Aalto University School of Science

CS-C3240 - Machine Learning D

Machine Learning Project

Predicting cruise ship deadweight using regression models

Contents

1 Introduction

Predicting cruise ship deadweight is a vital undertaking in ship design, and it carries significant implications for the economic viability of cruise ship ventures. The deadweight of a cruise ship affects numerous aspects of its operation. Utilizing regressive predictive models to estimate the deadweight allows cruise companies to strike a balance between safety and performance standards and economic viability.

In Section 2, we will define the machine learning problem we aim to solve. Section 3 will delve into the methodology, including data preparation, the selection of regression models, the introduction of our loss function, model validation, and a brief discussion on feature selection. Afterward, in Section 4, we will analyze the results and present the most promising regression method for our application domain. Finally, Section 5 is the conclusion.

2 Problem Formulation

To solve this problem, we will solely rely on the ship's physical dimensions, as they are the most relevant factors for the deadweight. Ships with larger dimensions invariably tend to weigh more, and there exists a direct correlation between ship size and deadweight. The machine learning task in this project is supervised learning.

The dataset used consists of data points, each representing a different ship along with their respective dimensions and deadweight. The data types in this dataset include categorical (Name) and continuous (L=Length, B=Breadth, T=Draft, DW=Deadweight) variables.

3 Methodology

3.1 Data preparation

The dataset has a total of 87 data points including the name of the ship, the ship's dimensions and corresponding deadweight. The dataset was collected from Prof. Jani Romanoff's Passenger Ships D -course. [1]

Our initial steps involve printing the data to ensure it's properly formatted as a pandas DataFrame. Subsequently, we perform data preprocessing and cleaning. Specifically, we remove the "Name" column and eliminate any non-alphanumeric characters, allowing us to prepare the data for further analysis. Afterwards, the data is plotted in 3D format. Lastly, we use the preprocessing standard scaler for data scaling. This allows us to obtain the mean and standard deviation values for each feature and the label.

3.2 Feature selection

To enhance the predictive accuracy of the model, it's essential to consider multiple features than, for example, only the vessel's length. The deadweight of a ship is linked to various dimensions, including the length, breadth, and draft of a ship, making them all crucial factors. The 3D plot of the data included in the code (Appendix) shows the complex relationship between all these dimensions and the deadweight of the ship. No single dimension exhibits a significantly stronger correlation with deadweight compared to the others. This is attributed to the intercorrelation among the dimensions; longer ships tend to have greater breadth and consequently deeper drafts.

3.3 Model selection

Linear regression is a widely-used machine learning method that models the relationship between one or more input variables (features) and a continuous output variable. It aims to find the best-fit straight line (or hyperplane in multiple dimensions) that minimizes the difference between predicted and observed values. Linear regression can be enhanced through regularization, a technique that transforms it into Lasso Regression. This regularization technique helps prevent overfitting and can lead to more interpretable models. [2]

Lasso regression is well-suited for our purpose because it not only models the relationships between input features and the continuous target variable but also effectively handles feature selection by driving some feature coefficients to exactly zero. This feature selection property makes Lasso regression valuable for tasks where only a subset of features is relevant. In our case, Lasso regression will help us analyze how various ship dimensions relate to its deadweight while selecting the most informative features. [2]

On top of Lasso regression, we will also be using polynomial regressive models, due to their ability to handle non-linear relationships between the features and the target variable. It fits a polynomial equation to the data, making it suitable for capturing complex relationships where linear regularized models fall short. This choice is based on our belief that the connection between ship dimensions and deadweight may not be purely linear; there could be non-linear patterns that polynomial regression can effectively capture. [3]

