EDS232 Lab 1: Regression

Overview

In this lab, we will introduce the basics of machine learning in **Python** with **regression** algorithms, a core technique used to predict continuous outcomes. We will use the popular **scikit-learn** library, which provides easy-to-use tools for building and evaluating machine learning models.

Specifically, we will learn how regression algorithms can help us model and predict water quality data.

Objectives

By the end of this lab, you will be able to:

- Understand the concept of regression
- Implement regression models in Python
- Evaluate model performance using **R^2** and **MSE**
- Visualize regression prediction results

Key Concepts

- **Regression**: A machine learning method for predicting continuous values.
    - **Simple Linear Regression**: A regression model with one independent variable.
    - **Polynomial Regression**: A regression model which models the relationship between X and Y as an n-degree polynomial.
- **Scikit-learn**: A Python library that provides simple and efficient tools for data mining and machine learning. We will use it for:
    - **Data Preprocessing**: Preparing data for the model.
    - **Model Training**: Fitting the regression model to our data.
    - **Model Evaluation**: Assessing model performance using model evaluation metrics.
- **Model Evaluation Metrics**: Tools to assess how well our model fits the data, such as:

- **R^2 (R-squared)**: Measures the proportion of variance in the dependent variable that is predictable from the independent variable(s).
- **MSE (Mean Square Error)**: The average squared differences between predicted and actual values.

About the data

Hurricane Irene caused extensive flood and wind damage as it traveled across the Caribbean and up the East coast of the United States. The Hurricane made landfall in the United States near Cape Lookout, North Carolina on August 27th, 2011 and was downgraded to a Tropical Storm by the time it hit the New York City region on Sunday, August 28th, 2011.

A dataset from the Hudson River Environmental Conditions Observing System (HRECOS) offers a detailed look at the effects of Hurricane Irene on the river's ecosystem through high-frequency, 15-minute interval measurements over a ten-day period. It includes variables critical to understanding ecological health, such as water temperature, dissolved oxygen, turbidity, depth, and meteorological data like rainfall and wind speed. Analyzing these variables helps answer questions about how extreme weather events like hurricanes can disrupt river ecosystems and impact water quality. **You can access the data and metadata here.**

Step 1: Import libraries and load data

Load libraries

In [ ]:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score
import sklearn.linear_model
from sklearn.preprocessing import PolynomialFeatures

np.random.seed(42)
```

```
/Users/takeenshamloo/opt/anaconda3/lib/python3.8/site-packages/scipy/__init__.py:138: UserWarning: A NumPy vers
ion >=1.16.5 and <1.23.0 is required for this version of SciPy (detected version 1.24.4)
  warnings.warn(f"A NumPy version >={np_minversion} and <{np_maxversion} is required for this version of "
```

Load the data

Turbidity levels in water can be significantly affected by major weather events such as hurricanes. Turbidity measures how much light is blocked by particles in water. In an event like a hurricane, we expect wind and rainfall to bring in suspended particles, increasing

the turbidity of the water. When light is blocked by particles in water, oxygen production is impacted as well. When a natural disaster like a hurricane alters turbidity levels in a body of water, how is dissolved oxygen impacted? Let's find out.

In this lab, we are interested in the turbidity and dissolved oxygen variables. Read the data into the `hurricane_do` and `hurricane_turbidity` variables. Then, merge these two dataframes. Store the result in the `df` variable. Drop the columns that contain data for Piermont, as Piermont does not contain any turbidity data. We are only interested in the Port of Albany and Norrie Point for this lab.

*Notice that the data is not a csv file and is instead a **.xlsx** file! Use the `pandas.read_excel` function to read in your data. You can find more documentation on reading in .xlsx files here.*

In [ ]:
```
# Storing the file path to a variable for reuse.
file_path = '../Week1/data/Hurricane Irene and the Hudson River.xlsx'

