Project Report

Student Performance Prediction using Linear Regression

Contents

Analysis of the Code for Linear Regression and It's Evaluation Metrics

Overview of the Dataset

The dataset used contains information about student performance, which includes the following features:

- **Hours Studied:** The number of hours a student studied.
- **Previous Scores:** The student's previous test scores.
- **Extracurricular Activities:** Whether the student participated in extracurricular activities (categorical variable, e.g., Yes or No).
- **Sleep Hours:** The number of hours the student sleeps per day.
- **Sample Question Papers Practiced:** The number of sample question papers the student practiced.
- **Performance Index (Target Variable):** The dependent variable, representing the student's overall performance on a given test or exam.

The goal of this analysis is to predict the **Performance Index** using linear regression, while also evaluating the performance of the model using various metrics.

1 Importing the Necessary Libraries

First, the code imports the necessary libraries:

- **pandas** and **numpy**: For data manipulation and numerical operations.
- **matplotlib**, **seaborn**, **plotly**: For visualizations (2D and 3D plots).
- **sklearn modules**: For data preprocessing, train-test split, model training, and evaluation.
- **scipy.stats**: For statistical plots (like the Q-Q plot).

2 Data Preprocessing and Exploratory Data Analysis (EDA)

2.1 Data Preprocessing

2.1.1 Load Dataset

- `data = pd.read_csv('Student_Performance.csv')`: Loads a CSV file into a DataFrame.
- `print()` calls display the first few rows, data types, basic statistics, column names, and shape (rows × columns).

2.1.2 Check for Missing Values

- `data.isnull().mean() * 100`: Calculates the percentage of missing values for each column. None are found.

2.1.3 Encode Categorical Variables

- 'Extracurricular Activities' is likely a text column ("Yes"/"No"), so it is converted into 0 and 1 using `LabelEncoder`.

2.1.4 Define Features (X) and Target (y)

- `X` = Selected input columns.
- `y` = 'Performance Index', the outcome to predict.

2.1.5 Train-Test Split

- Divides data into 80% training and 20% testing sets.

2.1.6 Feature Scaling

- Standardizes features (mean = 0, std = 1) to improve model performance.

2.2 Exploratory Data Analysis (EDA)

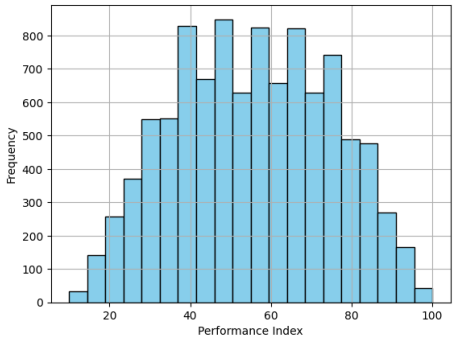
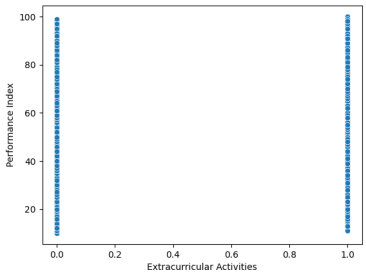
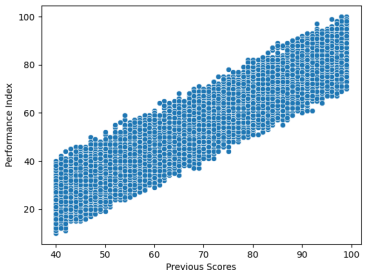


Figure 1: Distribution of Performance Index

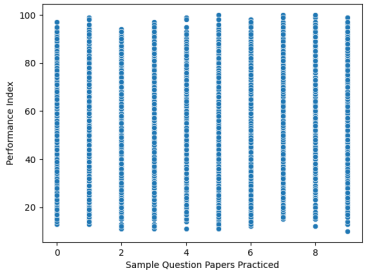
- **Distribution Plot**: Histogram of the 'Performance Index' to check the data distribution (normal, skewed, etc.).