3.4 Loss function

In our regression problem to predict continuous values like ship weight, we use Root-Mean Squared Error (RMSE) as our primary loss function. RMSE gauges average prediction error by calculating the square root of the average squared difference between actual and predicted values. We also employ Mean Absolute Error (MAE) to assess error magnitudes, regardless of size. The combined use of RMSE and MAE is motivated by their ability to measure the average magnitude of prediction errors, which is relevant in the context of continuous and concrete numerical values like ship weights, providing a comprehensive assessment of predictive performance. They also provide a balanced evaluation of prediction errors in our case, where both large and small errors can have practical implications. Additionally, we utilize R-squared (R^2) to quantify the model's explanatory power. For lasso regression, we also consider the L1-norm as a loss function. [2, 3]

3.5 Model validation

The dataset was divided with a single split into training and validation set. The training set is used to train the regression model, while the validation set is used to evaluate the model's performance. The split ratio is set to 80% for training and 20% for validation. With a relatively small dataset, allocating the majority of the data for training provides the model with a decent amount of information to learn from. This allows the model to capture the underlying patterns and relationships between ship dimensions and deadweight effectively. This design choice allows for the assessment of how well the model generalizes to new unseen data by comparing its predictions on the test set to the actual target values. [3]

4 Results

After testing the models, we can present the results. The test dataset consists of 10 data points and is also gathered from Prof. Jani Romanoff's course. The structure of the test set is similar to the training and validation sets. Tab. 1 below presents the key performance metrics for the regression models. [1]

Table 1: Model Performance Comparison

Model	Degree	Dataset	MAE	RMSE	R^2
Lasso Regression	-	Training	1604.38	2058.96	0.76
		Validation	1125.67	1366.67	0.86
		Test	1312.45	1606.83	0.75
Polynomial Regression	2	Training	1400.03	1759.16	0.83
		Validation	952.09	1175.58	0.90
		Test	1112.92	1300.54	0.84
Polynomial Regression	3	Training	1238.43	1552.76	0.86
		Validation	1038.47	1327.62	0.87
		Test	1147.73	1412.66	0.81
Polynomial Regression	4	Training	1498.16	1799.53	0.82
		Validation	1765.13	2141.04	0.66
		Test	1373.05	1729.71	0.71

When evaluating the various regression methods, distinct differences in performance emerged. Notably, the 2nd-degree polynomial regression model stood out as the most promising choice due to its ability to find a balance between model complexity and predictive accuracy. During testing, the 2nd-degree polynomial regression model demonstrated great results. It achieved a Root-Mean Squared Error (RMSE) of 1300.54 and a Mean Absolute Error (MAE) of 1112.92, providing valuable insights into the model's predictive capability. Beyond these metrics, the model's strength becomes even more apparent as it successfully captured 84% of the variance in ship deadweights, as indicated by the R-squared value of 0.84. This highlights the model's practical utility and its ability to explain a significant portion of the variability in cruise ship deadweight.

It's essential to recognize that not all models performed with equal success. Lasso regression and polynomial regression with a 4th-degree polynomial exhibited less favorable outcomes when compared to their 2nd and

3rd-degree counterparts during our testing phase. Lasso regression, although conceptually solid with its regularization approach, faced challenges in capturing the underlying data patterns. This can be seen in a higher RMSE, signifying an increased rate of prediction errors, and a lower R-squared value, suggesting a reduced explanatory power in comparison.

On the other hand, the 4th-degree polynomial regression model managed to achieve lower prediction errors, which might initially seem promising. However, it's imperative to consider that this apparent success comes at a significant cost — overfitting. Overfitting implies that the model has become overly complex and is fitting not only the actual underlying patterns but also the noise present in the data. Because of this, it struggles to generalize to new, unseen data, which is a crucial requirement for reliable predictions in real-world scenarios.

5 Conclusion

In summary, this machine learning project aimed to predict cruise ship deadweight using regression models based on ship dimensions. We explored linear regularized (Lasso) regression and polynomial regression methods, with different degrees, to find the most suitable model. After careful evaluation, we found that the 2nd-degree polynomial regression model performed the best, striking a balance between complexity and predictive accuracy.