# Reading in the 6th sheet of our xlsx file and drop Piermont.
hurricane_do = pd.read_excel(file_path, sheet_name = 5).drop(['Piermont D.O. (ppm)'], axis = 1)

# Read in the turbidity shee of our xlsx file and drop Piermont.
hurricane_turbidity = pd.read_excel(file_path, sheet_name = 'Turbidity').drop(['Piermont Turbidity in NTU'], ax

# Merge both tables on Date Time.
df = hurricane_do.merge(hurricane_turbidity, on = 'Date Time (ET)')
```

Step 2: Explore and clean the data

Do some initial exploratory analysis on the data. Check out what type of data you are working with, and plot your data. Write a few sentences on your findings.

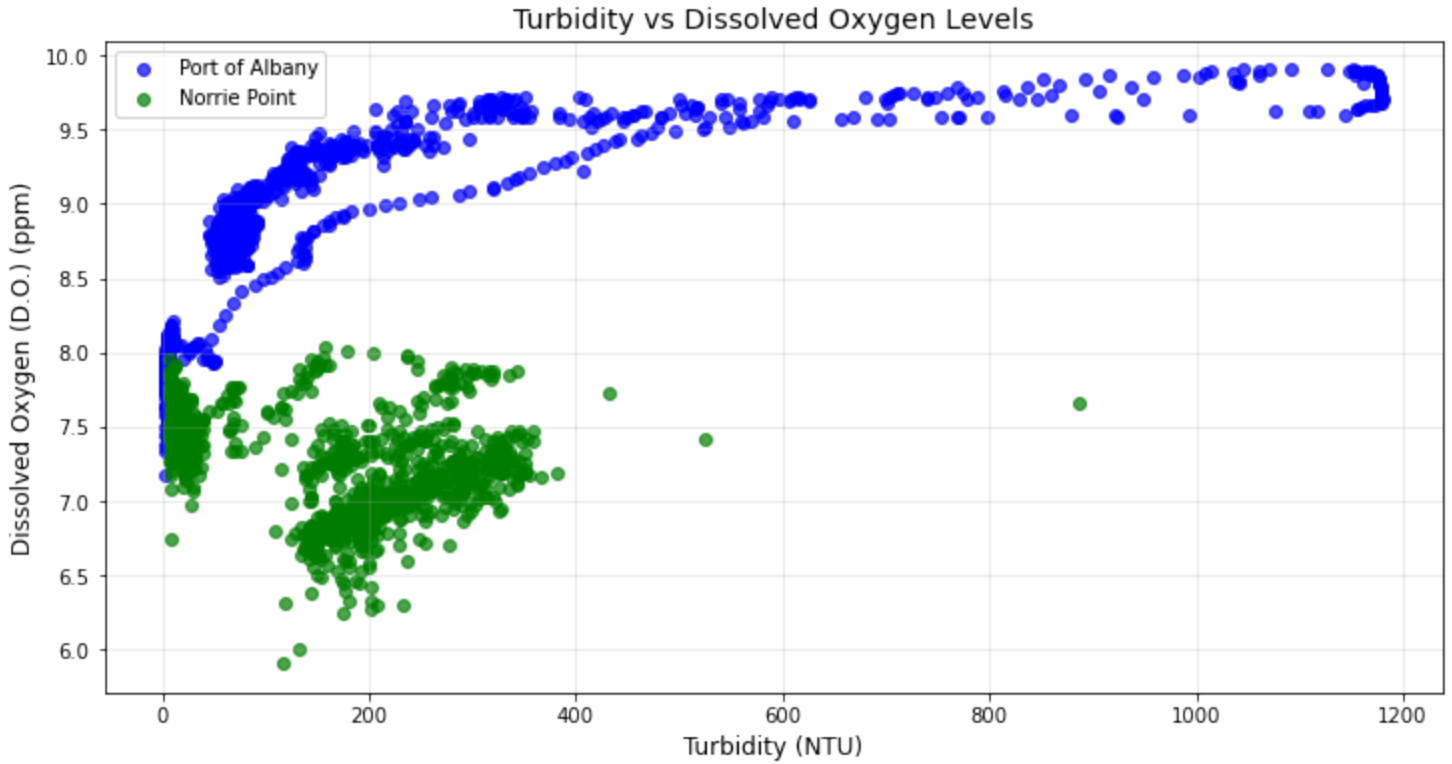
In [ ]:
```
# Show all the types of our columns.
print(df.dtypes)

