PREDICTING CONCRETE COMPRESSIVE STRENGTH USING PLS

Presentation by Dharmik Joshi

SEP 767: Multivariate Statistical Methods for Big Data Analysis and Process Improvement

McMaster University

INTRODUCTION

In this project, we are building a Partial Least Squares (PLS) regression model to predict the compressive strength of concrete based on its composition. We are using the "Concrete Compressive Strength" dataset from the UCI Machine Learning Repository to train and test our model.

To start, we perform exploratory data analysis (EDA) on the dataset to understand the distribution of the features and their correlation with the target variable. Next, we preprocess the data by separating the features and the target variable, and standardizing the features.

We then train the PLS model using a portion of the data, and use it to predict the target variable on the remaining portion.

We evaluate the performance of the model using the R-squared metric, mean squared error (MSE), and square prediction error (SPE). Finally, we visualize the results using scatter plots, trend lines, and slopes.

BACKGROUND

The compressive strength of concrete is an important factor in determining the durability and reliability of concrete structures. In construction, it is critical to accurately predict the strength of concrete before it is used to build structures to ensure their safety and longevity. Traditional methods for predicting compressive strength involve time-consuming and expensive lab tests, which are not always feasible for large-scale construction projects.

In this project, we leverage the power of PLS regression to develop a predictive model for the compressive strength of concrete, and we evaluate its performance using a range of metrics and statistical analyses. By doing so, we aim to provide a valuable tool for engineers and construction professionals to accurately predict the strength of concrete and ensure the safety and longevity of their structures.

DATASET DESCRIPTION

The "Concrete Compressive Strength" dataset contains the compressive strength of 1,030 samples of concrete, along with the amounts of various components used to make the concrete. Specifically, the dataset includes the following features:

- Cement: the amount of cement (kg/m3) used in the concrete mixture
- Blast Furnace Slag: the amount of blast furnace slag (kg/m3) used in the concrete mixture
- Fly Ash: the amount of fly ash (kg/m3) used in the concrete mixture
- Water: the amount of water (kg/m3) used in the concrete mixture
- Superplasticizer: the amount of superplasticizer (kg/m3) used in the concrete mixture
- Coarse Aggregate: the amount of coarse aggregate (kg/m3) used in the concrete mixture
- Fine Aggregate: the amount of fine aggregate (kg/m3) used in the concrete mixture
- Age: the age of the concrete sample (days)

The target variable is the compressive strength of the concrete (MPa). The dataset is sourced from experiments conducted by the Department of Civil Engineering at Istanbul Kultur University.

METHODOLOGY

DATA COLLECTION

The "Concrete Compressive Strength" dataset was obtained from the UCI Machine Learning Repository. The dataset was preprocessed to clean, normalize, and transform the data, and to split the dataset into training and test sets.

MODEL DEVELOPMENT

The PLS regression algorithm was used to develop a predictive model for the compressive strength of concrete, based on the other features in the dataset. The key steps involved in the PLS algorithm, including the calculation of the scores and weights for X and Y matrices, were implemented using Python programming language.

MODEL EVALUATION

The performance of the PLS regression model was evaluated using various metrics, including the R-squared value, mean squared error (MSE), and square prediction error (SPE).

