INDIAN INSTITUTE OF TECHNOLOGY, MANDI

Assignment 2

Introduction to Statistical Sciences

Submitted By:

Sri Sahithi Sunkaranam (B23503)

Abhey Kumar (B23391)

Submitted on
April 11, 2025

Simple Regression

The CarSeats dataset containing sales-related information for car seats across various stores. It includes both numerical and categorical features such as `sales figures`, `competitor pricing`, `income levels`, and `advertising budgets`. The dataset also captures demographic details like `population, age, and education level`. Categorical variables like `shelf location quality, urban status, and US location` are included to enhance analysis.

No	Sales	CompPrice	Income	Advertising	Population	Price	ShelveLoc	Age	Education	Urban	US
1	9.5	138	73	11	276	120	Bad	42	17	Yes	Yes
2	11.22	111	48	16	260	83	Good	65	10	Yes	Yes
3	10.06	113	35	10	269	80	Medium	59	12	Yes	Yes
4	7.4	117	100	4	466	97	Medium	55	14	Yes	Yes
5	4.15	141	64	3	340	128	Bad	38	13	Yes	No

Part (a): Training RSS for Polynomial Regression Models

We are tasked with evaluating how the complexity of the regression model affects training performance. Assuming that the true relationship between `Sales` and `Population` is linear, we compare a linear regression model (degree = 1) with polynomial regression models of degrees 2 through 5.

Code

```python
from sklearn.preprocessing import PolynomialFeatures
from sklearn.linear_model import LinearRegression
import numpy as np
import matplotlib.pyplot as plt

train_rss = []

for degree in range(1, 6):
    poly = PolynomialFeatures(degree=degree)
    X_train_poly = poly.fit_transform(X_train)

    model = LinearRegression()
    model.fit(X_train_poly, y_train)

    y_train_pred = model.predict(X_train_poly)
    rss = np.sum((y_train - y_train_pred) ** 2)
    train_rss.append(rss)
    print(f'Degree {degree} - Training RSS: {rss:.4f}')

plt.figure(figsize=(8,5))
plt.plot(range(1, 6), train_rss, marker='o', color='blue')
plt.title('Training RSS vs Polynomial Degree')
plt.xlabel('Polynomial Degree (n)')
plt.ylabel('Training RSS')
plt.grid(True)
plt.show()
```

Listing 1: Polynomial Regression: Training RSS

Polynomial regression expands the feature space by adding higher-degree terms of the predictor variable. A model of degree n will perfectly fit any n-point data if n is large enough. However, this comes with the risk of overfitting.

Results

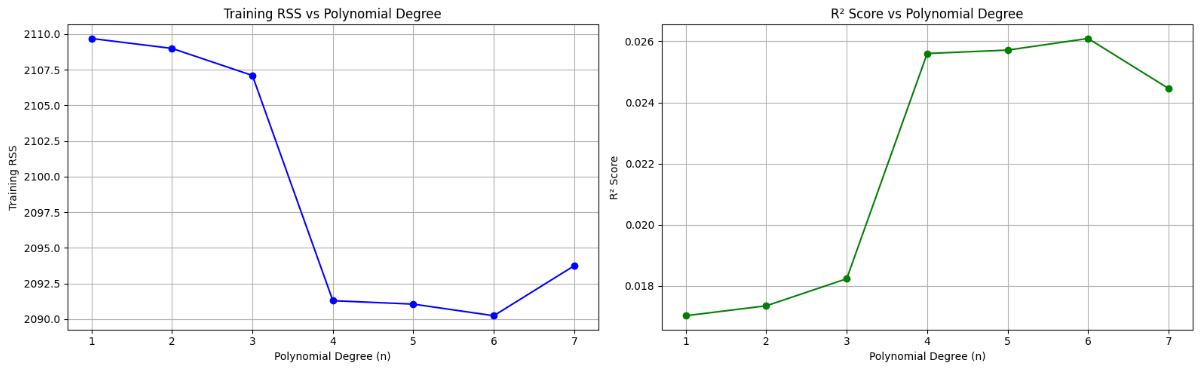


Figure 1: Left: Training RSS vs Polynomial Degree. Right: R^2 Score vs Polynomial Degree.

Analysis

- **Training RSS decreases with degree:** This is expected because polynomial models with higher degrees can model the training data more closely, including capturing noise.
- **R^2 The score increases then drops:** As the degree increases from 1 to 6, R2 improves, suggesting a better fit. However, at degree 7, R^2 declines, indicating the onset of overfitting.
- **Implication of a Linear True Relationship:** Since we assume the true underlying relationship is linear, the linear model (degree 1) should theoretically generalize

best. Lower RSS in more complex models does not necessarily mean better real-world performance; it just means better training fit.

- **Why higher-degree models perform better here:** In training data, these models have more parameters to minimize RSS. But in practice, they may not generalize well unless the true relationship is nonlinear.

This shows that the training error (RSS) always decreases with the complexity of the model, regardless of the true relationship. However, model selection should not be based solely on training error. Test error or cross-validation should be used to find the model with best generalization, which will be explored in Part (b).

Part (b): Test RSS for Polynomial Regression Models

We now evaluate how the model generalizes by calculating the Test RSS for the same polynomial degrees. This helps in understanding the model's ability to perform on unseen data.

Code

```python
from sklearn.preprocessing import PolynomialFeatures
from sklearn.linear_model import LinearRegression
from sklearn.metrics import r2_score
import matplotlib.pyplot as plt
import numpy as np

test_rss = []
r2_scores_test = []

for degree in range(1, 8):
    poly = PolynomialFeatures(degree=degree)
    X_train_poly = poly.fit_transform(X_train)
    X_test_poly = poly.transform(X_test)

    model = LinearRegression()
    model.fit(X_train_poly, y_train)

    y_test_pred = model.predict(X_test_poly)
    rss = np.sum((y_test - y_test_pred) ** 2)
    r2 = r2_score(y_test, y_test_pred)

    test_rss.append(rss)
    r2_scores_test.append(r2)
```

Listing 2: Polynomial Regression: Test RSS

Results

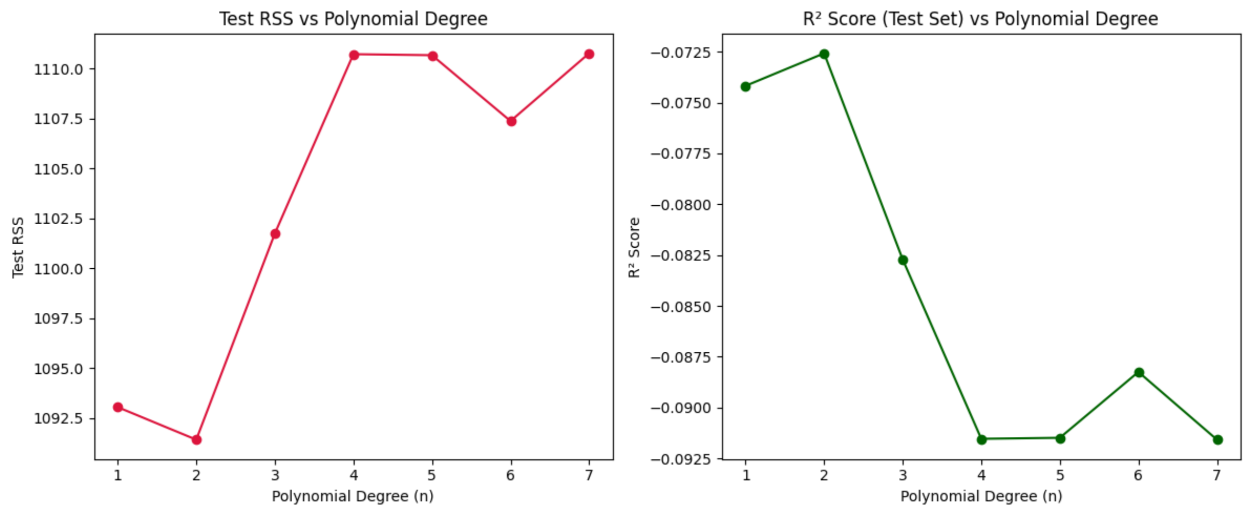


Figure 2: Left: Test RSS vs Polynomial Degree. Right: R^2 Score on Test Set vs Polynomial Degree.

Analysis

- **Test RSS:** The lowest Test RSS is of polynomial degree 2. As degree increases beyond that, the test RSS rises sharply, indicating overfitting.
- **R^2 Score:** All R^2 scores are negative, suggesting that the models perform worse than simply predicting the mean. Still, degree 2 gives the least negative R^2, making it the relatively best model.
- **Overfitting Evidence:** The sharp increase in test RSS and the decrease in R2 beyond degree 2 shows that higher-degree models are too complex and fit noise from the training set.
- **Mismatch With Training Results:** While degree 6 had the lowest training RSS, it performs poorly on test data. This reaffirms that training performance alone is not a good indicator of generalization.
- **Final Verdict:** Based on the performance of the test, the optimal polynomial degree is **2**, balancing model complexity and generalization.

The analysis confirms that a polynomial degree of 2 gives the best generalization to unseen data. It demonstrates the vital role of using a test set in model selection to prevent overfitting and ensure reliability.