# Show the first 5 elements of our data.
print(df.head())

# Checking for missing values
print(df.isnull().sum())

# Display a summary of the DataFrame.
print(df.info)
```

```
# Display the number of unique values for each column.
print(df.nunique)

print(df.columns)

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

# Plot for Port of Albany
plt.scatter(
    df[' Port of Albany Turbidity in NTU'],
    df[' Port of Albany D.O. (ppm)'],
    alpha=0.7,
    label='Port of Albany',
    color='blue'
)

# Plot for Norrie Point
plt.scatter(
    df['Norrie Point Turbidity in NTU'],
    df['Norrie Point D.O. (ppm)'],
    alpha=0.7,
    label='Norrie Point',
    color='green'
)

# Adding labels, title, and legend
plt.title('Turbidity vs Dissolved Oxygen Levels', fontsize=14)
plt.xlabel('Turbidity (NTU)', fontsize=12)
plt.ylabel('Dissolved Oxygen (D.O.) (ppm)', fontsize=12)
plt.legend()
plt.grid(alpha=0.3)
plt.show()
```

```
Date Time (ET)                      datetime64[ns]
 Port of Albany D.O. (ppm)                 float64
Norrie Point D.O. (ppm)                    float64
 Port of Albany Turbidity in NTU           float64
Norrie Point Turbidity in NTU              float64
dtype: object
       Date Time (ET)   Port of Albany D.O. (ppm)  Norrie Point D.O. (ppm)  \
0 2011-08-25 00:00:00                        7.68                     7.81
1 2011-08-25 00:15:00                        7.60                     7.73
2 2011-08-25 00:30:00                        7.57                     7.63
3 2011-08-25 00:45:00                        7.72                     7.67
4 2011-08-25 01:00:00                        7.74                     7.63
```

```
    Port of Albany Turbidity in NTU  Norrie Point Turbidity in NTU
0                               4.0                            9.3
1                               3.9                            8.4
2                               4.3                            7.9
3                               4.7                            8.1
4                               4.4                            8.4
Date Time (ET)                      0
 Port of Albany D.O. (ppm)          0
Norrie Point D.O. (ppm)             0
 Port of Albany Turbidity in NTU    0
Norrie Point Turbidity in NTU       0
dtype: int64
<bound method DataFrame.info of           Date Time (ET)   Port of Albany D.O. (ppm)  Norrie Point D.O. (ppm)
\
0    2011-08-25 00:00:00                     7.68                    7.81
1    2011-08-25 00:15:00                     7.60                    7.73
2    2011-08-25 00:30:00                     7.57                    7.63
3    2011-08-25 00:45:00                     7.72                    7.67
4    2011-08-25 01:00:00                     7.74                    7.63
...                  ...                      ...                     ...
1147 2011-09-05 22:45:00                     8.73                    6.84
1148 2011-09-05 23:00:00                     8.76                    6.78
1149 2011-09-05 23:15:00                     8.66                    6.83
1150 2011-09-05 23:30:00                     8.75                    6.79
1151 2011-09-05 23:45:00                     8.68                    6.78

       Port of Albany Turbidity in NTU  Norrie Point Turbidity in NTU
0                                  4.0                            9.3
1                                  3.9                            8.4
2                                  4.3                            7.9
3                                  4.7                            8.1
4                                  4.4                            8.4
...                                ...                            ...
1147                              47.2                          144.1
1148                              56.7                          139.7
1149                              47.0                          141.2
1150                              48.7                          127.9
1151                              49.5                          149.0