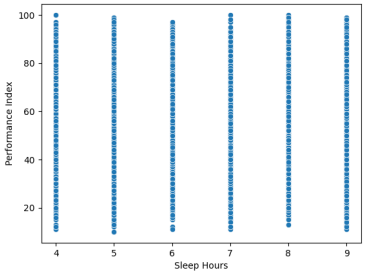
(a) Extra Activities



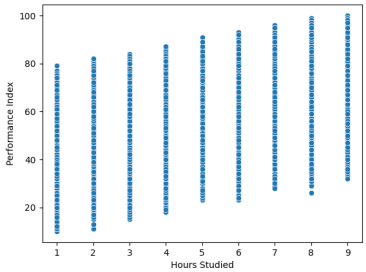
(b) Previous Scores



(c) Sample Questions



(d) Sleep Hours



(e) Hours Studied

Figure 2: Performance Index Visual Analysis

- **Scatter Plots**: For each feature (like 'Hours Studied', 'Previous Scores'), a scatter plot with 'Performance Index' is created to visually assess relationships.

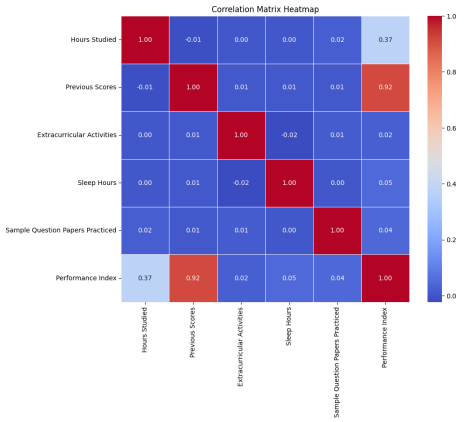


Figure 3: Correlation Matrix

- **Correlation Matrix + Heatmap**:
  - `data.corr()`: Finds correlation coefficients between numerical variables.
  - `seaborn heatmap`: Visualizes how strongly features are related to 'Performance Index'.

2.2.1 Correlation Matrix Output

- **Previous Scores** show a strong positive correlation (0.91) with the Performance Index, making them the most significant predictor.
- **Hours Studied** has a moderate positive correlation (0.37) with performance.
- **Sleep Hours** (0.048), **Sample Question Papers Practiced** (0.043), and **Extracurricular Activities** (0.024) have very weak positive correlations with the Performance Index.
- Overall, **past performance** and **study time** are more influential than other factors.

3 Linear Regression Model Implementation

3.1 Simple Linear Regression (Hours Studied only)

3.1.1 Train Simple Linear Model

- Only one feature (Hours Studied) is used to predict 'Performance Index'.
- Process: Split → Scale → Train `LinearRegression` → Predict.

3.1.2 Plot Results

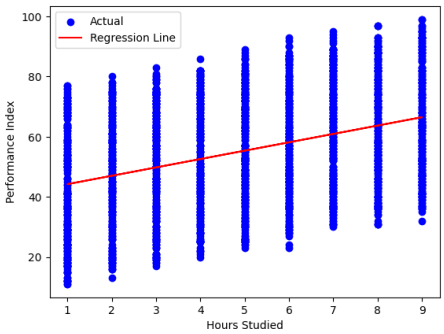


Figure 4: Visualization of Simple Linear Regresssion

- Scatter plot of actual points with regression line overlay.

3.2 Multiple Linear Regression (All Features)

3.2.1 Train Multiple Linear Model

- All five features are used.
- Coefficients (feature importance) are printed after training.

3.2.2 Multiple Linear Regression Coefficients Output

- **Previous Scores** has the highest positive coefficient (17.64), indicating it is the most influential factor on Performance Index.
- **Hours Studied** also has a significant positive impact (7.40) on performance.
- **Sleep Hours** (0.81), **Sample Question Papers Practiced** (0.55), and **Extracurricular Activities** (0.30) contribute positively but to a much smaller extent.
- Overall, **Previous Scores** and **Hours Studied** are the dominant predictors.

4 Model Evaluation

4.1 Simple Linear Regression Evaluation Metrics Output

- **MAE (Mean Absolute Error)**: 15.53
  This indicates that, on average, the model's predictions are off by approximately 15.53 units from the actual values.

- **MSE (Mean Squared Error)**: 321.81
  The model has a high MSE, meaning the squared differences between predicted and actual values are large on average, suggesting the model is not a very good fit.