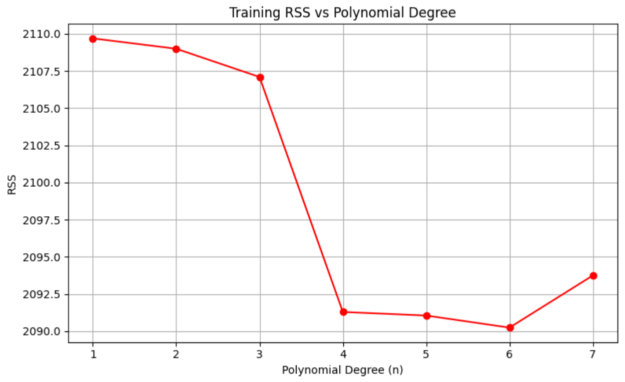
Part (c): Comparison of Training RSS: Linear vs Polynomial Regression

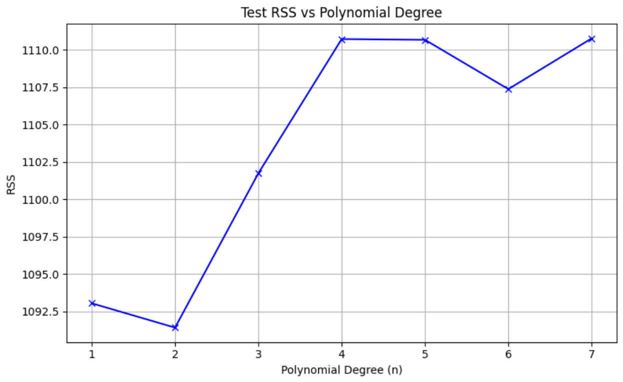
If the true relationship between X and Y is not linear, a polynomial regression model is more flexible and capable of fitting the training data more closely than a simple linear model. Therefore, we would expect the **training RSS for polynomial regression to be lower** than that of linear regression. This is because increasing the degree of the polynomial allows the model to capture more complex patterns in the data, potentially reducing the residual error.

Part (d): Comparison of Test RSS: Linear vs Polynomial Regression

Based on the test RSS plot, the degree 2 polynomial regression model achieves a lower test RSS than the linear model (degree 1), suggesting that it captures the underlying nonlinearity more effectively without overfitting. However, for degrees greater than 2, the test RSS increases, indicating overfitting. Thus, the polynomial model initially outperforms the linear model in the test set but becomes worse as the degree increases.

Part (e) Plot of Polynomial Degree vs RSS





The plot shows how the training RSS steadily decreases with increasing polynomial degree, while the test RSS reaches a minimum at degree 2 before rising again. This demonstrates the classic trade-off between underfitting and overfitting.

Multiple Linear Regression

Dataset Description

We again use the `CarSeats` dataset containing sales-related information for car seats across various stores.

Part (a): Multiple Linear Regression Model

We aim to build a multiple linear regression model to predict Sales of car seats using selected predictor variables. Specifically, we'll use Price, Urban, and US as independent variables. These features represent pricing strategy and location-based factors that may influence sales performance. The goal is to understand how each factor affects sales and assess their significance.

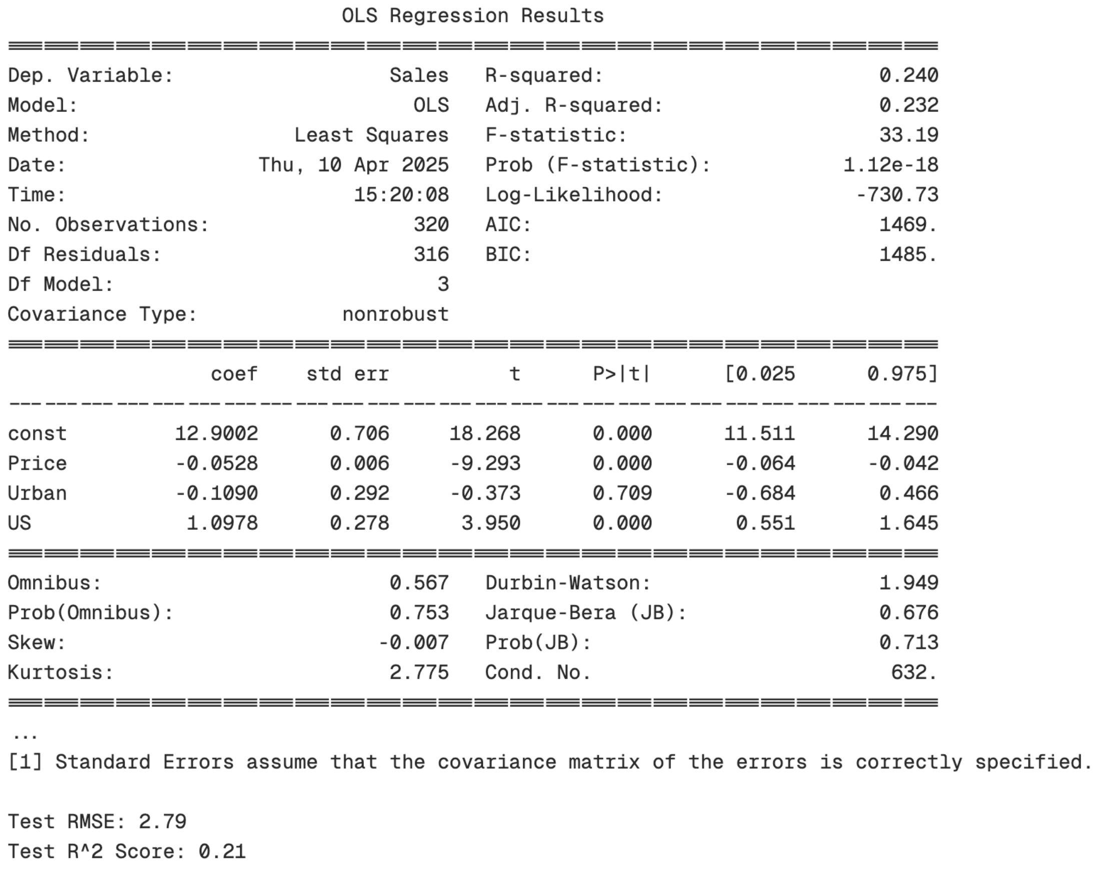
Code

```python
# Import required libraries
import pandas as pd
import statsmodels.api as sm
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error, r2_score
import numpy as np

# Load and encode data
df = pd.read_csv('./Datasets/Dataset for regression/CarSeats_dataset_regression.csv')
df['Urban'] = df['Urban'].map({'Yes': 1, 'No': 0})
df['US'] = df['US'].map({'Yes': 1, 'No': 0})

# Feature-target split
X = df[['Price', 'Urban', 'US']]
y = df['Sales']

# Train-test split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Add constant and fit model
X_train_const = sm.add_constant(X_train)
X_test_const = sm.add_constant(X_test)
model = sm.OLS(y_train, X_train_const).fit()

# Predict and evaluate
y_pred = model.predict(X_test_const)
```

```
rmse = np.sqrt(mean_squared_error(y_test , y_pred))
r2 = r2_score(y_test , y_pred)
```

We preprocess the data by encoding the categorical features as binary values. A multiple linear regression model is fit using only three predictors: `Price`, `Urban`, and `US`. We split the data to ensure we evaluate how well the model generalizes, not just fits. The model is trained using the OLS method and evaluated using RMSE and R^2 score on the test set.

Result / Output



```
                            OLS Regression Results
==============================================================================
Dep. Variable:                  Sales   R-squared:                       0.240
Model:                            OLS   Adj. R-squared:                  0.232
Method:                 Least Squares   F-statistic:                     33.19
Date:                Thu, 10 Apr 2025   Prob (F-statistic):           1.12e-18
Time:                        15:20:08   Log-Likelihood:                -730.73
No. Observations:                 320   AIC:                             1469.
Df Residuals:                     316   BIC:                             1485.
Df Model:                           3
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const         12.9002      0.706     18.268      0.000      11.511      14.290
Price         -0.0528      0.006     -9.293      0.000      -0.064      -0.042
Urban         -0.1090      0.292     -0.373      0.709      -0.684       0.466
US             1.0978      0.278      3.950      0.000       0.551       1.645
==============================================================================
Omnibus:                        0.567   Durbin-Watson:                   1.949
Prob(Omnibus):                  0.753   Jarque-Bera (JB):                0.676
Skew:                          -0.007   Prob(JB):                        0.713
Kurtosis:                       2.775   Cond. No.                         632.
==============================================================================
...
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.

Test RMSE: 2.79
Test R^2 Score: 0.21
```

Analysis of the Output

The model summary offers several key insights into the performance and interpretation of the regression:

- **R-squared (0.240):** This value implies that around 24% of the variance in `Sales` is explained by the predictors `Price`, `Urban`, and `US`. While not very high, this suggests the model has some explanatory power, but other important features are likely missing.

- **Intercept (const = 12.90):** This is the expected `Sales` when all predictors are 0. While not always practically meaningful, it serves as the model baseline.
- **Price (-0.0528):** This is a key predictor. For every one-unit increase in `Price`, `Sales` are expected to decrease by 0.0528 units, assuming other variables are held constant. The p-value here is ¡ 0.001, confirming strong statistical significance.
- **Urban (-0.1090):** Being in an urban location is associated with a slight decrease in `Sales`, but with a high p-value (0.709), this predictor is not statistically significant and likely not useful in the model.
- **US (1.0978):** Stores in the US are expected to sell 1.10 more units on average compared to non-US stores, all else equal. This variable is significant (p ¡ 0.001).
- **Confidence Intervals:** For example, the 95% CI for `Price` is [-0.064, -0.042], reinforcing that its negative effect on `Sales` is statistically robust.
- **Test RMSE (2.79):** This means that on average, the model's sales predictions are off by about 2.79 units — a metric of practical predictive accuracy.
- **Test $\mathbf{R^2}$ Score (0.21):** On unseen data, the model explains 21% of the variance in `Sales`, slightly lower than the training R^2. This suggests modest generalization capability.