When assessing the model's performance, we observed that it achieved a reasonable Root-Mean Squared Error (RMSE) of 1300.54, with a Mean Absolute Error (MAE) of 1112.92 on the test dataset. Additionally, the model explained 84% of the variance in ship deadweights, indicating its practical utility.

However, there is room for improvement. One noticeable aspect is that the 4th-degree polynomial regression model tended to overfit the data, leading to reduced generalization. To address this issue, exploring techniques to mitigate overfitting, such as different regularization methods, could be a valuable future direction. Also, obtaining a more extensive and diverse dataset could further enhance the model's performance. Adding features related to ship construction materials and their properties would make the deadweight estimation more precise.

In conclusion, the 2nd-degree polynomial regression model is a promising method to predict cruise ship deadweight accurately. However, there exist opportunities for refinement in various aspects, including model selection, feature engineering, and data collection, which can collectively contribute to the advancement of ship deadweight prediction through machine learning.

References

[1] Jani Romanoff. *Cruise Ship Datasheet*. Passenger ships D, 2022.

[2] Luca Massaron, Alberto Boschetti. *Regression Analysis with Python*. Packt Publishing, 2016.

[3] Deepti Chopra. *Building Machine Learning Systems Using Python: Practice to Train Predictive Models and Analyze Machine Learning Results with Real Use-Cases*. BPB Publications, 1st edition, 2021.

Appendix: Source code

September 20, 2023

[1]:
```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import PolynomialFeatures
from sklearn.metrics import mean_squared_error, r2_score
from sklearn.linear_model import Lasso
from sklearn.model_selection import GridSearchCV
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
from sklearn.metrics import mean_squared_error, mean_absolute_error,␣
↪r2_score
```

[2]:
```
#load and print the dataset
data = pd.read_csv("Cruise-Ship-Weight-Prediction-Model/data/
↪cruiseships.csv", sep=";")
data = data.fillna(0)
data = data[data["Name"] != 0]
data = data[data["L"] != 0]
data = data[data["B"] != 0]
data = data[data["T"] != 0]
data = data[data["DW"] != 0]
print(data)
```

```
                    Name    L   B  T       DW
0   Antherm of the seas  310  41  9  12000.0
1         Azamara Quest  253  26  6   2700.0
2     Carnival Conquest  348  36  8  10000.0
3      Carnival Destiny  181  35  8  11142.0
4      Carnival Ecstasy  290  31  8   7200.0
..                  ...  ...  .. ..      ...
82             Voyager  330  29  7   5400.0
83  Voyager of the Seas  181  48  8  11132.0
84             Whisper  251  25  6   2980.0
85                Wind  136  12  7   1790.0
86               Wolin  330  24  6   5143.0
```

```
[86 rows x 5 columns]
```