- **RMSE (Root Mean Squared Error)**: 17.94
  The RMSE indicates that the model's predictions, on average, deviate from the true values by about 17.94 units.

- **R^2 (R-squared)**: 0.1316
  Only about 13.16% of the variance in the target variable (Performance Index) is explained by the Simple Linear Regression model, which shows the model has poor predictive power.

- **Adjusted R^2**: 0.1312
  The adjusted R^2 is similar to R^2 and shows that even after considering the number of predictors, the model still explains only 13.12% of the variance in the target variable.

4.2 Multiple Linear Regression Metrics Output

- **MAE (Mean Absolute Error)**: 1.61
  The model's predictions are off by only 1.61 units on average, showing a significant improvement in accuracy compared to the SLR model.

- **MSE (Mean Squared Error)**: 4.08
  The lower MSE indicates that the model's squared errors are much smaller compared to the SLR model, showing better fit and accuracy.

- **RMSE (Root Mean Squared Error)**: 2.02
  The RMSE suggests that the Multiple Linear Regression model's predictions are, on average, only 2.02 units away from the actual values, a significant improvement over SLR.

- **R^2 (R-squared)**: 0.9890
  The model explains 98.90% of the variance in the target variable, which means it has a very high predictive power and is a much better fit compared to the Simple Linear Regression model.

- **Adjusted R^2**: 0.9890
  The adjusted R^2 shows that after adjusting for the number of predictors, the Multiple Linear Regression model still explains 98.90% of the variance in the target variable, confirming a very strong fit.

4.3 Residual Analysis

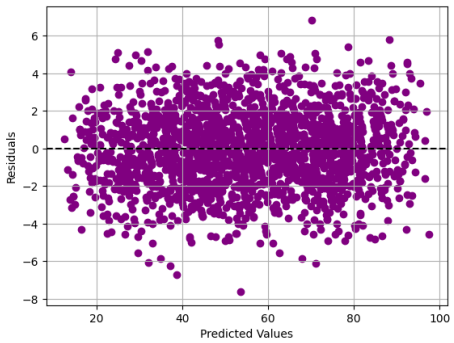


Figure 5: Scatter plot of the Residual Points

Residuals:

- `residual = actual - predicted`
- Plot residuals vs predicted values to check:
  - Random spread? (Good model)
  - Pattern? (Problem like non-linearity)

4.3.1 Check Normality of Residuals

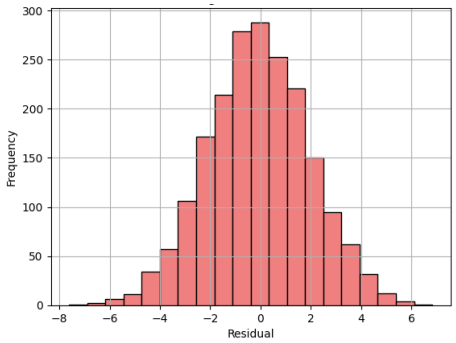


Figure 6: Histogram of Residuals

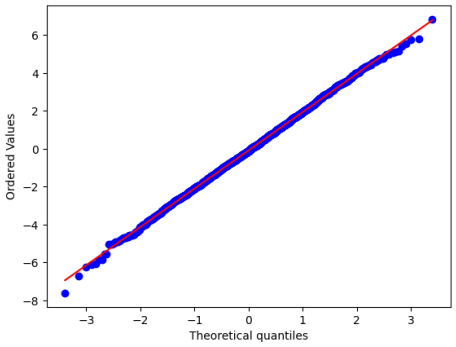


Figure 7: Q-Q Plot of Residuals

- Histogram: Should look normal (bell-shaped) as shown.
- Q-Q Plot: Should follow the diagonal line if residuals are normal, as shown.

5 3D Visualization (Multiple Linear Regression)

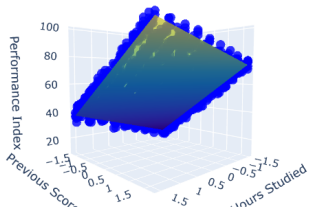


Figure 8: Multiple Linear Regression Plane

- 3D Scatter Plot + Regression Plane:
  - Takes two features ('Hours Studied', 'Previous Scores').
  - Plot actual points and create a 3D regression plane.
  - Uses Plotly for interactive 3D visualization.