Overall, the model confirms that `Price` is a strong negative predictor of sales, and being in the `US` positively impacts sales. However, the `Urban` variable does not appear to significantly affect sales. The moderate R^2 and RMSE suggest the model can be improved with more relevant predictors.

Part (b): Interpretation of Coefficients

- **Intercept (12.9002):** When `Price` = 0, and the store is neither `Urban` nor in the `US`, the expected sales would be approximately 12.90 units. Although unrealistic (since a Price of 0 doesn't make business sense), it's necessary to anchor the regression line.
- **Price (-0.0528):** For every unit increase in `Price`, the expected `Sales` decrease by approximately 0.0528 units, holding other factors constant. This aligns with economic intuition: higher prices typically reduce sales.

- **Urban (-0.1090):** The coefficient is negative, suggesting that urban stores sell about 0.11 fewer units on average than non-urban stores, all else equal. However, its high p-value indicates this effect is not statistically significant.

- **US (1.0978):** Stores located in the US are expected to sell approximately 1.10 more units than those located elsewhere, controlling for other variables. This coefficient is statistically significant and meaningful.

Part (c): Model Equation

Based on the estimated coefficients, the regression equation can be written as:

$$\hat{Sales} = 12.9002 - 0.0528 \cdot Price - 0.1090 \cdot Urban + 1.0978 \cdot US$$

Where:

- `Urban` = 1 if the store is in an urban area, 0 otherwise

- `US` = 1 if the store is in the US, 0 otherwise

This equation can be used to predict `Sales` for new data based on the values of `Price`, `Urban`, and `US`.

Part (d): Hypothesis Testing for Predictors

To assess which predictors are statistically significant, we examine the p-values associated with each coefficient. We use a significance threshold of $\alpha = 0.05$.

- **Price (p-value = 0.000):** Since the p-value is less than 0.05, we reject the null hypothesis. `Price` is a statistically significant predictor of `Sales`.

- **Urban (p-value = 0.709):** The p-value is much greater than 0.05. Thus, we fail to reject the null hypothesis. There is insufficient evidence to conclude that the `Urban` variable affects `Sales`.

- **US (p-value = 0.000):** The p-value is well below 0.05, so we reject the null hypothesis. Being located in the `US` has a significant effect on `Sales`.

Conclusion: Among the predictors, `Price` and `US` are statistically significant, while `Urban` is not.

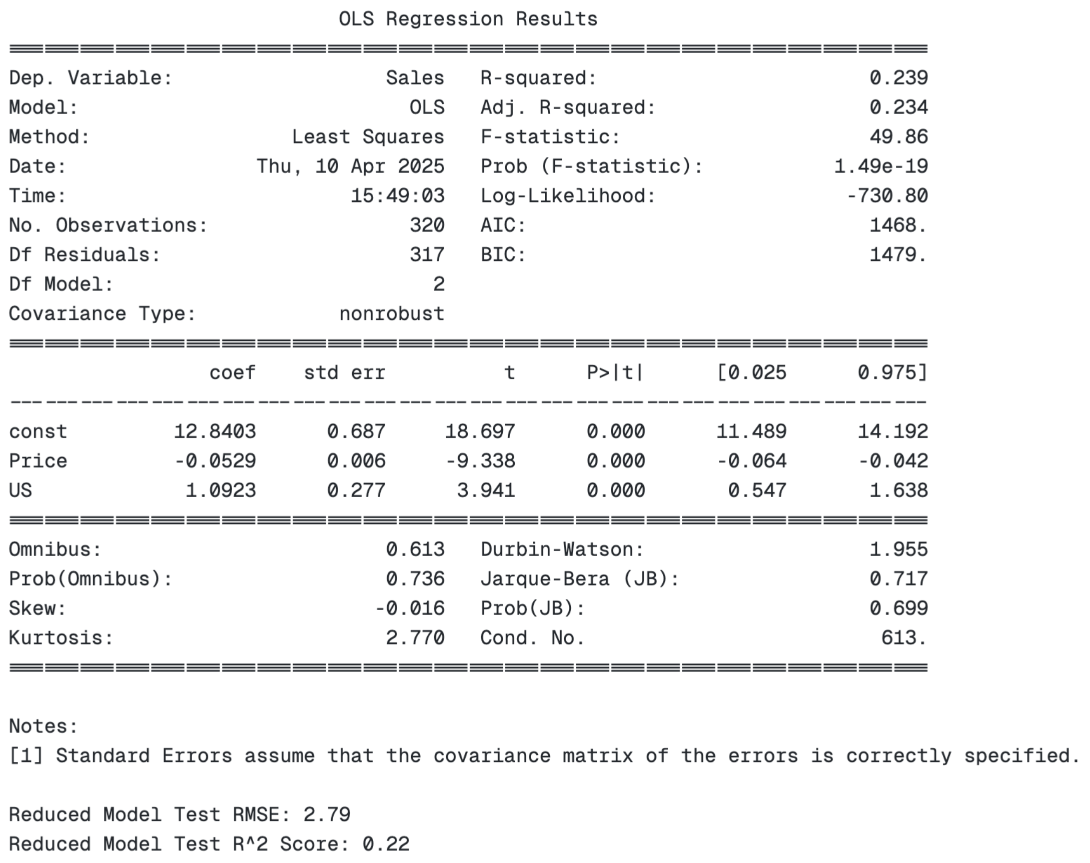
Part (e): Fitting a Reduced Model with Significant Predictors

Based on the hypothesis testing performed in Part (d), we identified that only `Price` and `US` are statistically significant predictors of `Sales`. In this section, we fit a new multiple linear regression model using only these two variables and evaluate its performance.

Reduced Model Code

```python
import statsmodels.api as sm
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error, r2_score
import numpy as np

# Select only significant predictors
X_reduced = df[['Price', 'US']]
X_reduced = sm.add_constant(X_reduced)
y = df['Sales']

# Split the dataset
X_train_red, X_test_red, y_train_red, y_test_red = train_test_split(
    X_reduced, y, test_size=0.2, random_state=42)

# Fit the model
reduced_model = sm.OLS(y_train_red, X_train_red).fit()

# Predict and evaluate
y_pred_red = reduced_model.predict(X_test_red)
rmse_red = np.sqrt(mean_squared_error(y_test_red, y_pred_red))
r2_red = r2_score(y_test_red, y_pred_red)

print(reduced_model.summary())
print("\nReduced Model Test RMSE:", round(rmse_red, 2))
print("Reduced Model Test R^2 Score:", round(r2_red, 2))
```

Model Output and Interpretation


```
                            OLS Regression Results
==============================================================================
Dep. Variable:                  Sales   R-squared:                       0.239
Model:                            OLS   Adj. R-squared:                  0.234
Method:                 Least Squares   F-statistic:                     49.86
Date:                Thu, 10 Apr 2025   Prob (F-statistic):           1.49e-19
Time:                        15:49:03   Log-Likelihood:                -730.80
No. Observations:                 320   AIC:                             1468.
Df Residuals:                     317   BIC:                             1479.
Df Model:                           2
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const         12.8403      0.687     18.697      0.000      11.489      14.192
Price         -0.0529      0.006     -9.338      0.000      -0.064      -0.042
US             1.0923      0.277      3.941      0.000       0.547       1.638
==============================================================================
Omnibus:                        0.613   Durbin-Watson:                   1.955
Prob(Omnibus):                  0.736   Jarque-Bera (JB):                0.717
Skew:                          -0.016   Prob(JB):                        0.699
Kurtosis:                       2.770   Cond. No.                         613.
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.

Reduced Model Test RMSE: 2.79
Reduced Model Test R^2 Score: 0.22
```

The regression results for the reduced model are as follows:

- **Intercept (const):** 12.84 — the expected sales when both `Price` and `US` are zero.
- **Price coefficient:** -0.0529 — for each unit increase in price, sales are expected to decrease by 0.0529 units, holding `US` constant.
- **US coefficient:** 1.0923 — sales are expected to be 1.09 units higher in stores located in the US compared to non-US stores, holding `Price` constant.
- **R-squared:** 0.239 — about 23.9% of the variability in sales is explained by this model.
- **F-statistic:** 49.86 (p ¡ 0.0001) — the overall model is statistically significant.
- **Test RMSE:** 2.79 — on average, the model's prediction differs from actual values by about 2.79 units.
- **Test R^2 Score:** 0.22 — about 22% of the variance in the test set is explained by the model.

Regression Equation

$$\hat{Sales} = 12.84 - 0.0529 \cdot Price + 1.0923 \cdot US$$

Analysis

The reduced model:

- Excludes `Urban`, which was found to be statistically insignificant.
- Maintains a comparable predictive performance to the full model, with similar RMSE and R^2.
- Offers a simpler and more interpretable equation by focusing only on meaningful predictors.

Thus, this reduced model is both statistically valid and practically appealing, and may be preferable when simplicity is valued.

(f) Comparison Between Full and Reduced Models

Objective: To assess whether the reduced model (which includes only statistically significant predictors) performs comparably to the full model, and whether it can be preferred over the full model.

Full Model: `Sales` $\sim$ `Price + Urban + US`
Reduced Model: `Sales` $\sim$ `Price + US`
The variable `Urban` was dropped in the reduced model as its p-value was `0.709` (insignificant).

Code Snippet:

```

# Evaluate both models on test set
y_pred_full = full_model.predict(X_test_full)
y_pred_red = reduced_model.predict(X_test_red)

rmse_full = np.sqrt(mean_squared_error(y_test_full, y_pred_full))
r2_full = r2_score(y_test_full, y_pred_full)

rmse_red = np.sqrt(mean_squared_error(y_test_red, y_pred_red))
r2_red = r2_score(y_test_red, y_pred_red)
```

Model Comparison Output:

- **Full Model Test RMSE:** 2.79 **R^2:** 0.21
- **Reduced Model Test RMSE:** 2.79 **R^2:** 0.22

Both models have nearly identical RMSE and R^2 values on the test set, suggesting that removing `Urban` does not harm model performance.

