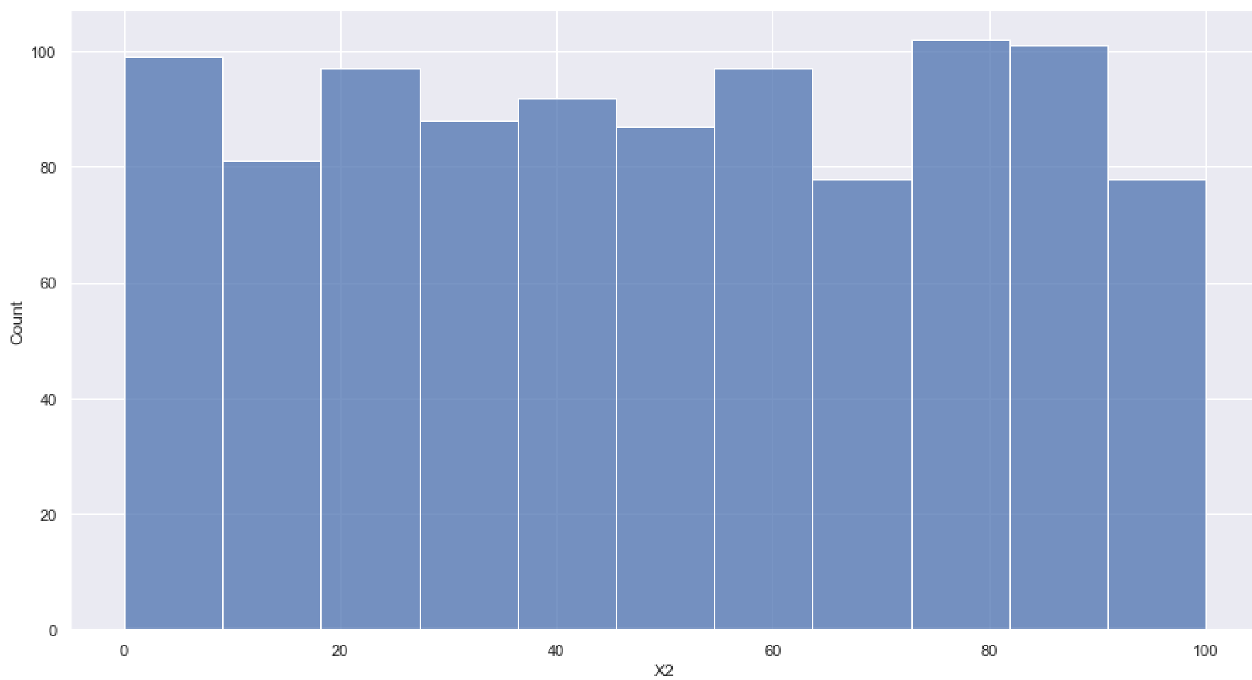

In [17]:
```
sns.histplot(prob1_dataset['X1'])
```

Out[17]: <AxesSubplot:xlabel='X1', ylabel='Count'>



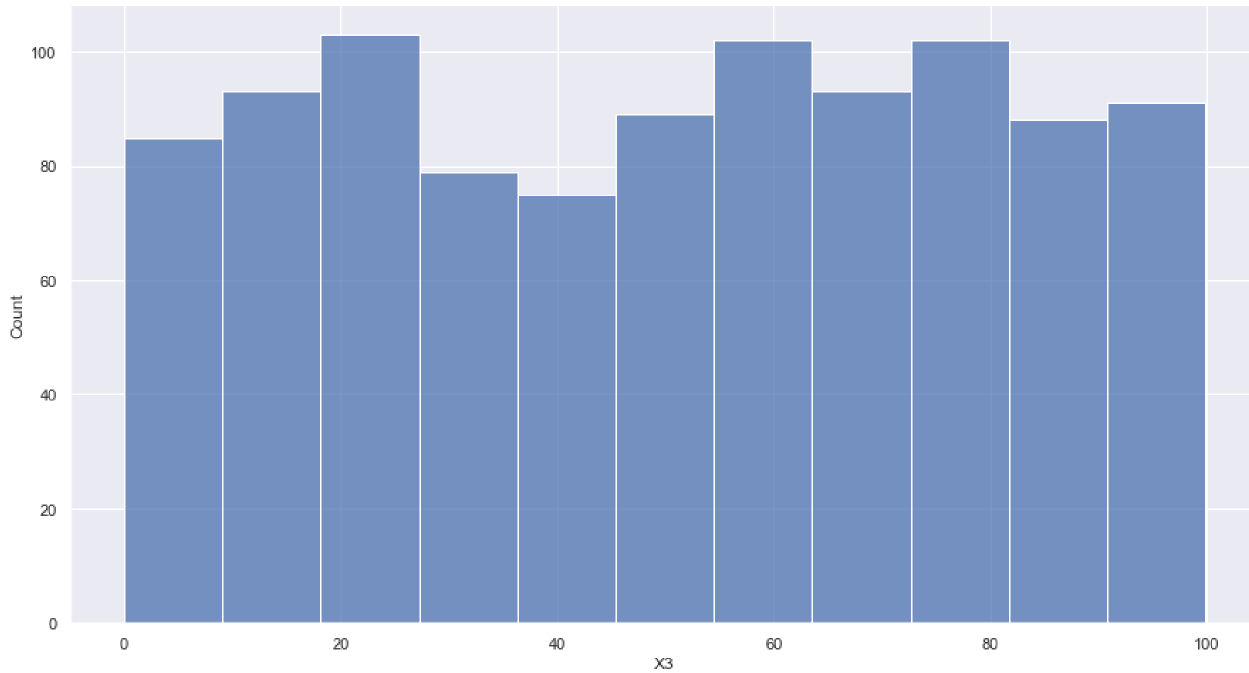

In [18]:
```
sns.histplot(prob1_dataset['X2'])
```

Out[18]: <AxesSubplot:xlabel='X2', ylabel='Count'>



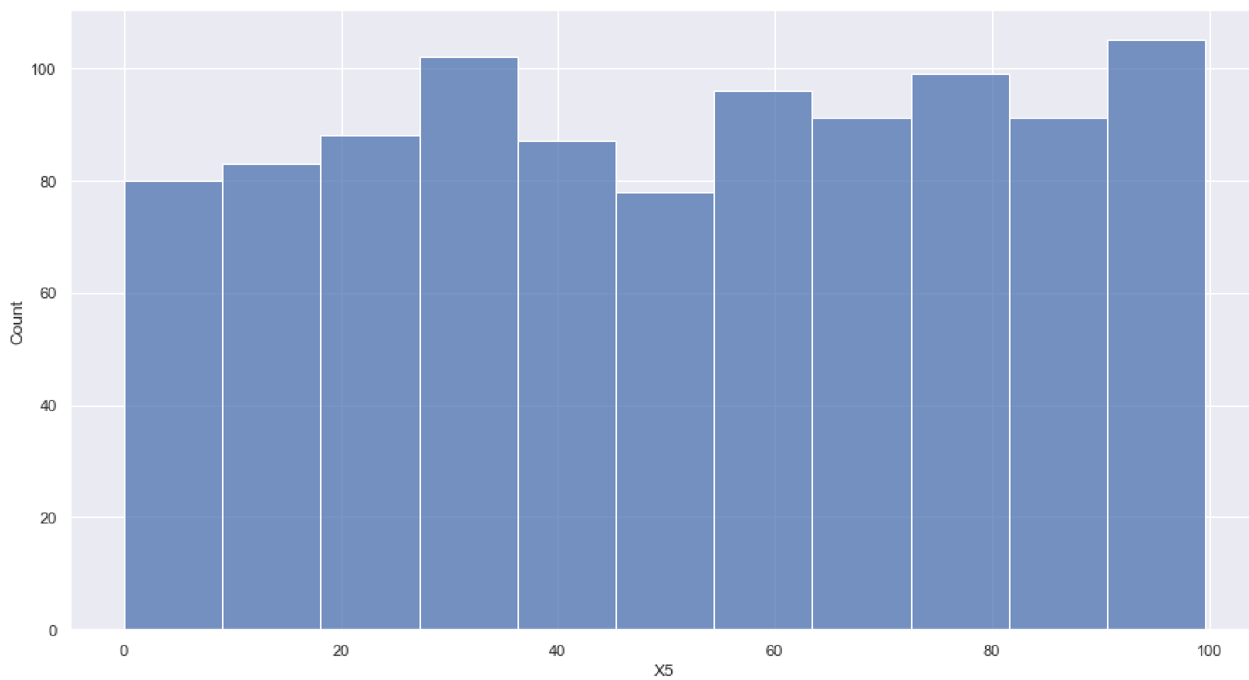

In [19]:
```
sns.histplot(prob1_dataset['X3'])
```

Out[19]: <AxesSubplot:xlabel='X3', ylabel='Count'>




In [20]:
```
sns.histplot(prob1_dataset['X5'])