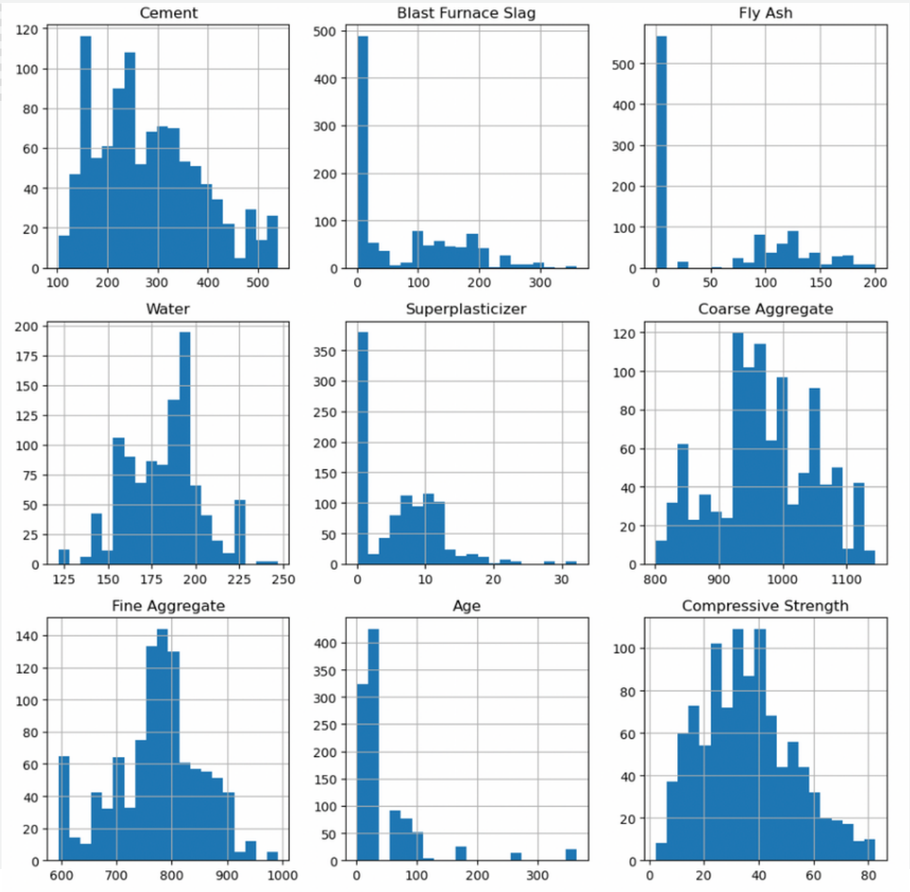
RESULTS

The results of the PLS regression model showed that it was able to accurately predict the compressive strength of concrete, with an R-squared value of 0.612
and Mean Squared Error: 0.352

SEP 767: Multivariate Statistical Methods for Big Data Analysis and Process Improvement

McMaster University

DATA COLLECTION



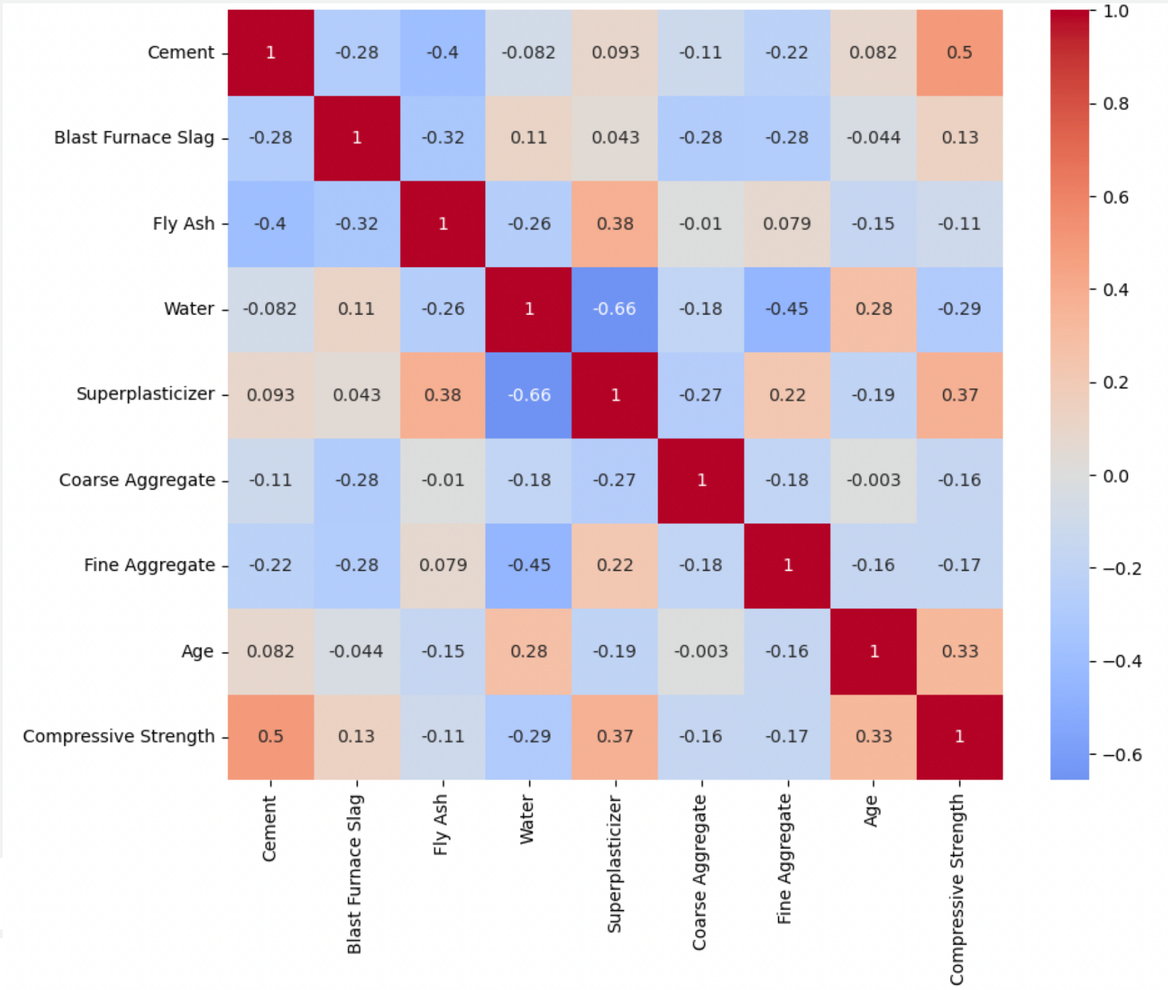
In [1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