[1152 rows x 5 columns]>
<bound method DataFrame.nunique of           Date Time (ET)   Port of Albany D.O. (ppm)  Norrie Point D.O. (pp
m)  \
0    2011-08-25 00:00:00                     7.68                    7.81
1    2011-08-25 00:15:00                     7.60                    7.73
2    2011-08-25 00:30:00                     7.57                    7.63
3    2011-08-25 00:45:00                     7.72                    7.67
4    2011-08-25 01:00:00                     7.74                    7.63
...                  ...                      ...                     ...
```

```
1147 2011-09-05 22:45:00                      8.73                      6.84
1148 2011-09-05 23:00:00                      8.76                      6.78
1149 2011-09-05 23:15:00                      8.66                      6.83
1150 2011-09-05 23:30:00                      8.75                      6.79
1151 2011-09-05 23:45:00                      8.68                      6.78

       Port of Albany Turbidity in NTU  Norrie Point Turbidity in NTU
0                                  4.0                            9.3
1                                  3.9                            8.4
2                                  4.3                            7.9
3                                  4.7                            8.1
4                                  4.4                            8.4
...                                ...                            ...
1147                              47.2                          144.1
1148                              56.7                          139.7
1149                              47.0                          141.2
1150                              48.7                          127.9
1151                              49.5                          149.0

[1152 rows x 5 columns]>
Index(['Date Time (ET)', ' Port of Albany D.O. (ppm)',
       'Norrie Point D.O. (ppm)', ' Port of Albany Turbidity in NTU',
       'Norrie Point Turbidity in NTU'],
      dtype='object')
```



The dataset contains 1152 rows with no missing values, and the column types are appropriate for analysis. The scatter plot shows a clear relationship between turbidity and dissolved oxygen levels, with data from Port of Albany (blue) exhibiting a structured trend and stable values while data from Norrie Point (green) demonstrates more variability and some outliers. There could be potential differences in environmental conditions or influencing factors between the two locations thats causesing these outliers that we are unaware of.

When you were exploring the data, you may have noticed that the column names aren't the cleanest. Update the column names to the following : `date` , `albany_DO` , `norrie_DO` , `albany_turbidity` , `norrie_turbidity` (**make sure your column names are in that order!!**).

In [ ]:
```
# Change all the column names to the appropriate cleaned version.
df.columns = ['date', 'albany_DO', 'norrie_DO', 'albany_turbidity', 'norrie_turbidity']

df # Check to make sure column names were updated.
```

Out[ ]:

	date	albany_DO	norrie_DO	albany_turbidity	norrie_turbidity
0	2011-08-25 00:00:00	7.68	7.81	4.0	9.3
1	2011-08-25 00:15:00	7.60	7.73	3.9	8.4
2	2011-08-25 00:30:00	7.57	7.63	4.3	7.9
3	2011-08-25 00:45:00	7.72	7.67	4.7	8.1
4	2011-08-25 01:00:00	7.74	7.63	4.4	8.4
...	...	...	...	...	...
1147	2011-09-05 22:45:00	8.73	6.84	47.2	144.1
1148	2011-09-05 23:00:00	8.76	6.78	56.7	139.7
1149	2011-09-05 23:15:00	8.66	6.83	47.0	141.2
1150	2011-09-05 23:30:00	8.75	6.79	48.7	127.9
1151	2011-09-05 23:45:00	8.68	6.78	49.5	149.0

1152 rows × 5 columns

Step 3: Prepare the data for machine learning

It is time to split our data into training and testing data for our linear regression model. The `train_test_split` function from the `sklearn.model_selection` module will let us accomplish this.

The `train_test_split` function takes two inputs: X and Y, and produces four outputs: X_train, X_test, Y_train, and Y_test. It also takes two parameters, `test_size`, which specifies the proportion of data to be used in the testing set and `random_state`.

This process allows us to train the model on a subset of the data (training set) and then evaluate its performance and generalizability on unseen data (testing set). By doing this, we can assess how well the model predicts dissolved oxygen levels based on turbidity after a storm.

Select your data such that `albany_turbidity` is your feature or independent variable (X) and `albany_DO` is your target or dependent variable (Y). Then split it using `train_test_split`.

Use a test size of 0.33 and a random state of 42.

In [ ]:
```
# Select features and target
X = df['albany_turbidity']
Y = df['albany_DO']