6 Regularization Techniques with Cross-Validation

6.1 Why Regularization?

- Ordinary Linear Regression can overfit when features are correlated or too many.

- Regularization techniques penalize large coefficients.

6.2 Ridge Regression (L2 Regularization)

- `RidgeCV` uses Cross-Validation (5 folds) to automatically find best alpha (penalty term).

6.3 Lasso Regression (L1 Regularization)

- `LassoCV` similarly finds best alpha.
- Also performs feature selection by shrinking some coefficients exactly to 0.

6.4 Best Alpha Values for Ridge and Lasso Regression

- **Best Ridge alpha**: 0.0001
  This is the optimal regularization parameter for the Ridge Regression model, which helps balance the trade-off between bias and variance.
- **Best Lasso alpha**: 0.00032374575428176434
  This is the optimal regularization parameter for the Lasso Regression model, which performs both regularization and variable selection.

7 Final Model Comparisons

- A function `evaluate_model()` is defined to print performance metrics.
- Original Linear, Ridge, and Lasso models are compared using:

Metric	Original MLR	Ridge Regression	Lasso Regression
MAE	1.61	1.61	1.61
MSE	4.08	4.08	4.08
RMSE	2.02	2.02	2.02
R^2	0.9890	0.9890	0.9890

Table 1: Comparison of Regression Models Evaluation Metrics

7.1 Feature Importance using Lasso Regression

Lasso regression helps identify the most important features by shrinking the coefficients of less relevant features to zero. We analyzed the coefficients and found the top 3 important features:

Top 3 Important Features from Lasso Regression:
['Previous Scores', 'Hours Studied', 'Sleep Hours']

Next, we retrained the model using only these features and evaluated its performance:

Model Using Only Top 3 Features:

Top 3 Features Multiple Linear Regression Evaluation:
MAE: 1.70
MSE: 4.55
RMSE: 2.13
R^2: 0.9877

The model performed nearly as well as the original, with an R^2 of 0.9877, indicating 98.77% of the variation in the performance index was explained. Error metrics (MAE, MSE, RMSE) slightly increased, but the model remains highly accurate with fewer features.

8 Scenario-Based Question 1: Overfitting and Model Complexity

After training the multiple linear regression (MLR) model on the provided dataset, we observed high R^2 values for both training and test data, indicating no significant overfitting.

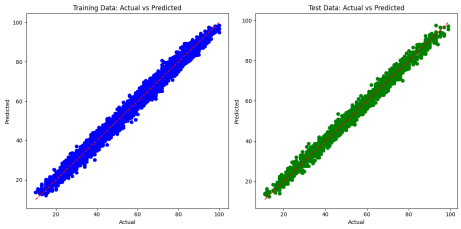


Figure 9: MLR on Training Data (Left) MLR on Test Data (Right) Visualized

8.1 Diagnosis of Overfitting

- **R^2 Scores**: The model achieved R^2 values of 0.9887 for the training set and 0.9890 for the test set, which are nearly identical. This suggests that the model does not overfit the training data since the performance is consistent across both datasets. - **Error Metrics**: The MAE, MSE, and RMSE values are also very similar for both training and test sets, indicating no significant performance degradation on the test set.

8.2 Evaluation Metrics Comparison

Metric	Training Data	Test Data
MAE (Mean Absolute Error)	1.62	1.61
MSE (Mean Squared Error)	4.17	4.08
RMSE (Root Mean Squared Error)	2.04	2.02
R^2 (R-squared)	0.9887	0.9890
Adjusted R^2	0.9887	0.9890

8.3 Proposed Strategy

Although the model does not exhibit significant overfitting, **regularization** methods like **Lasso** or **Ridge regression** can still be useful for further improvement by controlling model complexity. Regularization reduces overfitting by adding a penalty to the magnitude of coefficients, encouraging the model to focus on the most important features.

We will apply Lasso Regression to address any potential issues of overfitting in the current model and evaluate its performance on the test data.

8.4 Conclusion

The model does not show signs of overfitting based on the evaluation metrics. However, implementing regularization (such as Lasso) can further enhance the model's robustness by penalizing irrelevant features and improving generalization.