# Load dataset
url = "https://archive.ics.uci.edu/ml/machine-learning-databases/concrete/compressiv
dataset = pd.read_excel(url)


# Rename the columns for better readability
column_names = ['Cement', 'Blast Furnace Slag', 'Fly Ash', 'Water', 'Superplasticize
dataset.columns = column_names
```

In [2]:
```
# Histograms
dataset.hist(bins=20, figsize=(10, 10))
plt.tight_layout()
plt.show()


# Correlation heatmap
plt.figure(figsize=(10, 8))
sns.heatmap(dataset.corr(), annot=True, cmap='coolwarm', center=0)
plt.show()
```

SEP 767: Multivariate Statistical Methods for Big Data Analysis and Process Improvement

McMaster University

DATA VISUALIZATION



SEP 767: Multivariate Statistical Methods for Big Data Analysis and Process Improvement

McMaster University

DATA PREPROCESSING

The code is separating the input features and target variable into two separate arrays, which is necessary for further data preprocessing and model training. The dataset was preprocessed to clean, normalize, and transform the data, and to split the dataset into training and test sets

In [6]:
```
from sklearn.model_selection import train_test_split
# Split data into training and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=4

# Standardize the data
X_mean = X_train.mean(axis=0)
X_std = X_train.std(axis=0)
y_mean = y_train.mean()
y_std = y_train.std()

X_train = (X_train - X_mean) / X_std
y_train = (y_train - y_mean) / y_std
X_test = (X_test - X_mean) / X_std
y_test = (y_test - y_mean) / y_std
```

MODEL DEVELOPMENT

```python
def PLS(X, y, n_components):
    T = np.zeros((X.shape[0], n_components))
    P = np.zeros((X.shape[1], n_components))
    W = np.zeros((X.shape[1], n_components))
    C = np.zeros((y.shape[1], n_components))
    U = np.zeros((y.shape[0], n_components))

    for i in range(n_components):
        # Calculate weight vector for X
        w = X.T @ y / np.linalg.norm(X.T @ y)
        # Calculate score vector for X
        t = X @ w
        # Calculate loading vector for Y
        c = y.T @ t / (t.T @ t)
        # Calculate score vector for Y
        u = y @ c / (c.T @ c)
        # Calculate loading vector for X
        p = X.T @ t / (t.T @ t)