# Create train/test split
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size = 0.33, random_state = 42)

# Reshape train/test array to a 2D array with 1 column
X_train = np.array(X_train).reshape(-1, 1)
X_test = np.array(X_test).reshape(-1, 1)
```

Step 4: Select your model

We are going to use linear regression to predict the turbidity in Albany. Is linear regression a good model to pick to achieve this goal? Answer in the markdown cell below.

Answer: The plot shows from our exploratory analysis showed that Albany, dissolved oxygen levels stay fairly consistent around 10 ppm even as turbidity increases. At Norrie Point, the relationship is less stable suggesting other factors might be affecting dissolved oxygen. This shows that linear regressions could be a good starting point.

In [ ]:
```
# Initialize and fit the model
model = LinearRegression().fit(X_train, Y_train)
```

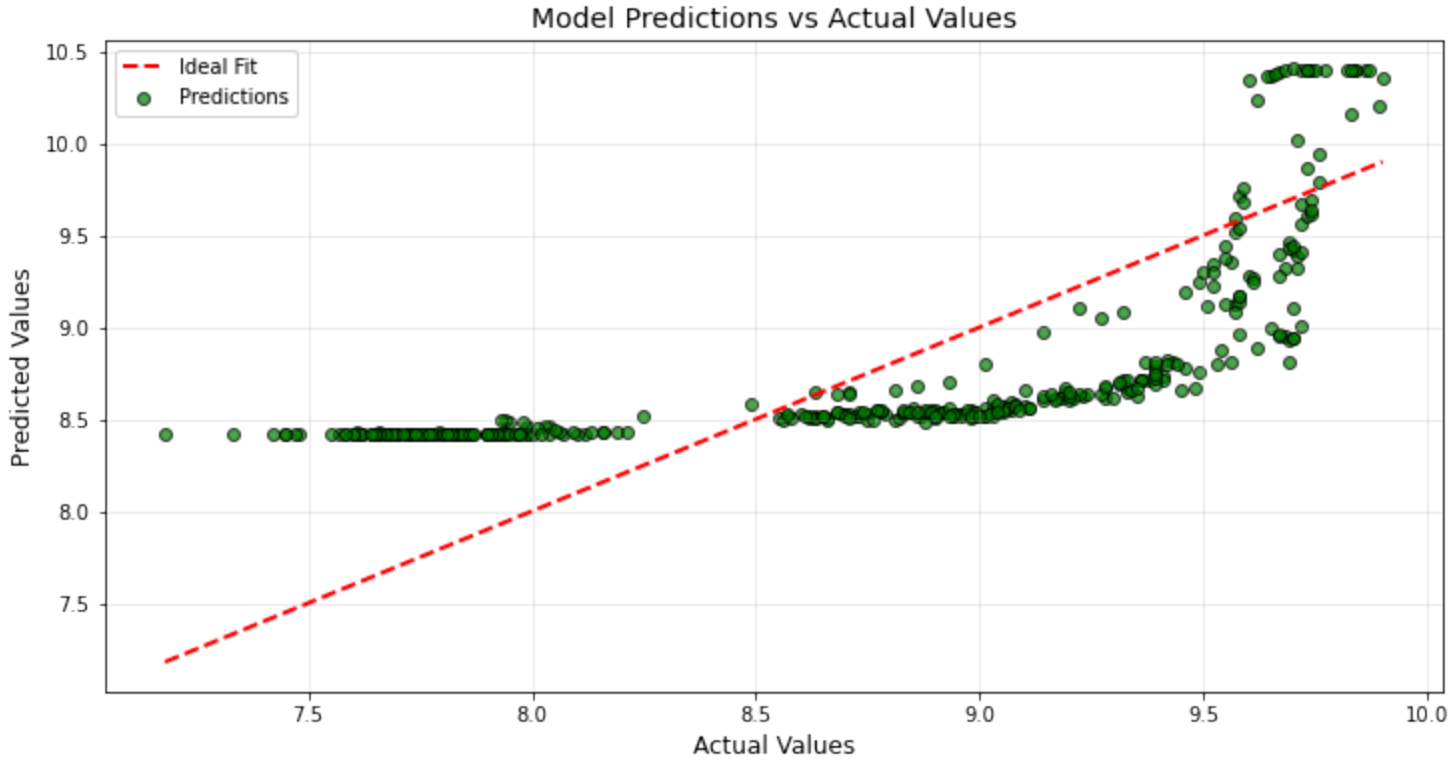
Step 5: Evaluate the model

Now it's time to see how well our model does on this task. To accomplish this, make predictions with your model on the test data and then check its performance by examining the MSE and the R^2 score. Because we held the test data out from the training process, these predictions give us an idea of how our model performs on unseen data. Then visualize your model's performance by creating a scatter plot of the Y predictions and your Y test data.

In [ ]:
```
# Make predictions
Y_pred = model.predict(X_test)