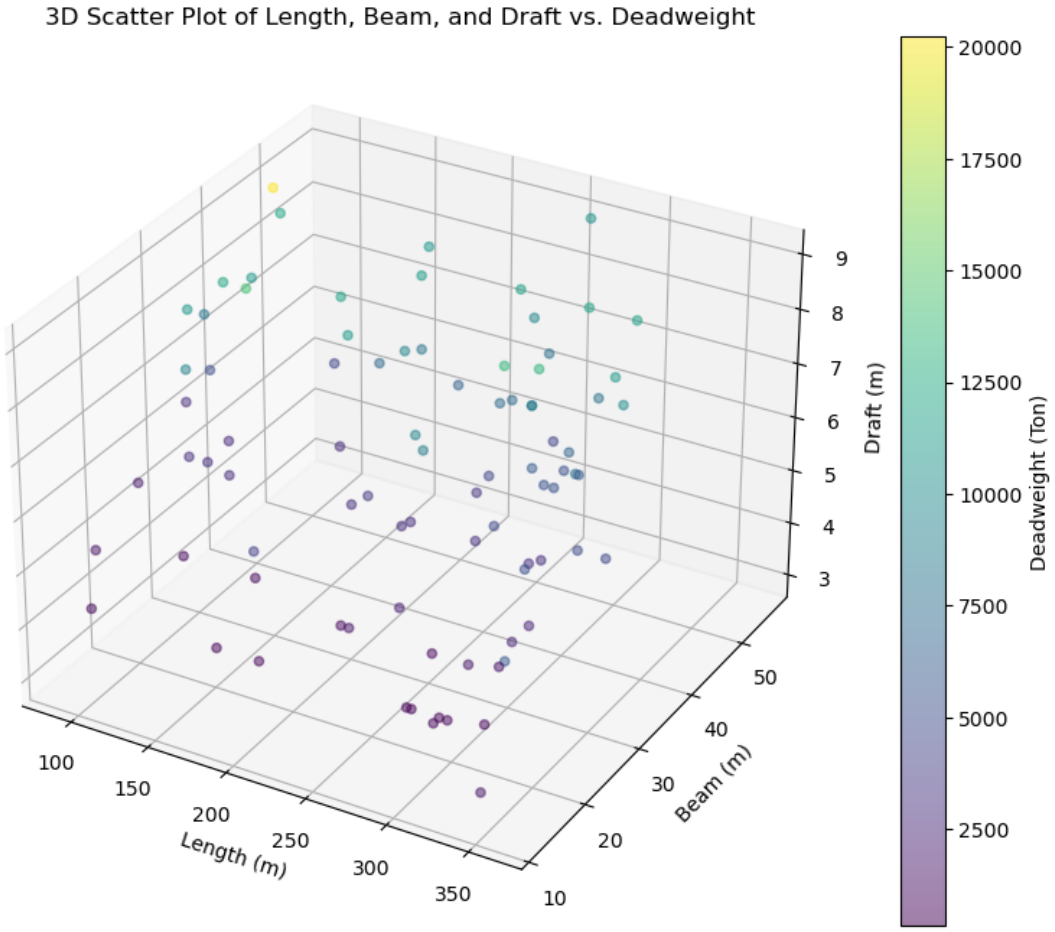
[3]:
```
#data preprocessing and cleaning
data = data.drop(columns=['Name'])
data.columns = data.columns.str.replace('[^\w\s]', '')

#plot the data using Matplotlib in 3D
fig = plt.figure(figsize=(10, 8))
ax = fig.add_subplot(111, projection='3d')

#scatter plot with "L," "B," and "T" as the 3D dimensions and "DW" as␣
↪the color scale
scatter = ax.scatter(data['L'], data['B'], data['T'], c=data['DW'],␣
↪cmap='viridis', marker='o', alpha=0.5)
plt.colorbar(scatter, label='Deadweight (Ton)')

#set labels for the axes and a title
ax.set_xlabel('Length (m)')
ax.set_ylabel('Beam (m)')
ax.set_zlabel('Draft (m)')
plt.title('3D Scatter Plot of Length, Beam, and Draft vs. Deadweight')

#show the plot
plt.grid(True)
plt.show()
```



[4]:
```
#data transformation - scaling
scaler = StandardScaler()

#fit the scaler to your data to compute the mean and standard
↪deviation
scaler.fit(data)

#transform your data using the scaler to scale it
scaled_data = scaler.transform(data)

#calculate the mean and standard deviation for each feature
mean_values = np.mean(scaled_data, axis=0)
std_deviation = np.std(scaled_data, axis=0)

#print the mean and standard deviation for each feature
for i, column in enumerate(data.columns):
    print(f"Feature: {column}")
```

```
    print(f"Mean: {mean_values[i]}")
    print(f"Standard Deviation: {std_deviation[i]}\n")
```

```
Feature: L
Mean: 1.2264091551091845e-16
Standard Deviation: 1.0

Feature: B
Mean: 2.1946269091427512e-17
Standard Deviation: 1.0

Feature: T
Mean: -2.2075364791965322e-16
Standard Deviation: 1.0

Feature: DW
Mean: -2.872379336966248e-17
Standard Deviation: 1.0
```