```

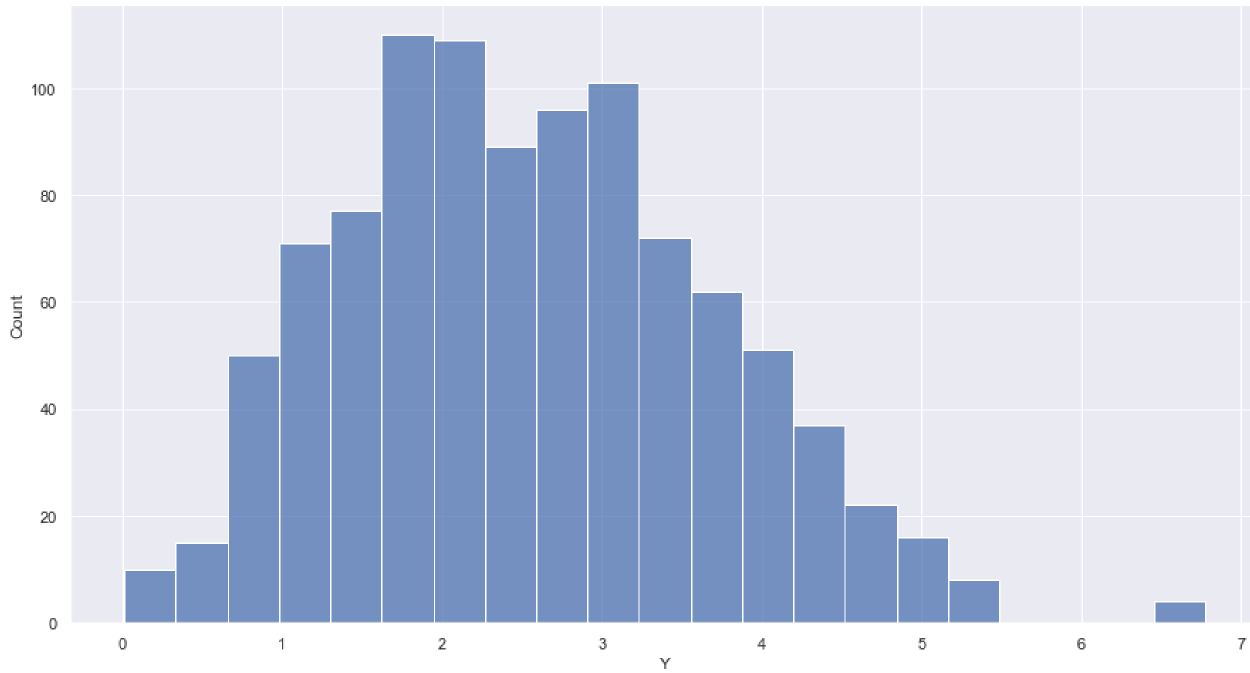
Out[20]: <AxesSubplot:xlabel='X5', ylabel='Count'>




We see that perfoming a log transformation solves the skew problem, so we decide to build a model to predict ln(Y) Our initial model has an $R^2$ of 0.949:

In [21]:
```
sns.histplot(np.log(prob1_dataset['Y']))
```

Out[21]: <AxesSubplot:xlabel='Y', ylabel='Count'>



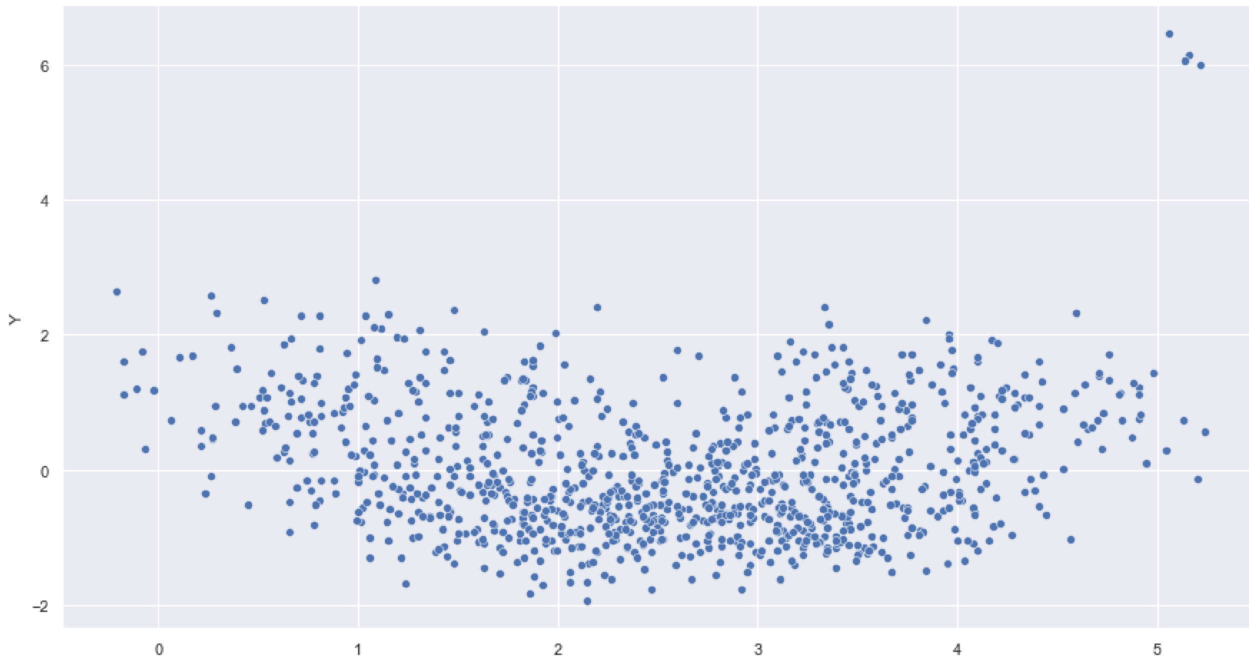

In [22]:
```
model_1 = LinearRegression(fit_intercept = True)
X = prob1_dataset.drop('Y', 1)
y = np.log(prob1_dataset['Y'])
model_1.fit(X = X, y = y)
y_hat = model_1.predict(X)
metrics.r2_score(y, y_hat)
```

Out[22]: 0.9486701002915429

Looking at a standardized residuals plot, we see that there appear to be four outliers with standardized residuals greater than 5, so we remove those from the dataset:

In [23]:
```
rse = np.sqrt(sm.OLS(y,sm.add_constant(X)).fit().mse_resid)
resids = (y - y_hat)/rse
sns.scatterplot(x = y_hat, y = resids)
```