# Calculate evaluation metrics using scikit-learn's mean_squared_error and r2_score
mse = np.sqrt(mean_squared_error(Y_test, Y_pred))
r2 = r2_score(Y_test, Y_pred)

print(f"Mean Squared Error: {mse}")
```

```
print(f"R² Score: {r2}")

# Visualize predictions vs. actual values
plt.figure(figsize = (12, 6))
plt.scatter(Y_test, Y_pred, alpha = 0.7, color = 'green', edgecolor = 'k')
plt.plot([Y_test.min(), Y_test.max()], [Y_test.min(), Y_test.max()], 'r--', lw = 2, label = 'Ideal Fit')
plt.title('Model Predictions vs Actual Values', fontsize = 14)
plt.xlabel('Actual Values', fontsize = 12)
plt.ylabel('Predicted Values', fontsize = 12)
plt.legend(['Ideal Fit', 'Predictions'])
plt.grid(alpha = 0.3)
plt.show()
```

```
Mean Squared Error: 0.5320298550773815
R² Score: 0.48988874519870107
```



Step 6: Present the Solution

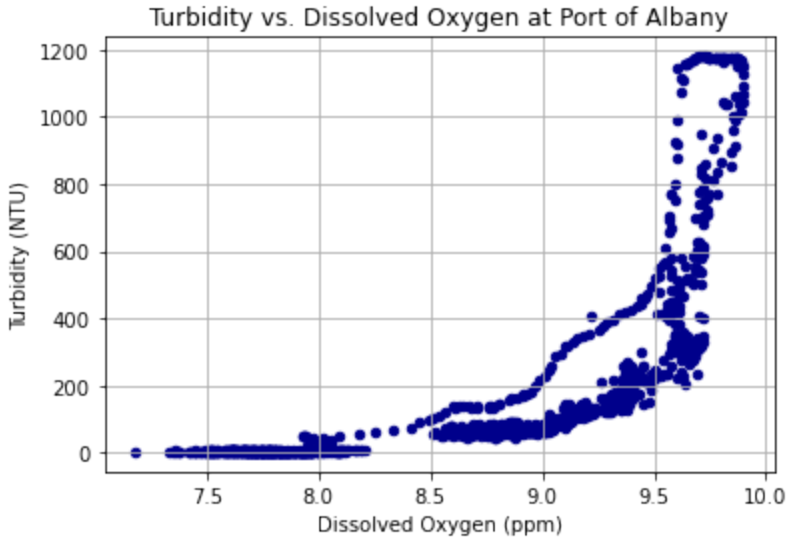
In the markdown cell below, discuss how your model performed overall. If the model performed poorly, why do you think it did so? If it performed well, why do you think it did so? What could future analysis include?

Answer: The model did okay, explaining about 49% of the variation in dissolved oxygen levels, but it struggled with higher values probably due to the relationship not being perfectly linear.

Before we selected our algorithm, we should have looked at the data for evidence of a linear relationship between variables. Let's check now!

In [ ]:
```
plt.figure(figsize=(10, 6))  # Setting the figure size for better visibility
df.plot.scatter(x='albany_DO', y='albany_turbidity', c='DarkBlue')