8.5 Scenario 2: Real-World Application - Predicting Student Performance

The Multiple Linear Regression (MLR) model provides the following actionable insights for improving student performance:

- **Study Hours:** Increasing study hours directly boosts performance. Students should aim for more focused study time.
- **Previous Scores:** Students with higher previous scores tend to perform better. Identifying those with lower past scores helps target extra support.
- **Sleep Hours:** Adequate sleep (7-8 hours) enhances cognitive function, significantly improving performance.
- **Extracurricular Activities:** While less impactful, balanced extracurricular involvement contributes to overall well-being and time management.
- **Practice Papers:** Regularly practicing sample question papers can improve exam preparedness and reduce stress.

8.6 Conclusion

Focus on increasing study hours, ensuring adequate sleep, and providing support to students with lower previous scores. Balancing extracurriculars and practicing sample papers further enhances performance.

9 Code

```
# Imports
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import plotly.graph_objects as go

from sklearn.preprocessing import StandardScaler, LabelEncoder
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score
import scipy.stats as stats
 Part 1: Data Preprocessing and Exploratory Data Analysis (EDA)

# --- Part 1: Data Preprocessing  ---

# Step 1: Load data
data = pd.read_csv('Student_Performance.csv')
print(data.head())
print(data.info())
print(data.describe())
print(data.columns)
print(data.shape)
# Step 1b: Check and handle missing values
missing_percentage = data.isnull().mean() * 100
print("\nMissing Values (%):\n", missing_percentage)
# Step 2: Encode categorical variables
le = LabelEncoder()
data['Extracurricular Activities'] = le.fit_transform(data['Extracurricular Activities'])
print(data.head())
# Step 3: Define features (X) and target (y)
X = data[['Hours Studied', 'Previous Scores', 'Extracurricular Activities', 'Sleep Hours',
y = data['Performance Index']
# Step 4: Train-test split (80:20)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
# Step 5: Feature scaling (Standardization)
scaler = StandardScaler()
```

```
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# --- Exploratory Data Analysis (EDA) ---

# Step 6: Visualize distribution of Performance Index
plt.hist(data['Performance Index'], bins=20, color='skyblue', edgecolor='black')
plt.title('Distribution of Performance Index')
plt.xlabel('Performance Index')
plt.ylabel('Frequency')
plt.grid(True)
plt.show()
# Step 7: Scatter plots
independent_vars = ['Hours Studied', 'Previous Scores', 'Extracurricular Activities', 'Sleep
for var in independent_vars:
    sns.scatterplot(data=data, x=var, y='Performance Index')
    plt.title(f'Performance Index vs {var}')
    plt.xlabel(var)
    plt.ylabel('Performance Index')
    plt.show()
# Step 8: Correlation matrix and heatmap
corr_matrix = data.corr(numeric_only=True)
target_corr = corr_matrix['Performance Index'].sort_values(ascending=False)
print("\nCorrelation with Performance Index:\n", target_corr)

plt.figure(figsize=(10, 8))
sns.heatmap(corr_matrix, annot=True, cmap='coolwarm', fmt=".2f", linewidths=0.5)
plt.title('Correlation Matrix Heatmap')
plt.show()
Part 2: Linear Regression Model Implementation
# --- Part 2: Linear Regression Model Implementation ---

# Step 9: Simple Linear Regression (Hours Studied vs Performance Index)

# Define for simple regression
X_simple = data[['Hours Studied']]
y_simple = data['Performance Index']

X_train_s, X_test_s, y_train_s, y_test_s = train_test_split(X_simple, y_simple, test_size=0.
# Scaling
scaler_simple = StandardScaler()
X_train_s_scaled = scaler_simple.fit_transform(X_train_s)
X_test_s_scaled = scaler_simple.transform(X_test_s)
# Train simple linear regression model
lr_simple = LinearRegression()
lr_simple.fit(X_train_s_scaled, y_train_s)
```

```
# Predict and plot
y_pred_simple = lr_simple.predict(X_test_s_scaled)

plt.scatter(X_test_s, y_test_s, color='blue', label='Actual')
plt.plot(X_test_s, y_pred_simple, color='red', label='Regression Line')
plt.xlabel('Hours Studied')
plt.ylabel('Performance Index')