Out[23]: <AxesSubplot:ylabel='Y'>




In [24]:

```
prob1_dataset = prob1_dataset[resids <= 5]
prob1_dataset = prob1_dataset.reset_index(drop=True)
```

In [25]:

```
model_1 = LinearRegression(fit_intercept = True)
X = prob1_dataset.drop('Y', 1)
y = np.log(prob1_dataset['Y'])
model_1.fit(X = X, y = y)
y_hat = model_1.predict(X)
metrics.r2_score(y, y_hat)
```

Out[25]: 0.9543054061305187

In [26]:

```
X
```

Out[26]:

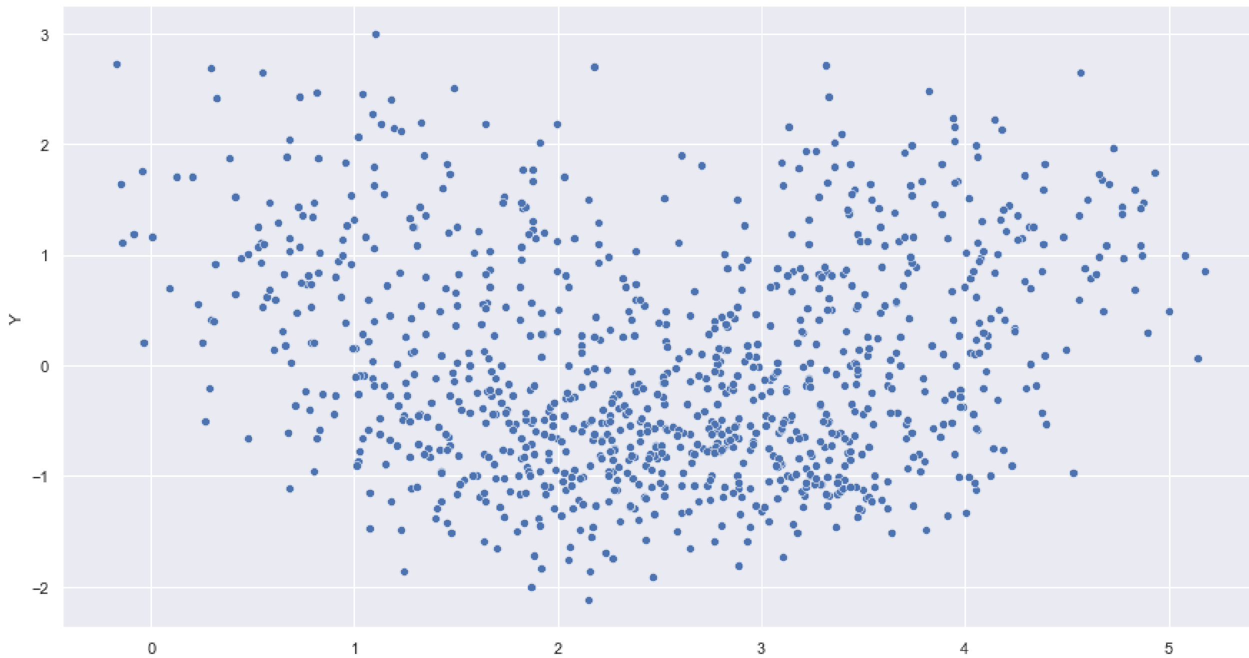
| | X1 | X2 | X3 | X5 |
|---|---|---|---|---|
| **0** | 41.702200 | 72.032449 | 0.011437 | 30.233257 |
| **1** | 14.675589 | 9.233859 | 18.626021 | 34.556073 |
| **2** | 39.676747 | 53.881673 | 41.919451 | 68.521950 |
| **3** | 20.445225 | 87.811744 | 2.738759 | 67.046751 |
| **4** | 41.730480 | 55.868983 | 14.038694 | 19.810149 |
| **...** | ... | ... | ... | ... |
| **991** | 80.014431 | 91.130835 | 51.314901 | 42.942203 |
| **992** | 9.159480 | 27.367730 | 88.664781 | 24.594434 |
| **993** | 62.118939 | 49.500740 | 9.437254 | 89.936853 |
| **994** | 6.777083 | 68.070233 | 97.199814 | 70.937379 |
| **995** | 32.590872 | 7.344017 | 57.973705 | 73.351702 |

996 rows × 4 columns

Now, we see that the residuals plot does not have observations with an SE of greater than 5, but there does appear to be a nonlinearity in the residuals:

In [27]:
```
rse = np.sqrt(sm.OLS(y,sm.add_constant(X)).fit().mse_resid)
resids = (y - y_hat)/rse
sns.scatterplot(x = y_hat, y = resids)
```