plt.title('Turbidity vs. Dissolved Oxygen at Port of Albany')
plt.xlabel('Dissolved Oxygen (ppm)')
plt.ylabel('Turbidity (NTU)')
plt.grid(True)
plt.show()
```

```
<Figure size 720x432 with 0 Axes>
```



Step 7: Check to see if polynomial regression performs better

We assumed linear regression would work well with our data, but this data doesn't look very linear. It's a good reminder of the importance of exploratory analysis. Let's check to see how a polynomial regression performs in comparison. Transform the features for polynomial regression. Use the `PolynomialFeatures` class from the `sklearn.preprocessing` module.

In [ ]:
```
# Transform features to include polynomial terms (degree 2 for quadratic terms)
poly = PolynomialFeatures(degree = 2)
X_poly_train = poly.fit_transform(X_train)
X_poly_test = poly.transform(X_test)

# View the transformed feature set (for insight)
print(X_poly_train)
```

```
[[1.00000e+00 5.00000e+00 2.50000e+01]
 [1.00000e+00 7.51000e+01 5.64001e+03]
 [1.00000e+00 7.39000e+02 5.46121e+05]
 ...
 [1.00000e+00 6.29000e+01 3.95641e+03]
 [1.00000e+00 8.18000e+01 6.69124e+03]
 [1.00000e+00 5.53000e+01 3.05809e+03]]
```

Step 8: Fit your model on the polynomial features

In [ ]:
```
# Train the model on polynomial features
poly_model = LinearRegression().fit(X_poly_train, Y_train)
```

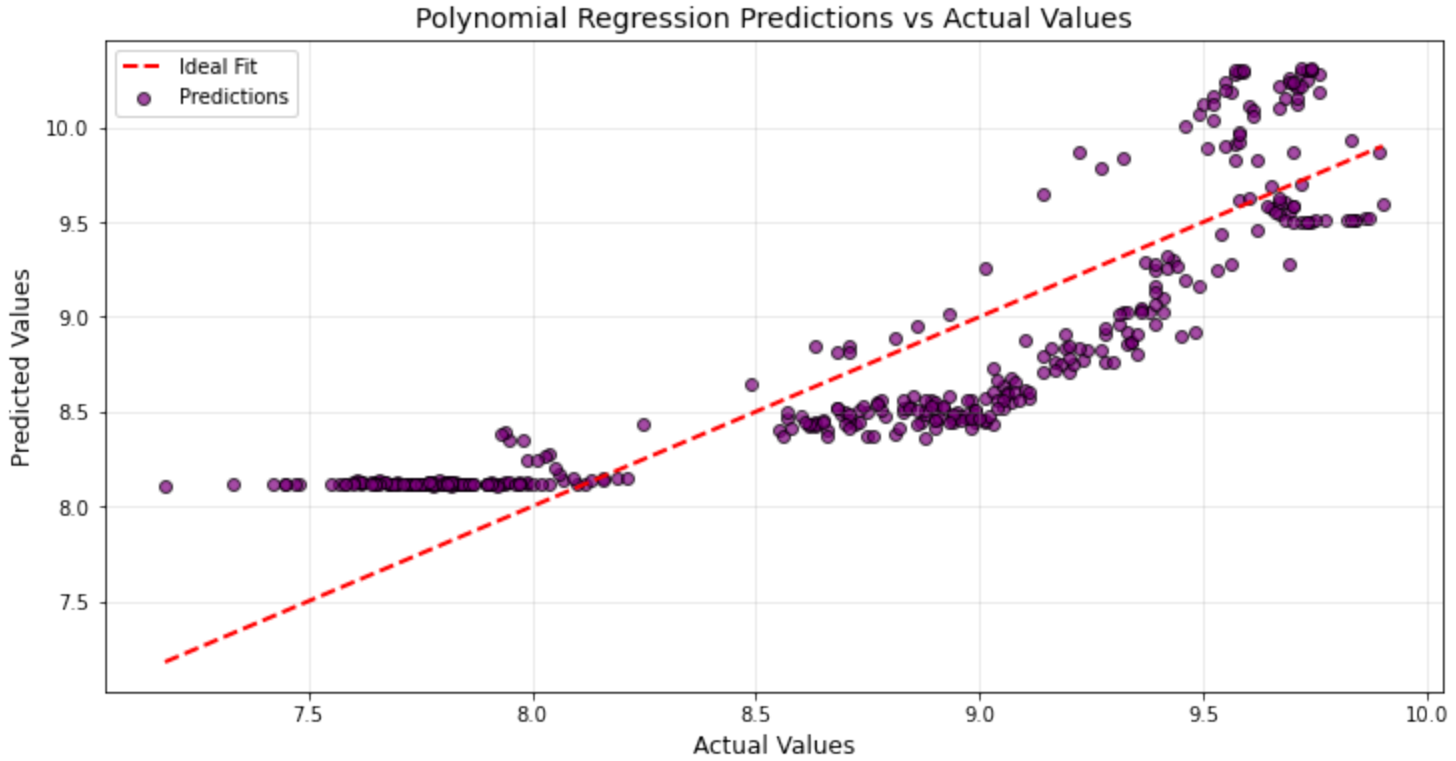
STEP 9: Evaluate the polynomial regression model

- Make predictions with your model and then check the performance of the model.
- Check your model performance by looking at the MSE and the R^2 score.
- Create a scatter plot of the Y polynomial predictions and your Y test data.

In [ ]:
```
# Make predictions using the polynomial model
Y_poly_pred = poly_model.predict(X_poly_test)