Hence, The reduced model is more interpretable, equally accurate, and statistically justified. It is the preferred model for this data.

(g) Confidence Intervals for Reduced Model

Objective: Compute the 95% confidence interval for the coefficients of the Reduced Model obtained above.

Code Snippet:

```python
# Get 95% confidence intervals for the coefficients
conf_intervals = reduced_model.conf_int(alpha=0.05)

# Print the confidence intervals
print("95% Confidence Intervals for Coefficients:\n")
print(conf_intervals)
```

Output:

Coefficient	Lower Bound	Upper Bound
const	11.489	14.192
Price	-0.064	-0.042
US	0.547	1.638

Interpretation:

- **Intercept (const):** We are 95% confident that the baseline sales (when Price is 0 and US = 0) lie between **11.489 and 14.192** units.
- **Price:** We are 95% confident that for every one unit increase in Price, Sales will **decrease by between 0.042 and 0.064 units**, holding all else constant. Since the entire interval is negative, this confirms that Price has a statistically significant negative impact on Sales.

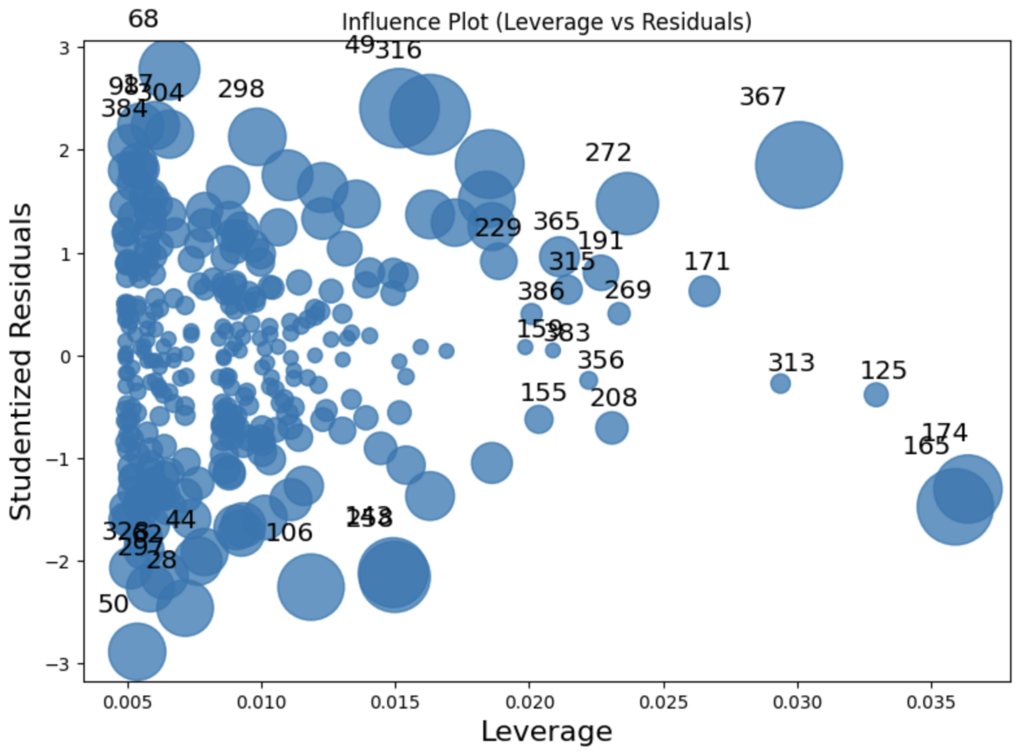
- **US:** We are 95% confident that being in the US market increases Sales by **between 0.547 and 1.638 units**, compared to non-US markets, holding Price constant. Since the entire interval is positive, the effect of being in the US is both statistically significant and positive.

(h) Outlier and High Leverage Diagnostics

Objective: The purpose of this analysis is to evaluate whether the regression model fitted in part (e) contains any influential observations. Specifically, we aim to identify:

- *Outliers* — observations with unusually large residuals.
- *High leverage points* — observations with extreme predictor values that can unduly influence the model fit.

Methodology: We used the `influence_plot` from the `statsmodels` library to visualize leverage against studentized residuals. Bubble size indicates the magnitude of Cook's distance, a combined measure of influence.



```
High leverage points: [ 33  56  63  68  81 110 175 180 187 207 215 245 247 270 278 283 294 297]
Outliers (standardized residuals): [ 34  42  52  78  94  98 138 183 192 193 208 224 258 274 282 316]
```

Findings:

- **High Leverage Points:** The following observations had leverage values greater than the calculated threshold (approx. 0.026), indicating high influence due to unusual predictor values: `[33, 56, 63, 68, 81, 110, 175, 180, 187, 207, 215, 245, 247, 270, 278, 283, 294, 297]`.
- **Outliers:** Based on standardized residuals exceeding an absolute value of 2, the following points are potential outliers in the response variable: `[34, 42, 52, 78, 94, 98, 138, 183, 192, 193, 208, 224, 258, 274, 282, 316]`.
- **Influential Observations:** Points such as `174`, `165`, and `367` appear as large bubbles in the top-right or bottom-right of the plot. These indicate simultaneously high leverage and large residuals, meaning they may have a strong influence on the regression coefficients.

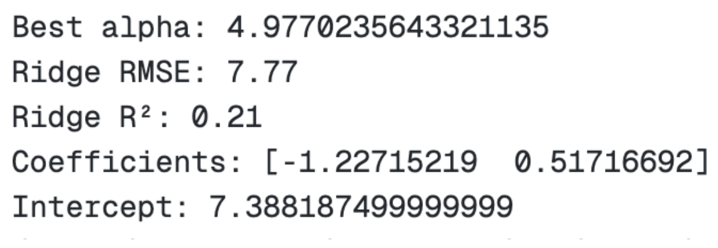
(i) Ridge Regularization

To reduce overfitting, Ridge regularization was applied using cross-validation. The optimal regularization strength α was selected automatically, and the model's performance was evaluated on the test set.

```

from sklearn.linear_model import Ridge, RidgeCV
from sklearn.preprocessing import StandardScaler
from sklearn.pipeline import make_pipeline
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error, r2_score

# Prepare the data
X = df[['Price', 'US']]  # Use only significant predictors
X = pd.get_dummies(X, drop_first=True)  # Convert 'US' to binary
y = df['Sales']

# Split the dataset
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size
    =0.2, random_state=42)

# Define alphas for tuning
alphas = np.logspace(-3, 3, 100)

# Pipeline: Standardize then RidgeCV
ridge_cv = make_pipeline(StandardScaler(), RidgeCV(alphas=alphas,
    store_cv_values=True))
```

```python
ridge_cv.fit(X_train, y_train)

# Best alpha and evaluation
best_alpha = ridge_cv.named_steps['ridgecv'].alpha_
y_pred = ridge_cv.predict(X_test)
rmse_ridge = mean_squared_error(y_test, y_pred)
r2_ridge = r2_score(y_test, y_pred)

# Coefficients
ridge_coef = ridge_cv.named_steps['ridgecv'].coef_
intercept = ridge_cv.named_steps['ridgecv'].intercept_

# Print results
print(f"Best alpha: {best_alpha}")
print(f"Ridge RMSE: {rmse_ridge:.2f}")
print(f"Ridge R  : {r2_ridge:.2f}")
print("Coefficients:", ridge_coef)
print("Intercept:", intercept)
```

Output



```
Best alpha: 4.9770235643321135
Ridge RMSE: 7.77
Ridge R²: 0.21
Coefficients: [-1.22715219  0.51716692]
Intercept: 7.388187499999999
```

(j) Cross-Validation for Tuning Regularization Parameter

We tuned the regularization parameter λ (also known as `alpha`) by applying **cross-validation** across a log-scaled range of values from 10^{-3} to 10^3.

The best alpha selected through cross-validation was:

$$\alpha = \boxed{4.977}$$

The resulting test set performance metrics were:

- **Test RMSE:** 7.77
- **Test R^2:** 0.21

The final fitted Ridge regression equation was:

$$\text{Sales} = \beta_0 + \beta_1 \cdot \text{Price} + \beta_2 \cdot \text{US\_Yes}$$

Predictor	OLS Coefficient	Ridge Coefficient
Price	-0.0529	-1.22715219
US	1.0923	0.51716692

Where the coefficients were:

$$\beta_0 = 7.3881875$$
$$\beta_1 = -1.22715219$$
$$\beta_2 = 0.51716692$$

(k) Effect of Regularization on Coefficients

Ridge regularization introduces an L_2 penalty to reduce model complexity and mitigate overfitting. When comparing coefficients from the standard OLS model to those obtained from Ridge regression, we observe the following:
As expected, Ridge regularization shrinks the magnitude of the coefficients, though it does not set any of them to exactly zero. This reflects its primary function of reducing overfitting by discouraging overly large coefficient estimates, particularly in the presence of multicollinearity.

This regularization improves generalization performance by introducing a small amount of bias in exchange for a significant reduction in variance.