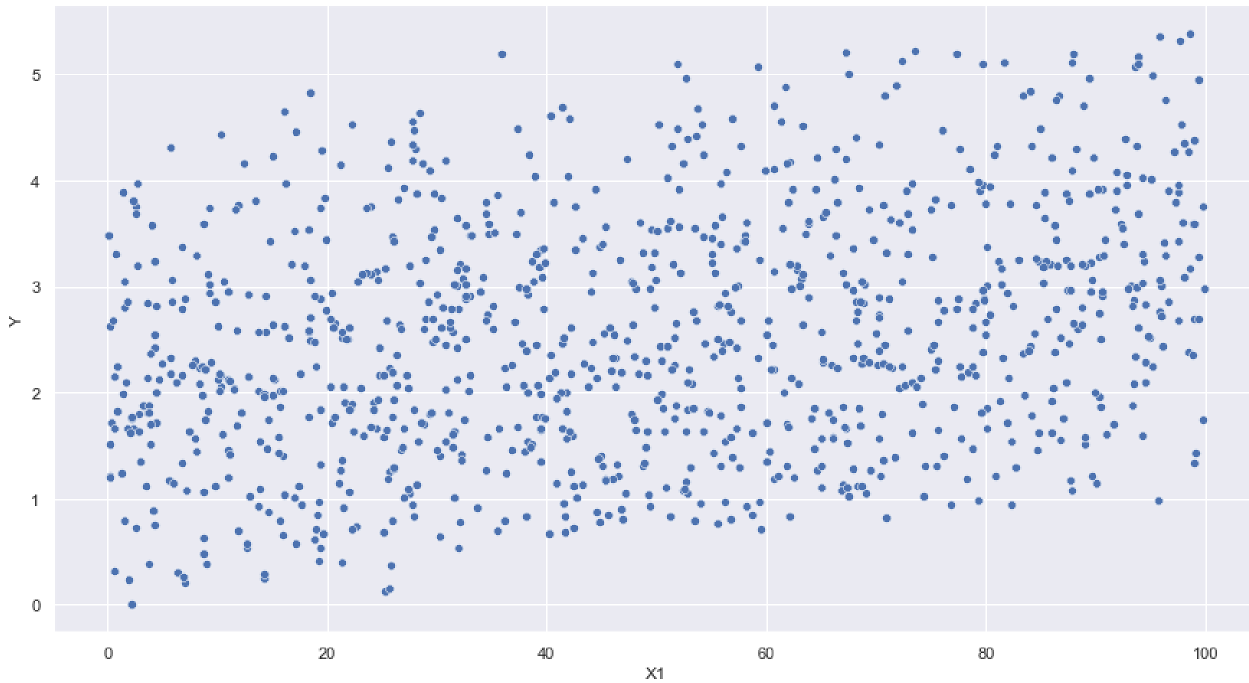
Out[27]: <AxesSubplot:ylabel='Y'>




Investigating the relationships between each predictor and the response:

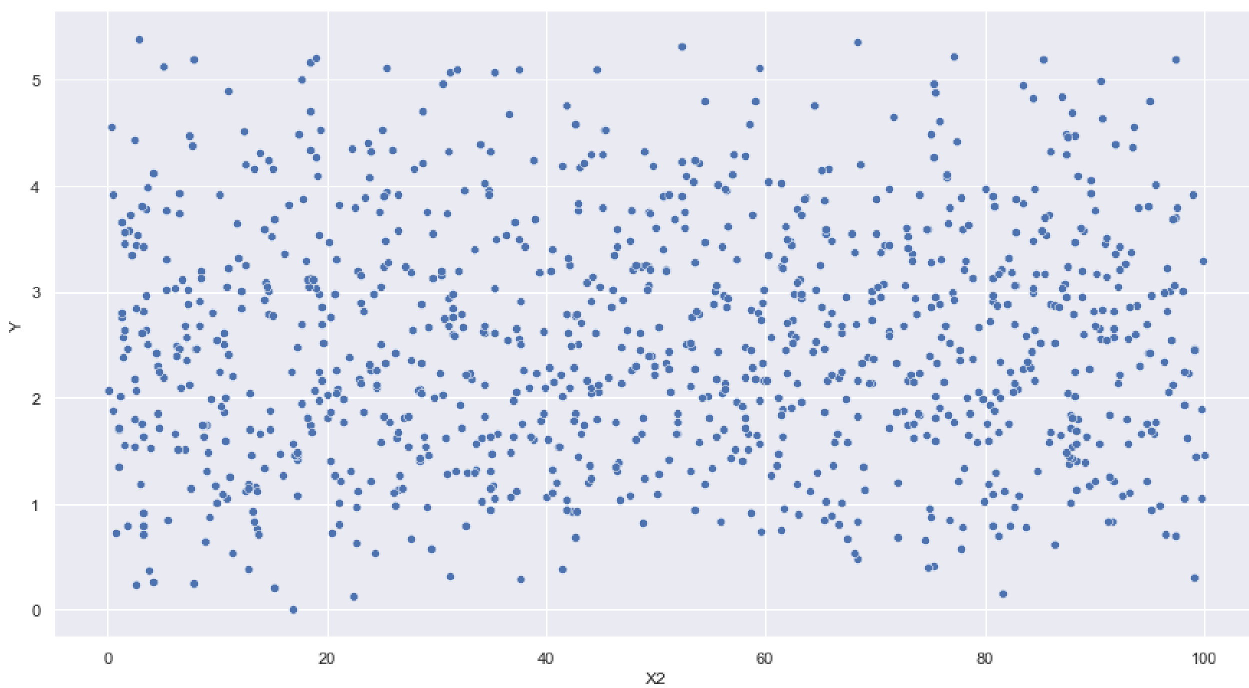
In [28]:
```
sns.scatterplot(x = prob1_dataset['X1'], y=y)
```