        # Deflate X and y matrices
        X = X - np.outer(t, p.T)
        y = y - np.outer(t, c.T)

        # Store T, P, W, C, and U matrices
        T[:, i] = t.flatten()
        P[:, i] = p.flatten()
        W[:, i] = w.flatten()
        C[:, i] = c.flatten()
        U[:, i] = u.flatten()

    return T, P, W, C, U
```

The code calculates the weight vector for X (w), score vector for X (t), loading vector for Y (c), and score vector for Y (u) iteratively for a specified number of components.

- The weight vector for X is determined for each iteration by dividing the dot product of the transpose of X and y by the norm of the dot product of the transpose of X and y. The dot product of X with the weight vector for X yields the score vector for X.
- The loading vector for Y is calculated by dividing the dot product of the transpose of the score vector for X by the score vector for X to get the loading vector for Y. The score vector for Y is calculated as the dot product of y and the loading vector for Y, divided by the dot product of the transpose of the loading vector for Y and the loading vector for Y.
- The loading vector for X is calculated as the dot product of the transpose of X and the score vector for X, divided by the dot product of the transpose of the score vector for X and the score vector for X.
- The code then deflates the X and y matrices and stores the calculated weight vector for X, score vector for X, loading vector for Y, score vector for Y, and the deflated X and y matrices in T, P, W, C, and U matrices, respectively. Finally, it returns these matrices.

The np.linalg function used in the code is a NumPy linear algebra module, used to compute matrix operations such as matrix multiplication, inversion, and norm calculation. The np.outer function is used to calculate the outer product of two arrays, while the flatten function is used to convert an array of arrays into a single array.

MODEL EVALUATION

- The code calls the **PLS()** function, which takes the input predictor matrix **X_train**, response matrix **y_train**, and the desired number of PLS components (**n_components**) as arguments. It returns the score matrices **T** and **U**, loading matrices **P** and **C**, and the weight matrix **W**

- The code computes the regression coefficients matrix B using the weight matrix W, loading matrix P, and loading matrix C

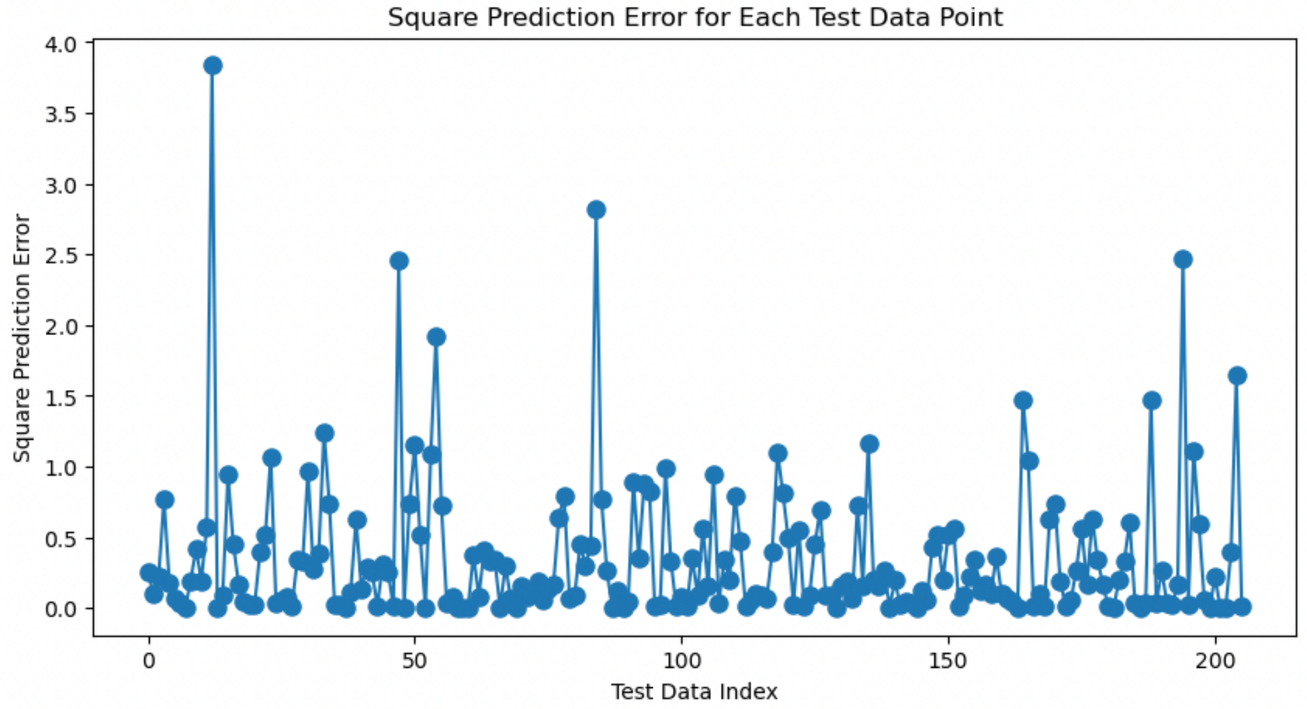
- The code calculates the R-squared value for the model using the test data y_test and the predicted response y_pred

- The code computes the mean squared error and the square prediction error using the mean_squared_error() and square_prediction_error() functions defined earlier