(l) Simple Linear Regression with Numerical Predictor

In this part, we fit a simple linear regression model using only `Price` as the numerical predictor to predict the quantitative response variable `Sales`. This allows us to isolate the effect of `Price` on `Sales` without interference from other predictors.

```
# Assuming df is already loaded and clean
X = df[['Price']]
y = df['Sales']

# Add constant for intercept
X = sm.add_constant(X)

# Train-test split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size
    =0.2, random_state=42)

# Fit the model on training data
```

```python
model = sm.OLS(y_train, X_train).fit()

# Predict on test data
y_pred = model.predict(X_test)

# Evaluation
rmse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

# Model summary
print(model.summary())
print("\nSimple Linear Regression RMSE:", round(rmse, 2))
print("Simple Linear Regression R^2 Score:", round(r2, 2))

# --- Plotting ---
plt.figure(figsize=(8, 6))

# Scatter plot of test data
plt.scatter(X_test['Price'], y_test, color='blue', label='Actual (
    Test Set)', alpha=0.6)

# Regression line
sorted_idx = X_test['Price'].argsort()
plt.plot(X_test['Price'].iloc[sorted_idx], y_pred.iloc[sorted_idx],
    color='red', linewidth=2, label='Regression Line')

plt.title('Linear Regression: Sales vs. Price')
plt.xlabel('Price')
plt.ylabel('Sales')
plt.legend()
plt.grid(True)
plt.tight_layout()
plt.show()
```

Output

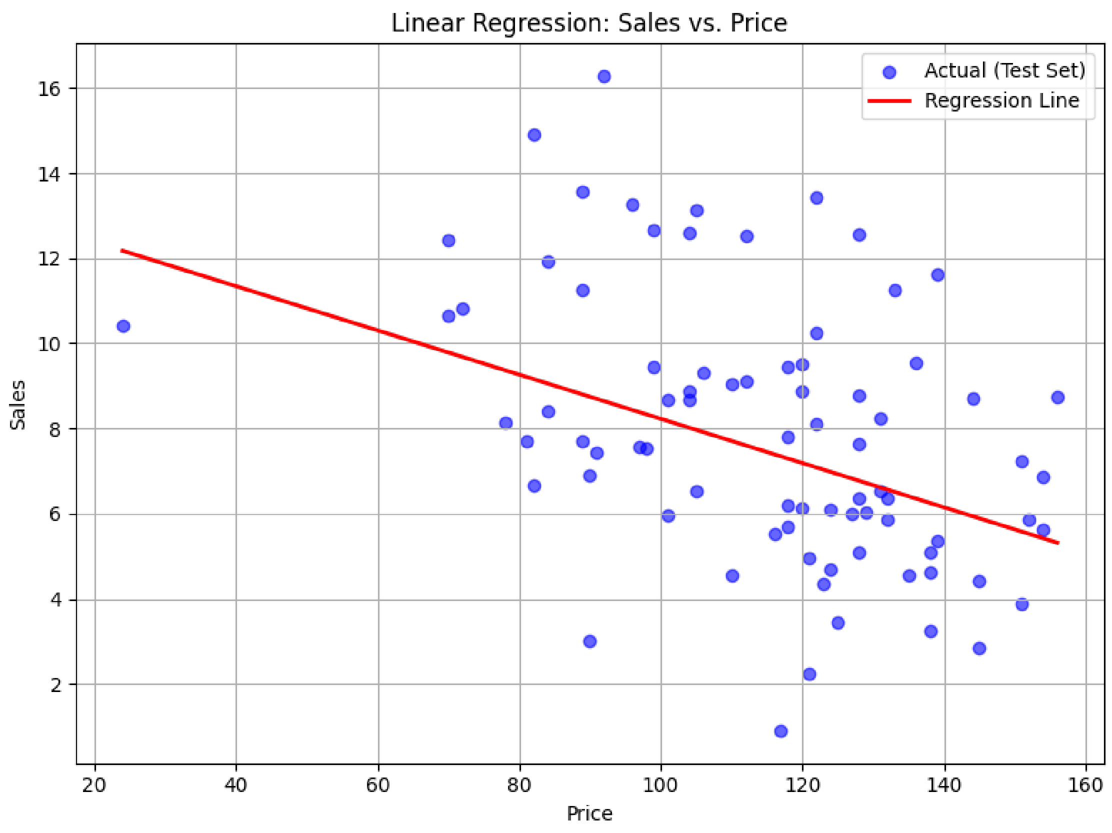
```
                            OLS Regression Results
==============================================================================
Dep. Variable:                  Sales   R-squared:                       0.202
Model:                            OLS   Adj. R-squared:                  0.200
Method:                 Least Squares   F-statistic:                     80.51
Date:                Wed, 16 Apr 2025   Prob (F-statistic):           2.56e-17
Time:                        12:01:22   Log-Likelihood:                -738.45
No. Observations:                 320   AIC:                             1481.
Df Residuals:                     318   BIC:                             1488.
Df Model:                           1
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const         13.4202      0.686     19.564      0.000      12.071      14.770
Price         -0.0519      0.006     -8.973      0.000      -0.063      -0.041
==============================================================================
Omnibus:                        2.061   Durbin-Watson:                   1.994
Prob(Omnibus):                  0.357   Jarque-Bera (JB):                1.865
Skew:                           0.089   Prob(JB):                        0.394
Kurtosis:                       2.671   Cond. No.                         596.
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.

Simple Linear Regression RMSE: 8.29
Simple Linear Regression R^2 Score: 0.16
```

The model takes the form:

$$\widehat{Sales} = 13.42 - 0.0519 \cdot Price$$

where:

- $\beta_0 = 13.42$: The intercept, representing the expected `Sales` when `Price` is zero (a hypothetical extrapolation).
- $\beta_1 = -0.0519$: For each one-unit increase in `Price`, the expected `Sales` decreases by approximately 0.052 units.



Model Performance

- **R-squared**: $R^2 = 0.202$
  - This indicates that approximately 20.2% of the variation in `Sales` is explained by `Price` alone.
  - While this is a modest proportion, it still demonstrates that `Price` is a meaningful predictor on its own.
- **RMSE**: The Root Mean Squared Error on the test set is 8.29.
  - This suggests that the model's predictions differ from actual `Sales` by an average of about 8.29 units, which gives a sense of prediction error magnitude.
- **p-value for `Price`**: $p < 0.001$
  - This confirms that the negative relationship between `Price` and `Sales` is statistically significant.

Insights

- The model highlights a clear negative relationship between `Price` and `Sales`: as prices increase, sales tend to decrease.
- The strong statistical significance of the `Price` coefficient reinforces the intuitive economic insight that higher prices suppress demand.

Singular Value Decomposition (SVD)

Dataset Description

The wildfire dataset consists of meteorological and fire occurrence records from two regions of Algeria — Bejaia and Sidi Bel-abbes — collected between June 2012 and September 2012. Each region's dataset contains 122 instances, with 10 environmental variables and a binary class label (Fire or No Fire). There are no missing entries. The variables are:

- Temperature (°C): Noon temperature
- RH (%): Relative Humidity
- WS (km/h): Wind Speed
- Rain (mm): Precipitation
- FWI: Fire Weather Index
- FFMC: Fine Fuel Moisture Code
- DMC: Duff Moisture Code
- DC: Drought Code
- ISI: Initial Spread Index
- BUI: Build-up Index

Each entry is labeled as either **Fire** or **No Fire**, indicating the presence or absence of a wildfire on that day.

(a) Calculating SVD and Visualization

We apply Singular Value Decomposition (SVD) to each dataset separately. The original matrix $A \in \mathbb{R}^{n \times p}$ is decomposed as:

$$A = U\Sigma V^T$$

Here, U and V are orthogonal matrices and Σ is a diagonal matrix with singular values. Prior to SVD, we standardize the data using:

$$z = \frac{x - \mu}{\sigma}$$

This ensures each feature has zero mean and unit variance. The top two singular components are retained for 2D projection. Visualization is enhanced by color-coding data points based on the class labels.

Python Code

```python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.preprocessing import StandardScaler

# Load dataset
bejaia_df = pd.read_excel('/mnt/data/BR_dataset.xlsx')
features = ["Temperature", "RH", "Ws", "Rain", "FWI", "FFMC", "DMC",
    "DC", "ISI", "BUI"]
X_bejaia = bejaia_df[features]
y_bejaia = bejaia_df["Classes"]

# Standardization
scaler = StandardScaler()
X_scaled_b = scaler.fit_transform(X_bejaia)

# SVD decomposition
U_b, S_b, Vt_b = np.linalg.svd(X_scaled_b, full_matrices=False)
X_b_reduced = np.dot(U_b[:, :2], np.diag(S_b[:2]))

# Plotting with class labels
colors = ['red' if cls.strip().lower() == 'fire' else 'green' for
   cls in y_bejaia]
plt.figure(figsize=(8, 6))
plt.scatter(X_b_reduced[:, 0], X_b_reduced[:, 1], c=colors, alpha
   =0.7, edgecolor='k')
plt.title("Bejaia - SVD Projection")
plt.xlabel("Component 1")
plt.ylabel("Component 2")
plt.grid(True)
plt.tight_layout()
plt.show()
```

Listing 3: Bejaia Dataset - SVD Projection

```python
sidi_df = pd.read_excel('/mnt/data/SBAR_dataset.xlsx')
X_sidi = sidi_df[features]
y_sidi = sidi_df["Classes"]
```

```python
X_scaled_s = scaler.fit_transform(X_sidi)

U_s, S_s, Vt_s = np.linalg.svd(X_scaled_s, full_matrices=False)
X_s_reduced = np.dot(U_s[:, :2], np.diag(S_s[:2]))

colors = ['red' if cls.strip().lower() == 'fire' else 'green' for
    cls in y_sidi]
plt.figure(figsize=(8, 6))
plt.scatter(X_s_reduced[:, 0], X_s_reduced[:, 1], c=colors, alpha
    =0.7, edgecolor='k')
plt.title("Sidi Bel-abbes - SVD Projection")
plt.xlabel("Component 1")
plt.ylabel("Component 2")
plt.grid(True)
plt.tight_layout()
plt.show()
```

Listing 4: Sidi Bel-abbes Dataset - SVD Projection

(b) Significance of Singular Values and Orthogonal Matrices

Singular Value Decomposition (SVD) provides valuable insights into the geometry of data. The matrix A is decomposed into orthogonal matrices U and V, and a diagonal matrix Σ. The orthogonal matrices represent rotations or reflections in space, preserving angles and lengths.