Out[28]: <AxesSubplot:xlabel='X1', ylabel='Y'>



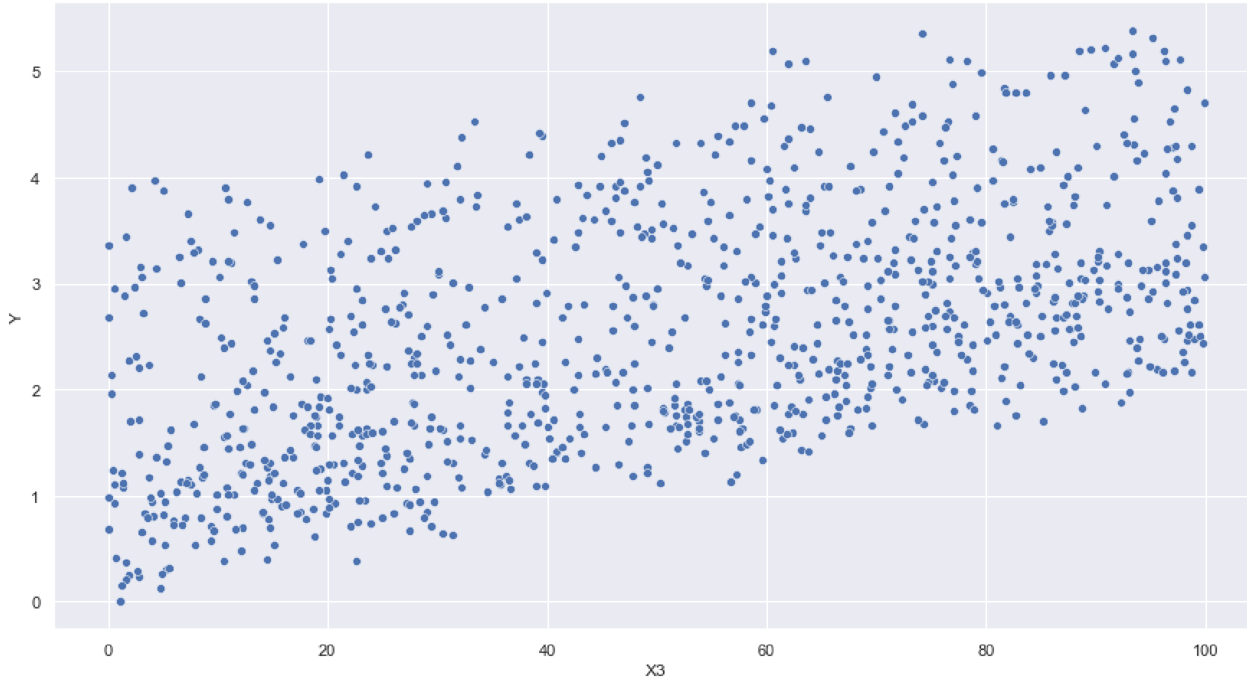

In [29]:
```
sns.scatterplot(x = prob1_dataset['X2'], y=y)
```

Out[29]: <AxesSubplot:xlabel='X2', ylabel='Y'>




In [30]:
```
sns.scatterplot(x = prob1_dataset['X3'], y=y)
```

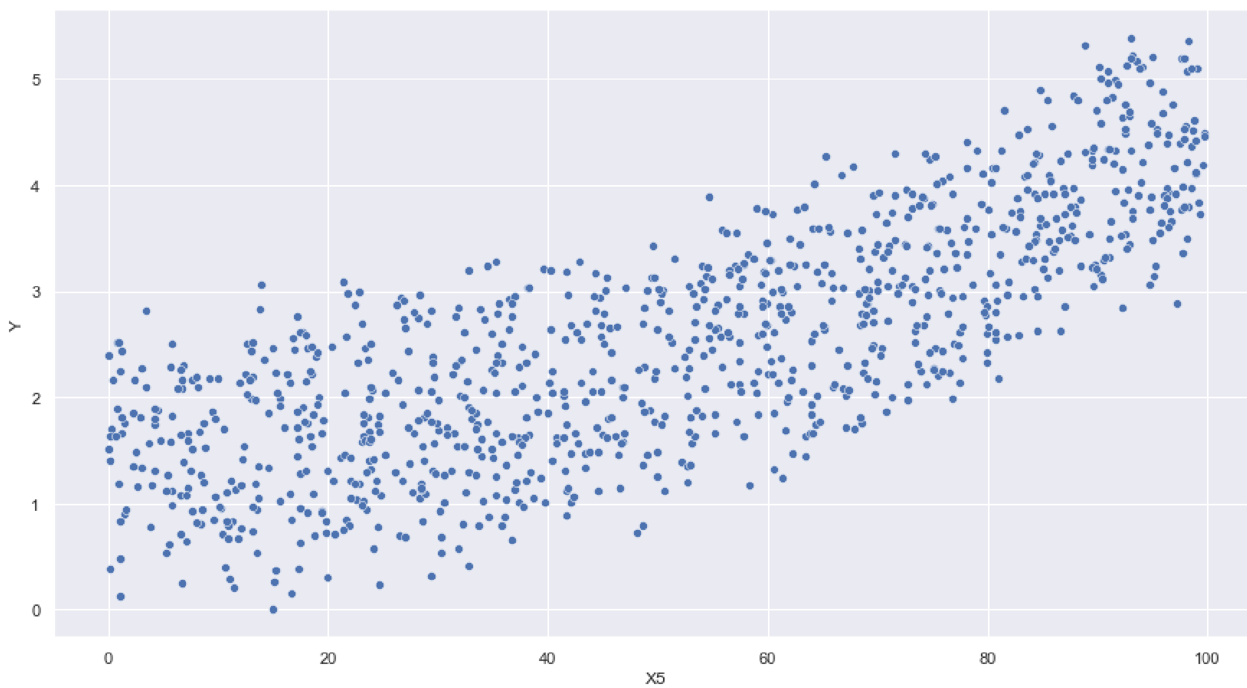
Out[30]: <AxesSubplot:xlabel='X3', ylabel='Y'>