[5]:
```
#split the data into training and validation sets
X = data[['L', 'B', 'T']]
y = data['DW']
X_train, X_val, y_train, y_val = train_test_split(X, y, test_size=0.
↪2, random_state=42)
```

[6]:
```
#define a range of alpha values to search over
alphas = [0.01, 0.1, 0.5, 1.0, 2.0, 4.0, 6.0, 8.0, 10.0]

#create a lasso regression model
lasso_model = Lasso()

#perform grid search with cross-validation
grid_search = GridSearchCV(lasso_model, param_grid={'alpha': alphas},␣
↪cv=5, scoring='neg_mean_squared_error')
grid_search.fit(X_train, y_train)

#get the best alpha value from the grid search
best_alpha = grid_search.best_params_['alpha']
print(f"Optimal Alpha: {best_alpha}")
```

```
Optimal Alpha: 10.0
```

[7]:
```
#create and train the Lasso regression model with regularization
lasso_model = Lasso(alpha=best_alpha)
lasso_model.fit(X_train, y_train)

#calculate the L1-norm of parameters (Lasso) for the model
```

```
lasso_coefficient_norm = np.sum(np.abs(lasso_model.coef_))
print(f"L1-norm of parameters (Lasso): {lasso_coefficient_norm:.2f}")
```

```
L1-norm of parameters (Lasso): 1266.31
```

[8]:
```
#make predictions on the training set
y_train_pred = lasso_model.predict(X_train)

#make predictions on the validation set
y_val_pred = lasso_model.predict(X_val)
```

[9]:
```
#calculate training error metrics
train_mae = mean_absolute_error(y_train, y_train_pred)
train_mse = mean_squared_error(y_train, y_train_pred)
train_rmse = np.sqrt(train_mse)
train_r2 = r2_score(y_train, y_train_pred)

#calculate validation error metrics
val_mae = mean_absolute_error(y_val, y_val_pred)
val_mse = mean_squared_error(y_val, y_val_pred)
val_rmse = np.sqrt(val_mse)
val_r2 = r2_score(y_val, y_val_pred)

#print the evaluation metrics
print("\nLasso regression training error metrics:")
print(f"Mean Absolute Error (MAE): {train_mae:.2f}")
print(f"Root Mean Squared Error (RMSE): {train_rmse:.2f}")
print(f"R-squared (R2): {train_r2:.2f}")

print("\nLasso regression validation error metrics:")
print(f"Mean Absolute Error (MAE): {val_mae:.2f}")
print(f"Root Mean Squared Error (RMSE): {val_rmse:.2f}")
print(f"R-squared (R2): {val_r2:.2f}")
```

```
Lasso regression training error metrics:
Mean Absolute Error (MAE): 1604.38
Root Mean Squared Error (RMSE): 2058.96
R-squared (R2): 0.76

Lasso regression validation error metrics:
Mean Absolute Error (MAE): 1125.67
Root Mean Squared Error (RMSE): 1366.67
R-squared (R2): 0.86
```

[10]:
```
#now, let's perform polynomial regression with degrees 2, 3, and 4
degrees = [2, 3, 4]
polyregr_models = {}  #create an empty dictionary to store the models
```