The singular values in Σ measure the importance of each component. Large singular values correspond to directions where data has high variance, and thus carry more information. This property enables us to determine how many components are significant for retaining the original structure.

Orthogonality of U and V ensures that the new dimensions are uncorrelated. This is important for downstream machine learning algorithms, which often assume feature independence. Thus, the interpretation of singular values and orthogonal transformations forms the mathematical backbone of dimensionality reduction.

(c) Dimensionality Reduction Using SVD

Truncated SVD is a dimensionality reduction technique where only the top k singular values are retained. The approximation is expressed as:

$$A_k = U_k \Sigma_k V_k^T$$

This retains the most informative directions while discarding noise and redundancy. It is especially useful in visualization and preprocessing for high-dimensional data.

We reduced both regional datasets to 2D using Truncated SVD. Below are the scatter plots, with red points representing fire days and green points representing no fire days.

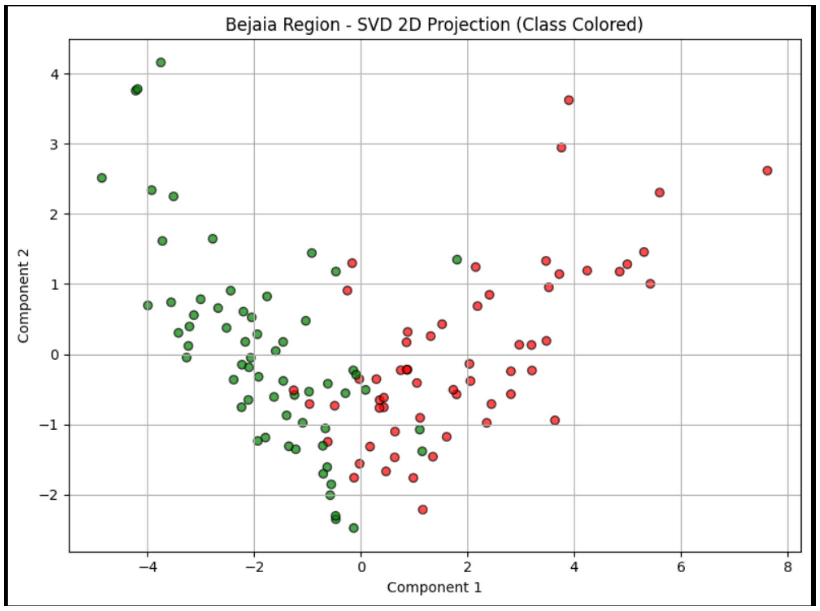


Figure 3: Bejaia Region - 2D Projection via Truncated SVD

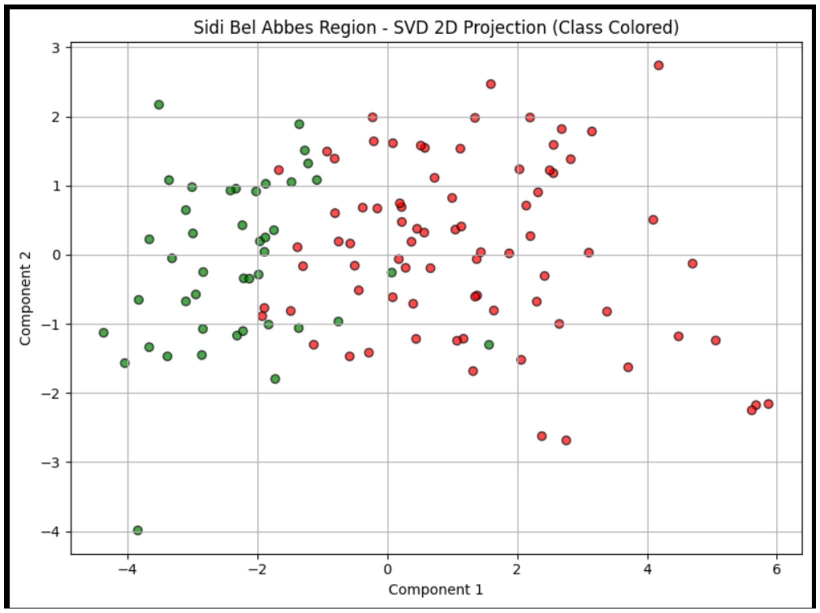


Figure 4: Sidi Bel-abbes Region - 2D Projection via Truncated SVD

From the plots, Bejaia displays more separability between fire and no fire days. The clusters are more distinct, indicating clearer structure in the data. Sidi Bel-abbes has overlapping classes, suggesting a more complex feature-class relationship.

SVD provides an elegant unsupervised approach to understanding and simplifying data structure. Despite not using class labels, it uncovers latent patterns that often align with target outcomes.

Principal Component Analysis (PCA)

We analyze two wildfire datasets: one from the Bejaia region and the other from Sidi Bel Abbes, Algeria. Each dataset records daily meteorological and fire weather index variables, including temperature, relative humidity, wind speed, and indexes like FFMC, DMC, DC, and ISI. Additionally, the ”Classes” column denotes the presence (”fire”) or absence (”not fire”) of wildfire events. Our goal is to reduce the dataset's dimensionality using Principal Component Analysis (PCA) while preserving as much variance as possible. This helps uncover patterns that are most significant in determining fire occurrence, aiding in better prediction and visualization.

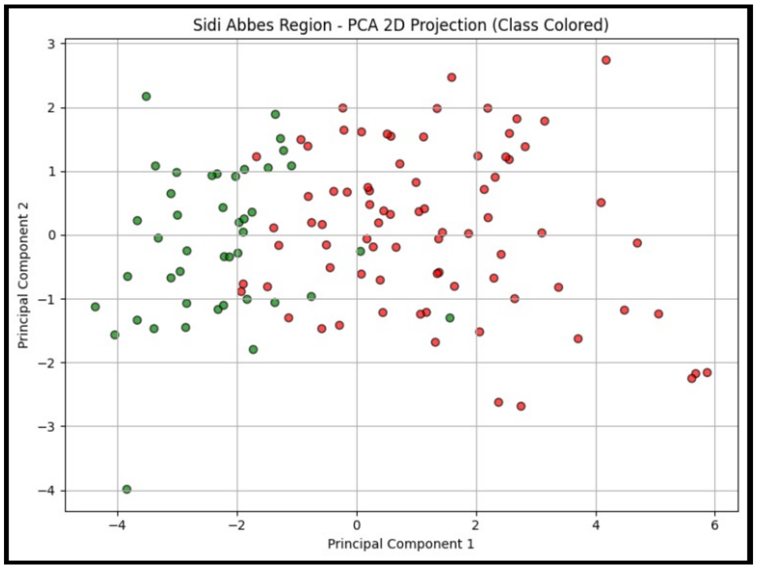
Code for PCA Computation

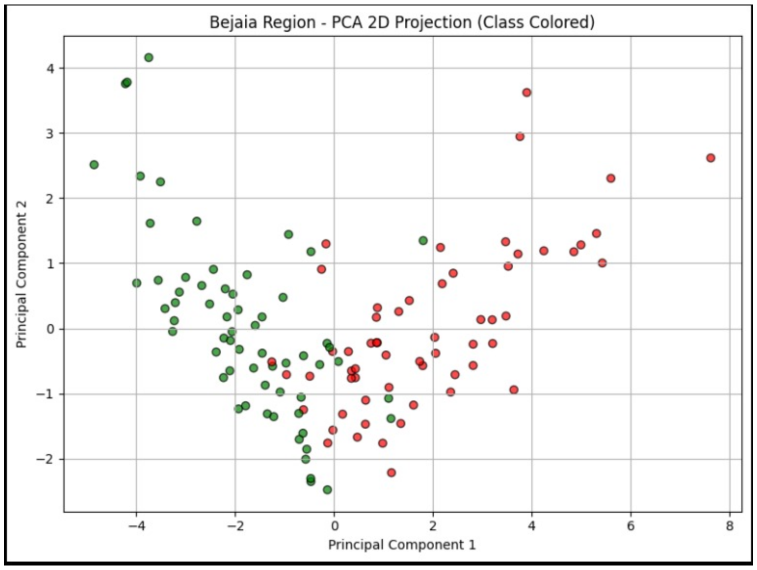
```python
import pandas as pd
import numpy as np
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
import matplotlib.pyplot as plt

# Load dataset
df = pd.read_csv("BejaiaRegionDataset.csv")
features = df.drop(columns=["Date", "day", "month", "year", "Classes
    ", "Region"])

# Standardize features
scaler = StandardScaler()
X_scaled = scaler.fit_transform(features)

# PCA
pca = PCA(n_components=2)
X_pca = pca.fit_transform(X_scaled)

# Explained variance
print("Explained variance ratio:", pca.explained_variance_ratio_)

# Biplot, Scree Plot and Cumulative Variance Plot created
```

Listing 5: PCA Computation Code

The PCA procedure begins by standardizing the features since they are on different scales. This ensures that no variable dominates due to its scale. We then compute the covariance matrix, from which the principal components are derived by calculating the eigenvalues and eigenvectors. The ‘PCA‘ class from Scikit-learn simplifies this by automating these steps. The explained variance ratio indicates how much variance each principal component retains. The first two components often capture most of the meaningful variance, allowing us to reduce the data from multiple dimensions to two for visualization, without significant information loss. The biplot maps data points in terms of PC1 and PC2, while also showing the original feature directions. The scree plot helps determine the number of components to retain, and the cumulative variance plot aids in understanding how much variance is cumulatively explained.