In [31]:

```
sns.scatterplot(x = prob1_dataset['X5'], y=y)
```

Out[31]: <AxesSubplot:xlabel='X5', ylabel='Y'>




It appears that X5 has a nonlinear relationship with Y, so we try adding $X^2$ to the model and get a significant improvement in the model $R^2$:

In [32]:

```
X2 = X.copy()
X2['X5^2'] = X2['X5']**2
model_2 = LinearRegression(fit_intercept = True)
y = np.log(prob1_dataset['Y'])
model_2.fit(X = X2, y = y)
y_hat = model_2.predict(X2)
metrics.r2_score(y, y_hat)
```

Out[32]: 0.9918940416332943

Using statsmodels to find the p-values of the coefficients it appears that X2 and X5 do not have a statistically significant relationship to the response variable

In [33]:

```
sm.OLS(y,sm.add_constant(X2)).fit().summary()
```

Out[33]:

OLS Regression Results

| | | | |
|---|---|---|---|
| **Dep. Variable:** | Y | **R-squared:** | 0.992 |
| **Model:** | OLS | **Adj. R-squared:** | 0.992 |
| **Method:** | Least Squares | **F-statistic:** | 2.423e+04 |
| **Date:** | Mon, 01 Aug 2022 | **Prob (F-statistic):** | 0.00 |
| **Time:** | 13:29:55 | **Log-Likelihood:** | 873.25 |
| **No. Observations:** | 996 | **AIC:** | -1735. |
| **Df Residuals:** | 990 | **BIC:** | -1705. |
| **Df Model:** | 5 | | |
| **Covariance Type:** | nonrobust | | |

| | **coef** | **std err** | **t** | **P>\|t\|** | **[0.025** | **0.975]** |
|---|---|---|---|---|---|---|
| **const** | -0.0201 | 0.014 | -1.476 | 0.140 | -0.047 | 0.007 |
| **X1** | 0.0101 | 0.000 | 89.836 | 0.000 | 0.010 | 0.010 |
| **X2** | 0.0002 | 0.000 | 1.841 | 0.066 | -1.35e-05 | 0.000 |
| **X3** | 0.0201 | 0.000 | 181.390 | 0.000 | 0.020 | 0.020 |
| **X5** | 0.0004 | 0.000 | 0.941 | 0.347 | -0.000 | 0.001 |
| **X5^2** | 0.0003 | 4.37e-06 | 67.755 | 0.000 | 0.000 | 0.000 |

| | | | |
|---|---|---|---|
| **Omnibus:** | 0.725 | **Durbin-Watson:** | 1.916 |
| **Prob(Omnibus):** | 0.696 | **Jarque-Bera (JB):** | 0.686 |
| **Skew:** | -0.064 | **Prob(JB):** | 0.710 |
| **Kurtosis:** | 3.012 | **Cond. No.** | 1.95e+04 |

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The condition number is large, 1.95e+04. This might indicate that there are
strong multicollinearity or other numerical problems.

Remove variables whose coefficient indicate they are not significant and re-run

In [34]:

```
X3 = X2.drop(['X2', 'X5'], 1)
```

```
X3
```

Out[34]:

| | X1 | X3 | X5^2 |
|---|---|---|---|
| **0** | 41.702200 | 0.011437 | 914.049845 |
| **1** | 14.675589 | 18.626021 | 1194.122161 |
| **2** | 39.676747 | 41.919451 | 4695.257637 |
| **3** | 20.445225 | 2.738759 | 4495.266822 |
| **4** | 41.730480 | 14.038694 | 392.442000 |
| **...** | ... | ... | ... |
| **991** | 80.014431 | 51.314901 | 1844.032796 |
| **992** | 9.159480 | 88.664781 | 604.886190 |
| **993** | 62.118939 | 9.437254 | 8088.637553 |
| **994** | 6.777083 | 97.199814 | 5032.111751 |
| **995** | 32.590872 | 57.973705 | 5380.472161 |

996 rows × 3 columns

In [35]:

```
model_3 = LinearRegression(fit_intercept = True)
y = np.log(prob1_dataset['Y'])
model_3.fit(X = X3, y = y)
y_hat = model_3.predict(X3)
metrics.r2_score(y, y_hat)
```

Out[35]: 0.991858415558614

In [36]:

```
sm.OLS(y,sm.add_constant(X3)).fit().summary()
```

Out[36]:

OLS Regression Results

| | | | |
|---|---|---|---|
| **Dep. Variable:** | Y | **R-squared:** | 0.992 |
| **Model:** | OLS | **Adj. R-squared:** | 0.992 |
| **Method:** | Least Squares | **F-statistic:** | 4.028e+04 |
| **Date:** | Mon, 01 Aug 2022 | **Prob (F-statistic):** | 0.00 |
| **Time:** | 13:29:55 | **Log-Likelihood:** | 871.07 |
| **No. Observations:** | 996 | **AIC:** | -1734. |
| **Df Residuals:** | 992 | **BIC:** | -1715. |
| **Df Model:** | 3 | | |
| **Covariance Type:** | nonrobust | | |

| | coef | std err | t | P>\|t\| | [0.025 | 0.975] |
|---|---|---|---|---|---|---|
| **const** | -0.0023 | 0.009 | -0.250 | 0.803 | -0.020 | 0.016 |

| | | | | | | |
|---|---|---|---|---|---|---|
| **X1** | 0.0101 | 0.000 | 90.079 | 0.000 | 0.010 | 0.010 |
| **X3** | 0.0201 | 0.000 | 181.292 | 0.000 | 0.020 | 0.020 |
| **X5^2** | 0.0003 | 1.07e-06 | 280.253 | 0.000 | 0.000 | 0.000 |

| | | | |
|---|---|---|---|
| **Omnibus:** | 0.755 | **Durbin-Watson:** | 1.903 |
| **Prob(Omnibus):** | 0.686 | **Jarque-Bera (JB):** | 0.709 |
| **Skew:** | -0.065 | **Prob(JB):** | 0.701 |
| **Kurtosis:** | 3.016 | **Cond. No.** | 1.29e+04 |

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The condition number is large, 1.29e+04. This might indicate that there are
strong multicollinearity or other numerical problems.

Thus our final model is given by $ln(Y) = 0.01 + 0.01X1 + 0.02X3 + 0.0003X^5$ with an $R^2 = 0.992$

# Validation Techniques

2) Reac the file "Problem 2 Dataset.csv" into a dataframe

In [37]:

```
prob2_dataset = pd.read_csv("Problem 2 Dataset.csv")
```

2a) Fit a linear regression model using the four predictors X1,X2,X3, and X4 to the response variable Y. Do not attempt to improve the model, just use the basic four predictors. Calculate and display mean squared error using the entire dataset for training and for validation.

In [38]:

```
X = prob2_dataset.drop('Y', 1)
y = prob2_dataset['Y']
model2b = LinearRegression(fit_intercept = True)
mse = metrics.mean_squared_error(y_true = y, y_pred = model2b.fit(X, y).predict(X))
formatted_mse = "{:,}".format(mse)
print('MSE:', formatted_mse)
```

```
MSE: 3,514,381.5006588465
```

2B) Now, divide the dataset into a test and training partition using the sklear train_test_split function with an 80/20 split (80% training / 20% test) and calculate the test partition MSE for this model. Set random_state = 0 so that we all get the same answer.

In [39]:

```
from sklearn.model_selection import train_test_split
X_test, X_train, y_test, y_train = train_test_split(X,y, test_size = 0.2, random_state
y_hat_test = model2b.fit(X_train, y_train).predict(X_test)
mse = metrics.mean_squared_error(y_true = y_test, y_pred = y_hat_test)
formatted_mse = "{:,}".format(mse)
print('Test partition MSE:', formatted_mse)
```

```
Test partition MSE: 9,786,625.315344865
```

2c) Without using any additional libraries, perform a k-vold cross validation on the model with 5 folds. Display the resulting mean sequred error.

In [40]:

```
k = 5
num_obs = X.shape[0]
fold_size = np.int_(num_obs/k)
mses = np.zeros(k)
for fold_num in range(k):
    X_test = X[fold_num*fold_size:(fold_num+1)*fold_size]
    y_test = y[fold_num*fold_size:(fold_num+1)*fold_size]
    temp = X_test.index
    X_train = X.drop(temp,0)
    y_train = y.drop(temp,0)
    y_hat_test = model2b.fit(X_train, y_train).predict(X_test)
    mses[fold_num] = metrics.mean_squared_error(y_true = y_test, y_pred = y_hat_test)

mse = sum(mses)/k
formatted_mse = "{:,}".format(mse)
print('K-fold cross validation MSE:',  formatted_mse)
```

```
K-fold cross validation MSE: 3,945,315.490506001
```

2d) Now, use the sklearn cross_val_score function to perform the same calculation and display the resulting mean squared error. If you have done this correctly, your answers to 2c and 2d should be the same.

Documentation on the cross_val_score function can be found at https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.cross_val_score.html

In [42]:

```
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import KFold
k=5
scores = cross_val_score(LinearRegression(), X, y, scoring = 'neg_mean_squared_error',
                         cv=KFold(n_splits=k, shuffle=False))
formatted_mse = "{:,}".format(-sum(scores)/k)
print('K-fold cross validation MSE:',  formatted_mse)
```

```
K-fold cross validation MSE: 3,945,315.490506001
```

In [ ]:

```
```