[11]:
```
for degree in degrees:
    #create polynomial features
    poly = PolynomialFeatures(degree=degree)
    X_train_poly = poly.fit_transform(X_train)
    X_val_poly = poly.transform(X_val)

    #create and train a linear regression model on the polynomial␣
↪features
    polyregr_model = LinearRegression()
    polyregr_model.fit(X_train_poly, y_train)

    #store the model in the dictionary
    polyregr_models[f'polyregr_{degree}_model'] = polyregr_model

    #make predictions on the training set
    y_train_pred_poly = polyregr_model.predict(X_train_poly)

    #make predictions on the validation set
    y_val_pred_poly = polyregr_model.predict(X_val_poly)

    #calculate training error metrics for polynomial regression
    train_mae_poly = mean_absolute_error(y_train, y_train_pred_poly)
    train_mse_poly = mean_squared_error(y_train, y_train_pred_poly)
    train_rmse_poly = np.sqrt(train_mse_poly)
    train_r2_poly = r2_score(y_train, y_train_pred_poly)

    #calculate validation error metrics for polynomial regression
    val_mae_poly = mean_absolute_error(y_val, y_val_pred_poly)
    val_mse_poly = mean_squared_error(y_val, y_val_pred_poly)
    val_rmse_poly = np.sqrt(val_mse_poly)
    val_r2_poly = r2_score(y_val, y_val_pred_poly)

    #print the evaluation metrics for polynomial regression
    print(f"Polynomial regression (degree {degree}) training error␣
↪metrics:")
    print(f"Mean Absolute Error (MAE): {train_mae_poly:.2f}")
    print(f"Root Mean Squared Error (RMSE): {train_rmse_poly:.2f}")
    print(f"R-squared (R2): {train_r2_poly:.2f}")

    print(f"\nPolynomial regression (degree {degree}) validation␣
↪error metrics:")
    print(f"Mean Absolute Error (MAE): {val_mae_poly:.2f}")
    print(f"Root Mean Squared Error (RMSE): {val_rmse_poly:.2f}")
    print(f"R-squared (R2): {val_r2_poly:.2f}\n")
```

```
Polynomial regression (degree 2) training error metrics:
Mean Absolute Error (MAE): 1400.03
Root Mean Squared Error (RMSE): 1759.16
```

```
R-squared (R2): 0.83

Polynomial regression (degree 2) validation error metrics:
Mean Absolute Error (MAE): 952.09
Root Mean Squared Error (RMSE): 1175.58
R-squared (R2): 0.90

Polynomial regression (degree 3) training error metrics:
Mean Absolute Error (MAE): 1238.43
Root Mean Squared Error (RMSE): 1552.76
R-squared (R2): 0.86

Polynomial regression (degree 3) validation error metrics:
Mean Absolute Error (MAE): 1038.47
Root Mean Squared Error (RMSE): 1327.62
R-squared (R2): 0.87

Polynomial regression (degree 4) training error metrics:
Mean Absolute Error (MAE): 1498.16
Root Mean Squared Error (RMSE): 1799.53
R-squared (R2): 0.82

Polynomial regression (degree 4) validation error metrics:
Mean Absolute Error (MAE): 1765.13
Root Mean Squared Error (RMSE): 2141.04
R-squared (R2): 0.66
```

[12]:
```
#load and preprocess the test data
test_data = pd.read_csv("Cruise-Ship-Weight-Prediction-Model/data/
 ↪cruiseships_test.csv", sep=";")
test_data = test_data.fillna(0)
test_data = test_data[test_data["L"] != 0]
test_data = test_data[test_data["B"] != 0]
test_data = test_data[test_data["T"] != 0]
test_data = test_data[test_data["DW"] != 0]
print(test_data)
```

```
                  Name    L   B  T     DW
0   Caribbean Princess  362  36  8  11000
1      Carnival Legend  290  32  8   7089
2       Coral Princess  315  32  8   8015
3               Galaxy  340  29  6   4850
4        Genting Dream  340  40  8  10000
5   MV Seabourn Legend  171  20  5    790
6  Quantum of the Seas  296  41  9  10500
7              Regatta  123  26  6   2700
8         Silja Europa  326  32  7   4650
```

```
9   Vision of the Seas  198  32  8   6300
```

[13]:
```
#data preprocessing and cleaning
test_data = test_data.drop(columns=['Name'])
test_data.columns = test_data.columns.str.replace('[^\w\s]', '')
```