plt.title('Simple Linear Regression: Hours Studied vs Performance Index')
plt.legend()
plt.show()
Multiple Linear Regression (All features)
# Step 10: Multiple Linear Regression (All features)

# Train multiple regression model
lr_multiple = LinearRegression()
lr_multiple.fit(X_train_scaled, y_train)
# Predict
y_pred_multiple = lr_multiple.predict(X_test_scaled)


# Coefficients
coefficients = pd.DataFrame({'Feature': X.columns, 'Coefficient': lr_multiple.coef_})
print("\nMultiple Linear Regression Coefficients:\n", coefficients)
Part 3: Model Evaluation
# --- Part 3: Model Evaluation ---

# Step 11: Evaluation Metrics for Simple Linear Regression
mae_simple = mean_absolute_error(y_test_s, y_pred_simple)
mse_simple = mean_squared_error(y_test_s, y_pred_simple)
rmse_simple = np.sqrt(mse_simple)
r2_simple = r2_score(y_test_s, y_pred_simple)
adjusted_r2_simple = 1 - (1 - r2_simple) * (len(y_test_s) - 1) / (len(y_test_s) - X_train_s.

print("\nSimple Linear Regression Metrics:")
print(f"MAE: {mae_simple:.2f}")
print(f"MSE: {mse_simple:.2f}")
print(f"RMSE: {rmse_simple:.2f}")
print(f"R²: {r2_simple:.4f}")
print(f"Adjusted R²: {adjusted_r2_simple:.4f}")

# Step 12: Evaluation Metrics for Multiple Linear Regression
mae_multiple = mean_absolute_error(y_test, y_pred_multiple)
mse_multiple = mean_squared_error(y_test, y_pred_multiple)
rmse_multiple = np.sqrt(mse_multiple)
r2_multiple = r2_score(y_test, y_pred_multiple)
adjusted_r2_multiple = 1 - (1 - r2_multiple) * (len(y_test) - 1) / (len(y_test) - X_train.sh
```

```
print("\nMultiple Linear Regression Metrics:")
print(f"MAE: {mae_multiple:.2f}")
print(f"MSE: {mse_multiple:.2f}")
print(f"RMSE: {rmse_multiple:.2f}")
print(f"R²: {r2_multiple:.4f}")
print(f"Adjusted R²: {adjusted_r2_multiple:.4f}")

Residual Analysis
# --- Residual Analysis ---

# Step 13: Residuals calculation
residuals = y_test - y_pred_multiple

# Plot residuals
plt.scatter(y_pred_multiple, residuals, color='purple')
plt.axhline(y=0, color='black', linestyle='--')
plt.title('Residuals vs Predicted Values (Multiple Linear Regression)')
plt.xlabel('Predicted Values')
plt.ylabel('Residuals')
plt.grid(True)
plt.show()
# Step 14: Check if residuals are normally distributed (Histogram + Q-Q Plot)
# Histogram
plt.hist(residuals, bins=20, edgecolor='black', color='lightcoral')
plt.title('Histogram of Residuals')
plt.xlabel('Residual')
plt.ylabel('Frequency')
plt.grid(True)
plt.show()

# Q-Q plot
stats.probplot(residuals, dist="norm", plot=plt)
plt.title('Q-Q Plot of Residuals')
plt.show()
3D Visualization with Multiple Linear Regression

# --- 3D Visualization (Optional Bonus) ---

# Step 15: 3D Regression Visualization with two features
X_3d = data[['Hours Studied', 'Previous Scores']]
y_3d = data['Performance Index']

X_train_3d, X_test_3d, y_train_3d, y_test_3d = train_test_split(X_3d, y_3d, test_size=0.2, r

scaler_3d = StandardScaler()
```

```
X_train_3d_scaled = scaler_3d.fit_transform(X_train_3d)
X_test_3d_scaled = scaler_3d.transform(X_test_3d)

lr_3d = LinearRegression()
lr_3d.fit(X_train_3d_scaled, y_train_3d)

hours = X_test_3d_scaled[:, 0]
previous_scores = X_test_3d_scaled[:, 1]
performance_index = y_test_3d

hours_range = np.linspace(hours.min(), hours.max(), 30)
previous_scores_range = np.linspace(previous_scores.min(), previous_scores.max(), 30)
hours_mesh, previous_scores_mesh = np.meshgrid(hours_range, previous_scores_range)

features_flat = np.vstack((hours_mesh.ravel(), previous_scores_mesh.ravel())).T
performance_pred_flat = lr_3d.predict(features_flat)
performance_pred_mesh = performance_pred_flat.reshape(hours_mesh.shape)

fig = go.Figure()

fig.add_trace(go.Scatter3d(
    x=hours, y=previous_scores, z=performance_index,
    mode='markers',
    marker=dict(size=6, color='blue', opacity=0.8),
    name='Actual Data'
))

fig.add_trace(go.Surface(
    x=hours_mesh, y=previous_scores_mesh, z=performance_pred_mesh,
    colorscale='Viridis',
    opacity=0.7,
    name='Regression Plane'
))

fig.update_layout(
    title="Multiple Linear Regression (3D View)",
    scene=dict(
        xaxis_title='Hours Studied ',
        yaxis_title='Previous Scores ',
        zaxis_title='Performance Index'
    )
)

fig.show()
```