```
# Compute PLS components
n_components = 3
T, P, W, C, U = PLS(X_train, y_train, n_components)

# Calculate the regression coefficients
B = W @ np.linalg.inv(P.T @ W) @ C.T

# Predict compressive strength using the PLS model

y_pred = X_test @ B

# Compute the model's R-squared
R_squared = 1 - np.sum((y_test - y_pred) ** 2) / np.sum((y_test - y_test.mean()) ** 2)
print(f"R-squared: {R_squared:.3f}")

def mean_squared_error(y_true, y_pred):
    return np.mean((y_true - y_pred) ** 2)

def square_prediction_error(y_true, y_pred):
    return (y_true - y_pred) ** 2


MSE = mean_squared_error(y_test, y_pred)
SPE = square_prediction_error(y_test, y_pred)
```

MODEL EVALUATION

```
R-squared: 0.612
Mean Squared Error: 0.352
```



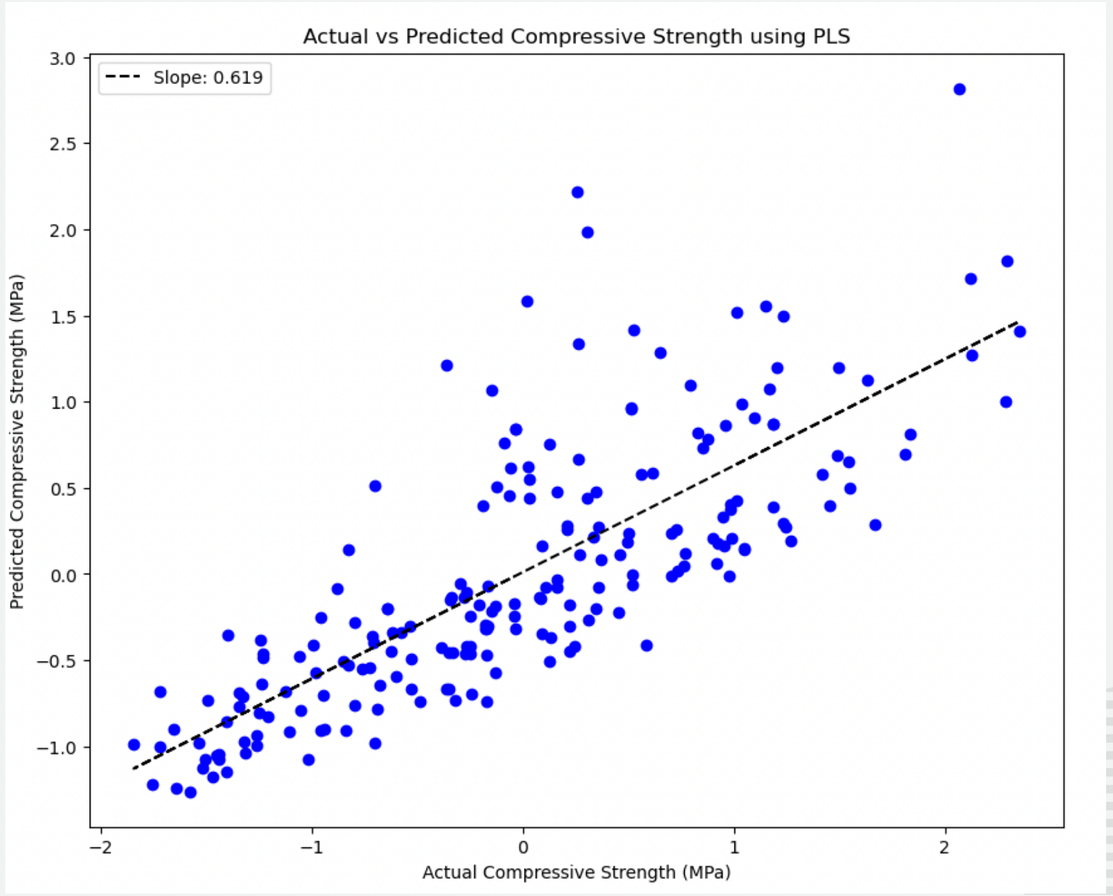
The plot shows the square prediction error for each test data point. It indicates how far the predicted values are from the actual values, with larger errors indicating a larger discrepancy between the predicted and actual values.

```
print(f"Mean Squared Error: {MSE:.3f}")
plt.figure(figsize=(10, 5))
plt.plot(SPE, 'o-', markersize=8)
plt.xlabel("Test Data Index")
plt.ylabel("Square Prediction Error")
plt.title("Square Prediction Error for Each Test Data Point")
plt.show()
```

MODEL EVALUATION

The plot shows the relationship between the actual and predicted compressive strength for the test data. The closer the points are to the line y=x, the better the model is performing in predicting the actual values.

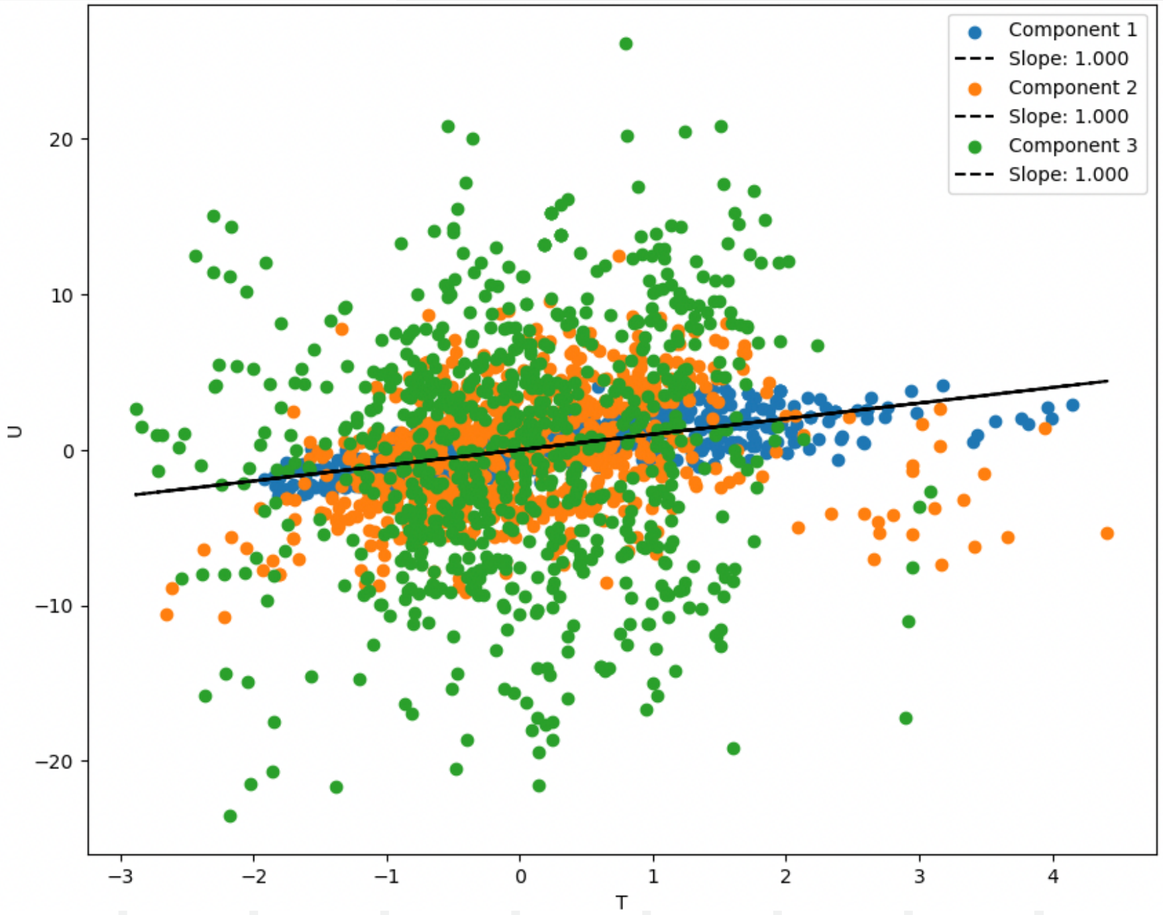
```
# Plot actual vs predicted compressive strength
plt.figure(figsize=(10, 8))
plt.scatter(y_test, y_pred, color="blue")
slope, intercept = np.polyfit(y_test.flatten(), y_pred.flatten(), 1)
plt.plot(y_test, slope * y_test + intercept, color="black", linestyle="--", label=f"Slo
plt.xlabel("Actual Compressive Strength (MPa)")
plt.ylabel("Predicted Compressive Strength (MPa)")
plt.title("Actual vs Predicted Compressive Strength using PLS")
plt.legend()
plt.show()
```



SEP 767: Multivariate Statistical Methods for Big Data Analysis and Process Improvement

McMaster University

MODEL EVALUATION



The plot shows the relationship between the scores of X and Y matrices. Each point in the plot represents the score value of the X and Y matrices for each component. The slope of each trendline shows the correlation between the scores of X and Y matrices for each component. The higher the correlation, the better the performance of the PLS model

```
# Plot t vs u
plt.figure(figsize=(10, 8))
for i in range(n_components):
    plt.scatter(T[:, i], U[:, i], label=f"Component {i+1}")
    slope, intercept = np.polyfit(T[:, i], U[:, i], 1)
    plt.plot(T[:, i], slope * T[:, i] + intercept, color="black", linestyle="--", label
plt.xlabel("T")
plt.ylabel("U")
plt.legend()
plt.show()
```

RESULTS

In conclusion, the project used the PLS algorithm to build a regression model for predicting the compressive strength of concrete based on various properties of the concrete mixture. The dataset used for the analysis contained 9 features, and the model was built using 3 PLS components.

The model achieved an R-squared value of 0.61, indicating that the model can explain 61% of the variance in the test data. The Mean Squared Error and Square Prediction Error were also calculated, with values of 0.352 for MSE

Overall, the project demonstrated the use of PLS algorithm for predicting the compressive strength of concrete based on various properties of the concrete mixture. The model achieved good performance and showed a clear linear relationship between the input features and the target variable.

ACADEMIC SOURCE AND REFERENCES

- Concrete Compressive Strength Data Set: https://archive.ics.uci.edu/ml/datasets/Concrete+Compressive+Strength
- Partial Least Squares Regression (PLS): https://en.wikipedia.org/wiki/Partial_least_squares_regression
- Nonlinear iterative partial least squares (NIPALS) algorithm: https://en.wikipedia.org/wiki/Nonlinear_iterative_partial_least_squares_algorithm
- Python Numpy library documentation: https://numpy.org/doc/stable/
- Python Matplotlib library documentation: https://matplotlib.org/stable/contents.html
- Python Seaborn library documentation: https://seaborn.pydata.org/
- https://www.kaggle.com/datasets/elikplim/concrete-compressive-strength-data-set
- I-Cheng Yeh, "Modeling of strength of high performance concrete using artificial neural networks," Cement and Concrete Research, Vol. 28, No. 12, pp. 1797-1808 (1998).

THANK YOU

Presentation by DHARMIK JOSHI
(400489130)

SEP 767: Multivariate Statistical Methods for Big Data Analysis and Process Improvement

McMaster University