# Calculate evaluation metrics using scikit-learn's mean_squared_error and r2_score
poly_mse = np.sqrt(mean_squared_error(Y_test, Y_poly_pred))
poly_r2 = r2_score(Y_test, Y_poly_pred)

print(f"Polynomial Regression Mean Squared Error: {poly_mse}")
print(f"Polynomial Regression R² Score: {poly_r2}")

# Plot predictions vs actual
# Plot predictions vs actual values
plt.figure(figsize=(12, 6))
plt.scatter(Y_test, Y_poly_pred, alpha = 0.7, color = 'purple', edgecolor = 'k')
```

```
plt.plot([Y_test.min(), Y_test.max()], [Y_test.min(), Y_test.max()], 'r--', lw = 2, label = 'Ideal Fit')
plt.title('Polynomial Regression Predictions vs Actual Values', fontsize = 14)
plt.xlabel('Actual Values', fontsize = 12)
plt.ylabel('Predicted Values', fontsize = 12)
plt.legend(['Ideal Fit', 'Predictions'])
plt.grid(alpha = 0.3)
plt.show()
```

```
Polynomial Regression Mean Squared Error: 0.38756190338796986
Polynomial Regression R² Score: 0.729308225707336
```



Step 10: Compare your polynomial and linear regression results

What differences did you notice between you polynomial regression and linear regression results? Which model performed better? Why do you think this is? Write your answer in the markdown cell below.

Answer: The polynomial regression performed better than the linear regression, with a higher R^2 score (0.73 vs. 0.49) and a lower RMSE (0.39 vs. 0.53). This shows that the relationship between the features and the target variable is non-linear, and the polynomial regression was better able to capture this complexity compared to the linear model.