Regularization Techniques with Cross-Validation

```
# --- 7. Regularization Techniques ---

from sklearn.linear_model import RidgeCV, LassoCV
from sklearn.model_selection import cross_val_score
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score
import numpy as np
import pandas as pd
# Step 1: Ridge Regression with Cross-Validation
ridge_alphas = np.logspace(-4, 4, 50)
ridge_cv = RidgeCV(alphas=ridge_alphas, scoring='neg_mean_squared_error', cv=5)
ridge_cv.fit(X_train_scaled, y_train)

y_pred_ridge = ridge_cv.predict(X_test_scaled)

print("\nBest Ridge alpha:", ridge_cv.alpha_)

# Step 2: Lasso Regression with Cross-Validation
lasso_alphas = np.logspace(-4, 1, 50)
lasso_cv = LassoCV(alphas=lasso_alphas, max_iter=10000, cv=5, random_state=42)
lasso_cv.fit(X_train_scaled, y_train)

y_pred_lasso = lasso_cv.predict(X_test_scaled)

print("\nBest Lasso alpha:", lasso_cv.alpha_)
# Step 3: Evaluation function
def evaluate_model(name, y_test, y_pred):
    print(f"\n{name} Evaluation:")
    print(f"MAE: {mean_absolute_error(y_test, y_pred):.2f}")
    print(f"MSE: {mean_squared_error(y_test, y_pred):.2f}")
    print(f"RMSE: {np.sqrt(mean_squared_error(y_test, y_pred)):.2f}")
    print(f"R²: {r2_score(y_test, y_pred):.4f}")


# Step 4: Compare models
print("\nOriginal Multiple Linear Regression:")
evaluate_model("Multiple Linear Regression", y_test, y_pred_multiple)

print("\nRidge Regression:")
evaluate_model("Ridge Regression", y_test, y_pred_ridge)

print("\nLasso Regression:")
evaluate_model("Lasso Regression", y_test, y_pred_lasso)


Feature Importance using Lasso
# --- 8. Feature Importance using Lasso ---
```

```
# Step 5: Identify important features
lasso_coefficients = pd.Series(lasso_cv.coef_, index=X.columns)
# Sort by absolute coefficient value
important_features = lasso_coefficients.abs().sort_values(ascending=False)
top_3_features = important_features.head(3).index.tolist()

print("\nTop 3 Important Features from Lasso Regression:\n", top_3_features)
# Step 6: Retrain using only top 3 features
X_train_top3 = X_train_scaled[:, [X.columns.get_loc(f) for f in top_3_features]]
X_test_top3 = X_test_scaled[:, [X.columns.get_loc(f) for f in top_3_features]]

# Train a new  model
from sklearn.linear_model import LinearRegression
lr_top3 = LinearRegression()
lr_top3.fit(X_train_top3, y_train)
y_pred_top3 = lr_top3.predict(X_test_top3)

# Step 7: Evaluate the model with top 3 features
print("\nModel Using Only Top 3 Features:")
evaluate_model("Top 3 Features Multipile Linear Regression", y_test, y_pred_top3)
```