Get All Imports

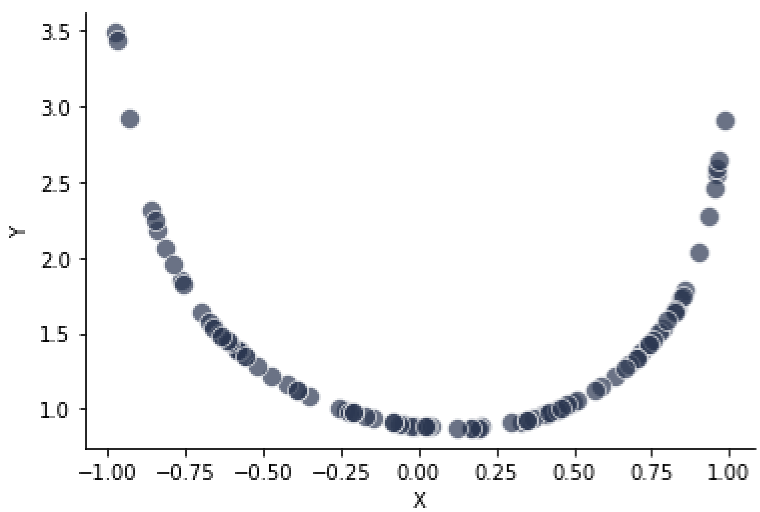
In [ ]:
```
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

Generate Data Points

In [ ]:
```
# Select the maximum polynomial degree
poly_degree = 15
# Set seed so that the same data is generated each time.
# Change this value to change the generated data
np.random.seed(27)

# Select 100 points from a uniform distribution
X = np.random.uniform(-1, 1, 100)
# Generate corresponding coefficients, including bias
coeff = np.random.uniform(-1, 1, poly_degree+1)
# Finally, use the input and coefficients to generate the target variable
Y = np.asarray([sum([coeff[j] * i**j for j in range(1, poly_degree+1)]) + coeff[0] for i

# Plot the data points
sns.scatterplot(x=X, y=Y, marker='o', s=100, color="#2B3751", edgecolors="#E5E5E5", alph
sns.despine()
plt.xlabel("X")
plt.ylabel("Y")
plt.show()
```



Define the linear regression model using Pseudo Inverse.

In [ ]:
```
class LinearRegression():
    # Constructor of this class
    def __init__(self):
        self.coeff = list()

    # This function is used to find the coefficients for the line of best fit
    def fit(self, A, Y):
```

```
        # Add Bias
        A = np.concatenate((np.ones((len(A), 1)), A), axis=1)
        # Find Pseudo Inverse
        pseudo_inv = np.matmul(np.linalg.inv(np.matmul(np.transpose(A), A)), np.transpos
        # Finally get the coefficients
        self.coeff = np.matmul(pseudo_inv, np.reshape(Y, (-1, 1)))

    # This function uses the found coefficients to deliver predictions
    def predict(self, A):
        A = np.concatenate((np.ones((len(A), 1)), A), axis=1)
        return np.squeeze(np.matmul(A, self.coeff))
```

Apply Piece-wise linear Regression

In [ ]:

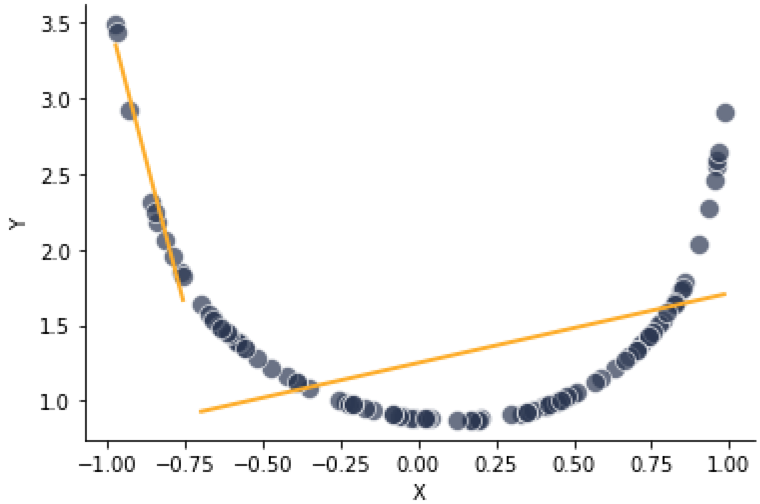
```
# Select a threshold to perform piecewise linear regression
threshold = -0.75