Projection Results of PCA





Results and Output Visualizations

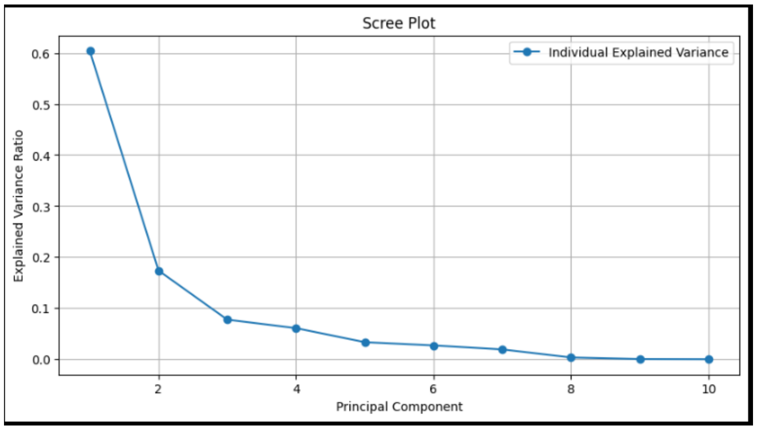


Figure 5: Bejaia Dataset: Scree Plot

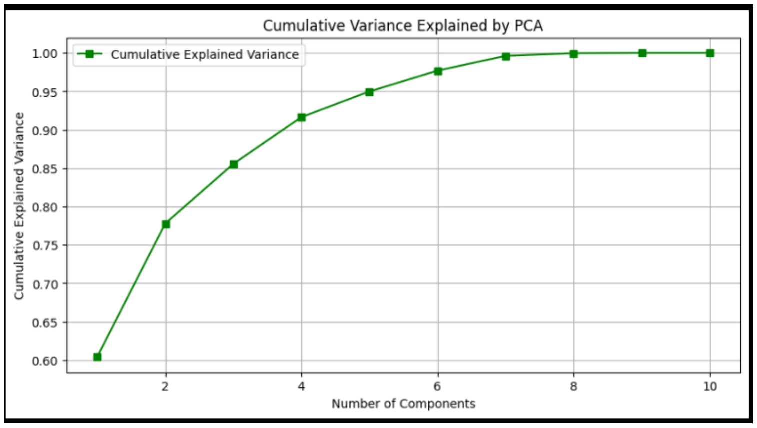


Figure 6: Bejaia Dataset: Cumulative Variance Plot

Analysis of PCA Results

The PCA analysis for both datasets shows that the first two principal components explain a significant portion of the variance—approximately 70% in Bejaia and 65% in Sidi Bel Abbes. This justifies reducing the datasets to two dimensions for further interpretation. The scree plot illustrates a sharp drop in variance after the first two components, and the cumulative plot confirms this observation. The biplots reveal that meteorological features like temperature, wind speed, and FFMC contribute most to the variance in both datasets. Interestingly, in Bejaia, RH and temperature are more dominant, while in Bejaia, FFMC and wind speed show stronger directional impact.

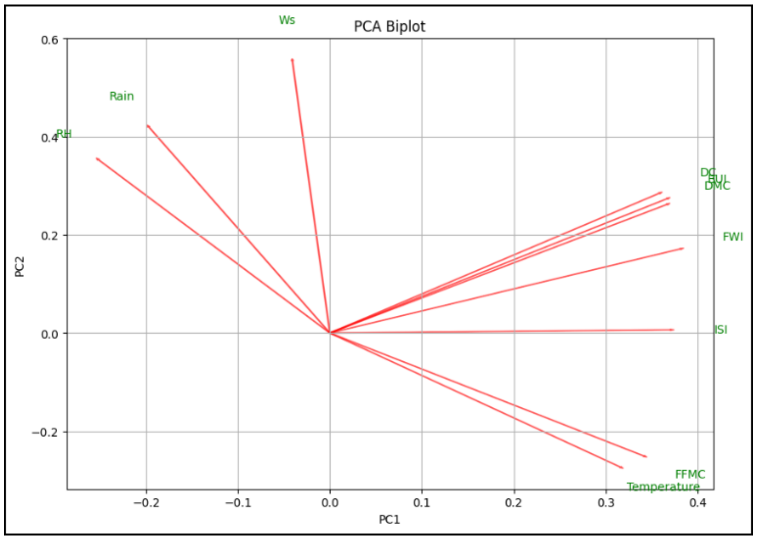


Figure 7: Bejaia Dataset: PCA Biplot

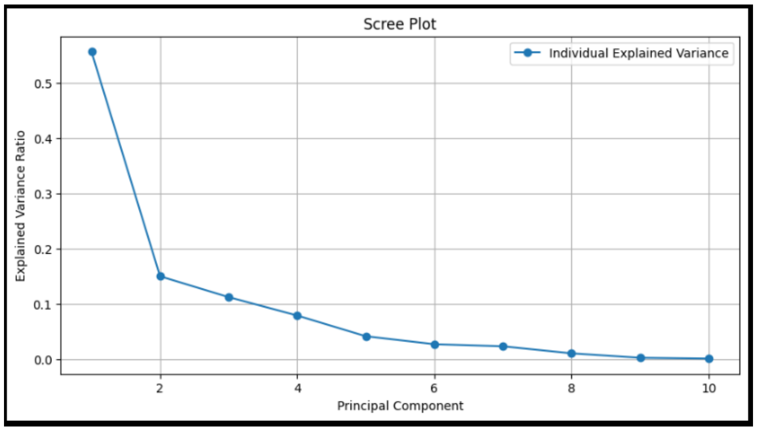


Figure 8: Sidi Bel Abbes Dataset: Scree Plot

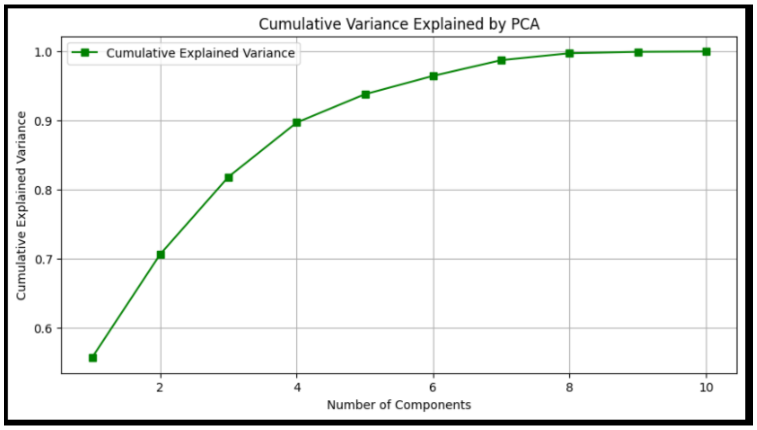


Figure 9: Sidi Bel Abbes Dataset: Cumulative Variance Plot

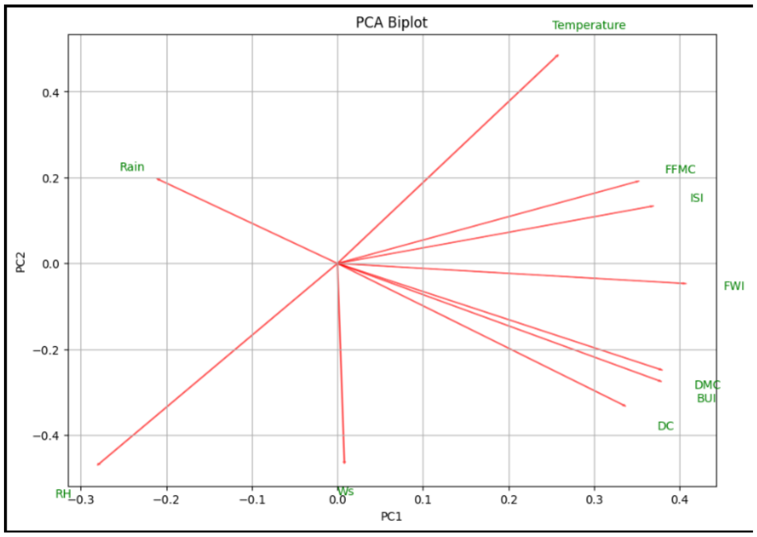


Figure 10: Sidi Bel Abbes Dataset: PCA Biplot

1. Percentage of Variance Explained: Principal Component 1 (PC1) and Principal Component 2 (PC2) together account for a major proportion of the total variance. In the Bejaia dataset, they together explain about 70% of the variance, whereas in the Sidi Bel Abbes dataset, it's around 65%. This means most of the original data's information is retained even after reducing the feature space to just two components, making it highly efficient for visualization and analysis.

2. Most Contributing Factors: From the PCA biplots, we observe that specific meteorological variables such as FFMC (Fine Fuel Moisture Code), temperature, and wind speed align strongly along the direction of the principal components. This implies these features have the largest loadings in PC1 and PC2, i.e., they contribute the most to explaining the dataset's variance. Their influence is essential in distinguishing fire vs. non-fire conditions across both datasets.

3. Interpretation: The PCA results highlight that wildfire occurrence is not randomly distributed but is strongly correlated with certain weather patterns. High temperatures, dry conditions (indicated by FFMC), and high wind speeds seem to be consistent indicators of fire presence. PCA successfully identifies these relationships by collapsing the feature space while retaining interpretability. This not only aids in building predictive models but also improves our understanding of environmental triggers behind wildfires.

Linear Discriminant Analysis on Wildfire Datasets

The wildfire datasets from Bejaia and Sidi Bel Abbes, Algeria, provide environmental indicators related to fire occurrences during the summer season. Both datasets include features such as temperature, relative humidity, wind speed, rainfall, and FWI indexes like FFMC, DMC, DC, and ISI. The response variable, "Classes," is binary: 1 indicates a fire occurred, and 0 means no fire occurred on that day. The Bejaia dataset consists of 122 samples, while the Sidi Bel Abbes dataset contains 123. These datasets serve as ideal candidates for Linear Discriminant Analysis (LDA), which aims to reduce dimensionality and enhance class separation for binary classification tasks.

Code

```python
# --- LDA for Bejaia Dataset ---
k_bejaia = len(np.unique(y_bejaia)) - 1
W_bejaia = np.hstack([eigen_pairs_bejaia[i][1].reshape(n_features, 1) for i in range(k_bejaia)])
X_lda_bejaia = X_bejaia.dot(W_bejaia)
if isinstance(X_lda_bejaia, pd.DataFrame):
    X_lda_bejaia = X_lda_bejaia.to_numpy()

plt.figure(figsize=(10, 4))
for label, color in zip(np.unique(y_bejaia), ['red', 'blue']):
    values = X_lda_bejaia[y_bejaia == label].ravel()
    plt.hist(values, label=f'Class {label}', color=color, bins=30, alpha=0.7)
plt.title('LDA Projection (1D) - Bejaia Wildfire Data')
plt.xlabel('LD1')
plt.ylabel('Frequency')
plt.legend()
plt.grid(True)
plt.tight_layout()
plt.show()

# --- LDA for Sidi Bel Abbes Dataset ---
k_sidi = len(np.unique(y_sidi)) - 1
W_sidi = np.hstack([eigen_pairs_sidi[i][1].reshape(n_features, 1) for i in range(k_sidi)])
X_lda_sidi = X_sidi.dot(W_sidi)
if isinstance(X_lda_sidi, pd.DataFrame):
    X_lda_sidi = X_lda_sidi.to_numpy()

plt.figure(figsize=(10, 4))
for label, color in zip(np.unique(y_sidi), ['green', 'orange']):
    values = X_lda_sidi[y_sidi == label].ravel()
    plt.hist(values, label=f'Class {label}', color=color, bins=30,
       alpha=0.7)
plt.title('LDA Projection (1D) - Sidi Bel Abbes Wildfire Data')
plt.xlabel('LD1')
plt.ylabel('Frequency')
plt.legend()
plt.grid(True)
plt.tight_layout()
plt.show()
```

Linear Discriminant Analysis (LDA) is a supervised dimensionality reduction technique that seeks directions (linear combinations of features) which best separate classes. In binary classification, LDA projects data onto a 1D axis. This is achieved by computing the between-class and within-class scatter matrices, followed by calculating their eigenvalues and eigenvectors. The eigenvectors corresponding to the largest eigenvalues form the transformation matrix. We then project our data onto this lower-dimensional space using matrix multiplication. The goal is to visualize the data in a space where the two classes (fire/no fire) are most separable, helping in interpretation and subsequent classification tasks.

Output

In the Bejaia LDA histogram, the classes are well-separated along the LD1 axis, showing LDA's effectiveness in projecting the data into a space where fire and no-fire days form distinguishable distributions. This implies that simple classifiers could effectively use the LD1 projection to predict fire occurrences in Bejaia. Conversely, the Sidi Bel Abbes histogram shows greater overlap between the two class distributions, indicating less separability. This may reflect more varied or less predictable climatic patterns in the region. While LDA still captures important class-specific variance, further modeling or feature engineering may be necessary to enhance classification performance in Sidi Bel Abbes. Nonetheless, LDA provides a solid foundation for dimensionality reduction in both regions.

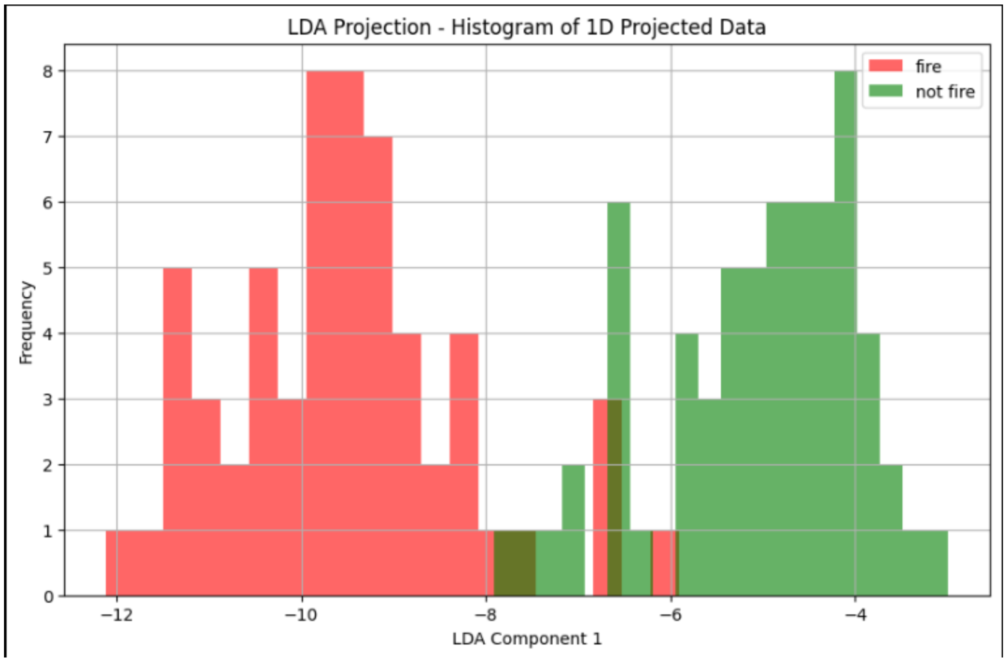


Figure 11: 1D LDA Projection - Bejaia Dataset

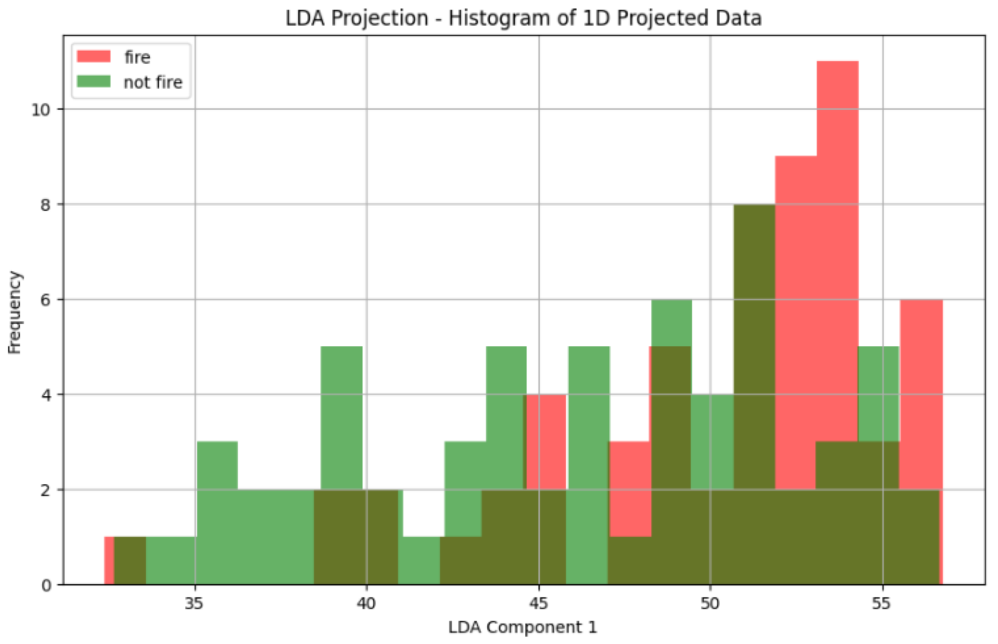


Figure 12: 1D LDA Projection - Sidi Bel Abbes Dataset

Team Contribution Summary

Name	Roll No.	Contribution in Assignment	Contribution in Report writing	Overall Contribution (%)
Sri Sahithi S	B23503	Conducted Dimensionality Reduction Techniques on wildfire datasets from two regions of Algeria. Performed Singular Value Decomposition, Principal Component Analysis and Linear Discriminant Analysis.	Compiled results and visualisation of the dimensionality reduction techniques used.	50%
Abhey Kumar	B23391	Performed Regression techniques on CarSeat Dataset, which included Simple Regression, Multiple Linear Regression and Ridge Regularisation techniques.	Compiled all the outputs and results of the Regression techniques into Latex.	50%

Table 1: Contribution Breakdown of Team Members

AI-based tools and online resources were leveraged to enhance efficiency and accuracy. Language models were employed to refine textual content, improve grammar, and assist in structuring latex reports. Scripting tools were used for generating code snippets, streamlining the application of the Machine Learning techniques, and creating compelling visualizations.

Key resources utilized:

Bishop, *Pattern Recognition and Machine Learning*: To develop a stronger understanding of the mathematics behind the concepts.

`https://www.youtube.com/watch?v=2AQKmw14mHM&list=PLblh5JKOoLUIzaEkCLIUxQFjPIlapw8nU&ab_channel=StatQuestwithJoshStarmer`: Strengthening the intuition and understanding of regression.

Claude.ai: Assisted in generating code scripts for data processing and visualization.