[14]:
```
#fit the scaler to your data to compute the mean and standard␣
↪deviation
scaler.fit(test_data)

#transform your data using the scaler to scale it
scaled_test_data = scaler.transform(test_data)

#calculate the mean and standard deviation for each feature
mean_values_test = np.mean(scaled_test_data, axis=0)
std_deviation_test = np.std(scaled_test_data, axis=0)

#print the mean and standard deviation for each feature
for i, column in enumerate(data.columns):
    print(f"Feature: {column}")
    print(f"Mean: {mean_values_test[i]}")
    print(f"Standard Deviation: {std_deviation_test[i]}\n")
```

```
Feature: L
Mean: -2.6645352591003756e-16
Standard Deviation: 1.0000000000000002

Feature: B
Mean: -2.2204460492503132e-17
Standard Deviation: 1.0

Feature: T
Mean: 1.3322676295501878e-16
Standard Deviation: 1.0

Feature: DW
Mean: 1.1934897514720432e-16
Standard Deviation: 1.0
```

[15]:
```
X_test = test_data[['L', 'B', 'T']]
y_test = test_data['DW']
#make predictions on test data
y_test_pred_lasso = lasso_model.predict(X_test)
```

[16]:
```
polyregr_test_predictions = {}

for degree in degrees:
```

```
    #transform test data for the specific polynomial degree
    test_poly = PolynomialFeatures(degree=degree)
    X_test_poly = test_poly.fit_transform(X_test)

    #make predictions on the test set using the polynomial regression␣
↪model
    y_test_pred_poly = polyregr_models[f'polyregr_{degree}_model'].
↪predict(X_test_poly)

    polyregr_test_predictions[f'polyregr_{degree}_model'] =␣
↪y_test_pred_poly
```

[17]:
```
#evaluate Lasso regression on the test data
test_mae_lasso = mean_absolute_error(y_test, y_test_pred_lasso)
test_rmse_lasso = np.sqrt(mean_squared_error(y_test,␣
↪y_test_pred_lasso))
test_r2_lasso = r2_score(y_test, y_test_pred_lasso)

print("Lasso regression test error metrics:")
print(f"Mean Absolute Error (MAE): {test_mae_lasso:.2f}")
print(f"Root Mean Squared Error (RMSE): {test_rmse_lasso:.2f}")
print(f"R-squared (R2): {test_r2_lasso:.2f}")

#evaluate Polynomial regression models on the test data
for degree in degrees:
    y_test_pred_poly =␣
↪polyregr_test_predictions[f'polyregr_{degree}_model']
    test_mae_poly = mean_absolute_error(y_test, y_test_pred_poly)
    test_rmse_poly = np.sqrt(mean_squared_error(y_test,␣
↪y_test_pred_poly))
    test_r2_poly = r2_score(y_test, y_test_pred_poly)

    print(f"\nPolynomial regression (degree {degree}) test error␣
↪metrics:")
    print(f"Mean Absolute Error (MAE): {test_mae_poly:.2f}")
    print(f"Root Mean Squared Error (RMSE): {test_rmse_poly:.2f}")
    print(f"R-squared (R2): {test_r2_poly:.2f}")
```

```
Lasso regression test error metrics:
Mean Absolute Error (MAE): 1312.45
Root Mean Squared Error (RMSE): 1606.83
R-squared (R2): 0.75

Polynomial regression (degree 2) test error metrics:
Mean Absolute Error (MAE): 1112.92
Root Mean Squared Error (RMSE): 1300.54
R-squared (R2): 0.84
```

```
Polynomial regression (degree 3) test error metrics:
Mean Absolute Error (MAE): 1147.73
Root Mean Squared Error (RMSE): 1412.66
R-squared (R2): 0.81

Polynomial regression (degree 4) test error metrics:
Mean Absolute Error (MAE): 1373.05
Root Mean Squared Error (RMSE): 1729.71
R-squared (R2): 0.71
```