# Select all data points below set threshold
indices1 = np.where(X < threshold)[0]
x, y = X[indices1], Y[indices1]
x = np.reshape(x, (-1, 1))
# Now apply linear regression as usual
model = LinearRegression()
model.fit(x, y)
preds1 = model.predict(x)
mse = np.sum(np.square(preds1-y))

# Select all data points above set threshold
indices2 = np.where(X >= threshold)[0]
x, y = X[indices2], Y[indices2]
# Another feature is added which is the original feature value minus the threshold
x = np.concatenate((np.reshape(x, (-1, 1)), np.reshape(x-threshold, (-1, 1))), axis=1)
# Now apply linear regression
model = LinearRegression()
model.fit(x, y)
preds2 = model.predict(x)
mse+= np.sum(np.square(preds2-y))

print("Mean Squared Error:", mse/len(Y))
# Plot the predicted points
sns.scatterplot(x=X, y=Y, marker='o', s=100, color="#2B3751", edgecolors="#E5E5E5", alph
sns.lineplot(x=X[indices1], y=preds1, color="#FDAC29", linewidth=2)
sns.lineplot(x=X[indices2], y=preds2, color="#FDAC29", linewidth=2)
sns.despine()
plt.xlabel("X")
plt.ylabel("Y")
plt.show()
```

```
Mean Squared Error: 0.16464797275216975
```



Apply Quadratic Piece-wise Linear Regression

In [ ]:

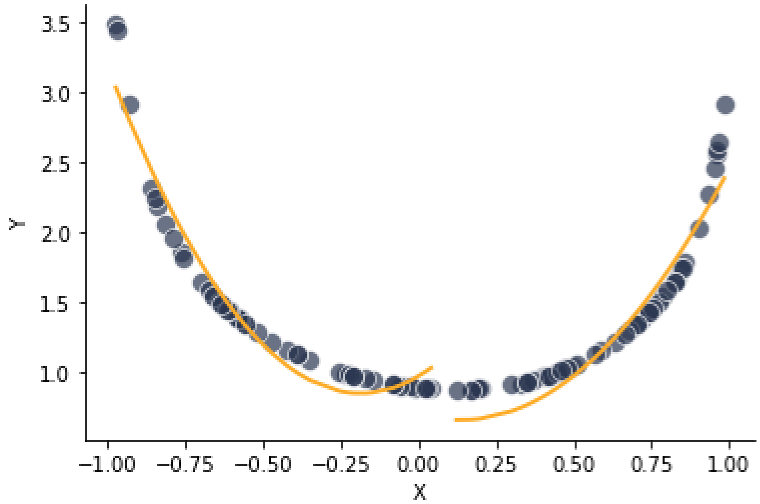
```
# Select a threshold to perform piecewise linear regression
threshold = 0.1

# Select all data points below set threshold
indices1 = np.where(X < threshold)[0]
x, y = X[indices1], Y[indices1]
x = np.concatenate((np.reshape(x, (-1, 1)), np.reshape(x**2, (-1, 1))), axis=1)
# Now apply linear regression as usual
model = LinearRegression()
model.fit(x, y)
preds1 = model.predict(x)
mse = np.sum(np.square(preds1-y))

# Select all data points above set threshold
indices2 = np.where(X >= threshold)[0]
x, y = X[indices2], Y[indices2]
# Another feature is added which is the original feature value minus the threshold
x = np.concatenate((np.reshape(x, (-1, 1)), np.reshape(x**2, (-1, 1)), np.reshape(x-thre
# Now apply linear regression
model = LinearRegression()
model.fit(x, y)
preds2 = model.predict(x)
mse+= np.sum(np.square(preds2-y))

print("Mean Squared Error:", mse/len(Y))
# Plot the predicted points
sns.scatterplot(x=X, y=Y, marker='o', s=100, color="#2B3751", edgecolors="#E5E5E5", alph
sns.lineplot(x=X[indices1], y=preds1, color="#FDAC29", linewidth=2)
sns.lineplot(x=X[indices2], y=preds2, color="#FDAC29", linewidth=2)
sns.despine()
plt.xlabel("X")
plt.ylabel("Y")
plt.show()
```

```
Mean Squared Error: 0.023466131095164128
```



In [ ]: