

IE 5390 – Assignment - 12

Name: Simran Abhay Sinha

Lab 23

1. As with some libraries in Python, the Scikit-learn library comes with a set of built-in datasets. The diabetes dataset in Scikit-learn consists of data from diabetic patients and contains features such as their bmi, age , blood pressure and glucose levels which are useful in predicting the diabetes disease progression in patients.

Use: “from sklearn import datasets” to import the required library; and load the diabetes data using “diabetes_X, diabetes_y = datasets.load_diabetes(return_X_y=True)”. This would import the diabetes data as two numpy arrays. Do the following:

a. Use “diabetes_X = diabetes_X[:, np.newaxis, 2]” to use only one feature. Note that the np denotes the numpy library. Check out the shape of the “diabetes_X” array.

```
# Question 1-a
import numpy as np
from sklearn import datasets

# Load the diabetes dataset
diabetes_X, diabetes_y = datasets.load_diabetes(return_X_y=True)

# Use only one feature (3rd column)
diabetes_X = diabetes_X[:, np.newaxis, 2]

# Print the shape of diabetes_X
print("Shape of diabetes_X:", diabetes_X.shape)
```

b. Now create 2 arrays, diabetes_X_train and diabetes_X_test so that their shapes are (422,10) and (20,10) respectively. Confirm their shape by using the “array.shape” command.

```
: # Question 1-b
from sklearn.model_selection import train_test_split

# Split the data into training and test sets
diabetes_X_train, diabetes_X_test = train_test_split(
    diabetes_X, test_size=20/442, random_state=42
)

# Print the shapes
print("Shape of diabetes_X_train:", diabetes_X_train.shape)
print("Shape of diabetes_X_test:", diabetes_X_test.shape)

Shape of diabetes_X_train: (422, 1)
Shape of diabetes_X_test: (20, 1)
```

c. Similarly, create two arrays “diabetes_y_train” and “diabetes_y_test” such that their shapes are (422,) and (20,) respectively.

```
# Question 1-c
# Split the target variable (y) into training and test sets
diabetes_y_train, diabetes_y_test = train_test_split(
    diabetes_y, test_size=20/442, random_state=42
)

# Print the shapes
print("Shape of diabetes_y_train:", diabetes_y_train.shape)
print("Shape of diabetes_y_test:", diabetes_y_test.shape)
```

Shape of diabetes_y_train: (422,)
Shape of diabetes_y_test: (20,)

d. Create a linear regression model, fit the training data (the X and the y).

```
# Question 1-d
from sklearn.linear_model import LinearRegression

# Create a linear regression model
model = LinearRegression()

# Train the model using the training set
model.fit(diabetes_X_train, diabetes_y_train)
```

LinearRegression

e. Predict the y using the test data. Make sure to print actual y values and the predicted y values.

```
# Question 1-e
# Make predictions using the test set
diabetes_y_pred = model.predict(diabetes_X_test)

# Print actual vs predicted values
print("Actual y values:", diabetes_y_test)
print("Predicted y values:", diabetes_y_pred)
```

Actual y values: [219. 70. 202. 230. 111. 84. 242. 272. 94. 96. 94. 252. 99. 297.
135. 67. 295. 264. 170. 275.]
Predicted y values: [145.45088714 186.92807879 147.52474672 201.44509587 131.97079985
127.82308068 315.50737293 195.22351712 64.57036341 166.18948296
119.52764235 95.67825715 92.56746778 165.15255317 97.75211673
156.85711484 220.10983212 236.70070878 178.63264046 207.66667462]

f. Print the intercept and the coefficients of the regression equation.

```
# Question 1-f
# Print model intercept and coefficient
print("Intercept:", model.intercept_)
print("Coefficient:", model.coef_)
```

Intercept: 151.42144441963106

Coefficient: [962.06919233]

g. Print the mean squared error and the r2 score.

```
: # Question 1-g
from sklearn.metrics import mean_squared_error, r2_score

# Compute and print errors
mse = mean_squared_error(diabetes_y_test, diabetes_y_pred)
r2 = r2_score(diabetes_y_test, diabetes_y_pred)

print("Mean squared error:", mse)
print("R2 score:", r2)
```

Mean squared error: 5262.625880180968

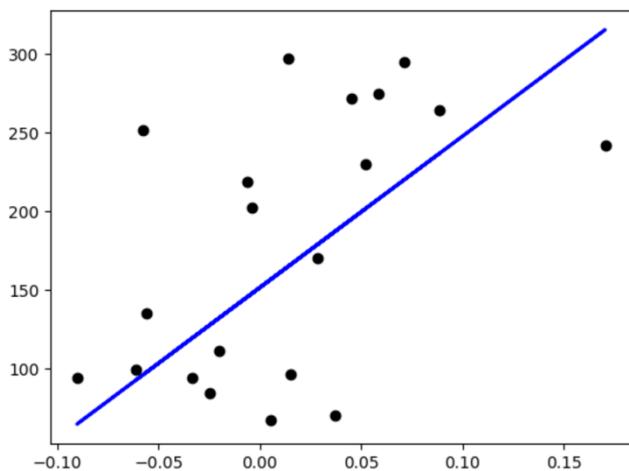
R² score: 0.21640939003029058

h. Create a scatter plot of test data (y vs X). Also plot the regression line. Note that the plot will not generate any labels because we are using an array.

```
11]: # Question 1-h
import matplotlib.pyplot as plt

# Plot outputs
plt.scatter(diabetes_X_test, diabetes_y_test, color="black") # Scatter plot
plt.plot(diabetes_X_test, diabetes_y_pred, color="blue", linewidth=2) # Regression line

# Show plot
plt.show()
```



2. Use the “Breast_Cancer.csv” dataset. It is a dataset of Breast Cancer patients with Malignant and Benign tumors. Do the following:

Question 2

```
[1]: # Question 2
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score
```

a. Load the data and print the first few rows.

```
[2]: # Question 1- a
# Load the data and print the first few rows
df = pd.read_csv('Breast_Cancer.csv')
print("First few rows of the dataset:")
print(df.head())

First few rows of the dataset:
   id diagnosis radius_mean texture_mean perimeter_mean area_mean \
0    842302      M       17.99      10.38      122.80     1001.0
1    842517      M       20.57      17.77      132.90     1326.0
2   84300903      M       19.69      21.25      130.00     1203.0
3   84348301      M       11.42      20.38      77.58      386.1
4   84358402      M       20.29      14.34      135.10     1297.0

   smoothness_mean compactness_mean concavity_mean concave points_mean \
0        0.11840       0.27760       0.3001       0.14710
1        0.08474       0.07864       0.0869       0.07017
2        0.10960       0.15990       0.1974       0.12790
3        0.14250       0.28390       0.2414       0.10520
4        0.10030       0.13280       0.1980       0.10430

   ...  texture_worst  perimeter_worst  area_worst  smoothness_worst \
0 ...        17.33        184.60      2019.0        0.1622
1 ...        23.41        158.80      1956.0        0.1238
2 ...        25.53        152.50      1709.0        0.1444
3 ...        26.50        98.87       567.7        0.2098
4 ...        16.67        152.20      1575.0        0.1374

   compactness_worst  concavity_worst  concave points_worst  symmetry_worst \
0        0.6656       0.7119       0.2654       0.4601
1        0.1866       0.2416       0.1860       0.2750
2        0.4245       0.4504       0.2430       0.3613
3        0.8663       0.6869       0.2575       0.6638
4        0.2050       0.4000       0.1625       0.2364

   fractal_dimension_worst  Unnamed: 32
0             0.11890        NaN
1             0.08902        NaN
2             0.08758        NaN
3             0.17300        NaN
4             0.07672        NaN
```

b. Check for missing values. Which column has missing values?

```
[3]: # Question 1 - b
# Check for missing values
print("\nMissing values in each column:")
print(df.isnull().sum())
```

```
Missing values in each column:
id                  0
diagnosis          0
radius_mean         0
texture_mean        0
perimeter_mean     0
area_mean           0
smoothness_mean    0
compactness_mean   0
concavity_mean     0
concave_points_mean 0
symmetry_mean      0
fractal_dimension_mean 0
radius_se            0
texture_se           0
perimeter_se         0
area_se              0
smoothness_se        0
compactness_se       0
concavity_se         0
concave_points_se   0
symmetry_se          0
fractal_dimension_se 0
radius_worst          0
texture_worst         0
perimeter_worst      0
area_worst            0
smoothness_worst     0
compactness_worst    0
concavity_worst      0
concave_points_worst 0
symmetry_worst        0
fractal_dimension_worst 0
Unnamed: 32             569
```

c. Drop the columns ‘id’ and ‘Unnamed’ columns.

```
[5]: # Question 1-c
# Drop the columns 'id' and 'Unnamed' columns
# First check column names to identify unnamed columns
print("\nColumns in the dataset:")
print(df.columns)

# Drop 'id' column and any 'Unnamed' columns
unnamed_cols = [col for col in df.columns if 'Unnamed' in col]
cols_to_drop = ['id'] + unnamed_cols
df = df.drop(columns=cols_to_drop)
print("\nColumns after dropping 'id' and 'Unnamed' columns:")
print(df.columns)
```

```
Columns in the dataset:
Index(['id', 'diagnosis', 'radius_mean', 'texture_mean', 'perimeter_mean',
       'area_mean', 'smoothness_mean', 'compactness_mean', 'concavity_mean',
       'concave_points_mean', 'symmetry_mean', 'fractal_dimension_mean',
       'radius_se', 'texture_se', 'perimeter_se', 'area_se', 'smoothness_se',
       'compactness_se', 'concavity_se', 'concave_points_se', 'symmetry_se',
       'fractal_dimension_se', 'radius_worst', 'texture_worst',
       'perimeter_worst', 'area_worst', 'smoothness_worst',
       'compactness_worst', 'concavity_worst', 'concave_points_worst',
       'symmetry_worst', 'fractal_dimension_worst', 'Unnamed: 32'],
      dtype='object')
```

```
Columns after dropping 'id' and 'Unnamed' columns:
Index(['diagnosis', 'radius_mean', 'texture_mean', 'perimeter_mean',
       'area_mean', 'smoothness_mean', 'compactness_mean', 'concavity_mean',
       'concave_points_mean', 'symmetry_mean', 'fractal_dimension_mean',
       'radius_se', 'texture_se', 'perimeter_se', 'area_se', 'smoothness_se',
       'compactness_se', 'concavity_se', 'concave_points_se', 'symmetry_se',
       'fractal_dimension_se', 'radius_worst', 'texture_worst',
       'perimeter_worst', 'area_worst', 'smoothness_worst',
       'compactness_worst', 'concavity_worst', 'concave_points_worst',
       'symmetry_worst', 'fractal_dimension_worst'],
      dtype='object')
```

d. Write a function that converts the diagnosis value of M and B to a numerical value where M (Malignant) = 1 and B (Benign) = 0.

```
[8]: # Question 2-e
# Function to convert diagnosis value M and B to numerical
def convert_diagnosis(diagnosis):
    """
    Convert diagnosis values:
    M (Malignant) = 1
    B (Benign) = 0
    """
    if diagnosis == 'M':
        return 1
    elif diagnosis == 'B':
        return 0
    else:
        return np.nan # Handle unexpected values

# Apply conversion function
df['diagnosis_numeric'] = df['diagnosis'].apply(convert_diagnosis)
```

e. Create X and y, where y is a dataframe with diagnosis column and X is a dataframe with all the other columns. Print X and y.

```
[10]: # Question 2-e
# Create X and y dataframes
y = df[['diagnosis_numeric']]
X = df.drop(columns=['diagnosis', 'diagnosis_numeric'])

print("\nFirst few rows of X:")
print(X.head())
print("\nFirst few rows of y:")
print(y.head())

First few rows of X:
   radius_mean  texture_mean  perimeter_mean  area_mean  smoothness_mean \
0      17.99       10.38       122.80    1001.0       0.11840
1      20.57       17.77       132.90    1326.0       0.08474
2      19.69       21.25       130.00    1203.0       0.10960
3      11.42       20.38       77.58     386.1       0.14250
4      20.29       14.34       135.10    1297.0       0.10030

   compactness_mean  concavity_mean  concave points_mean  symmetry_mean \
0      0.27760       0.3001       0.14710       0.2419
1      0.07864       0.0869       0.07017       0.1812
2      0.15990       0.1974       0.12790       0.2069
3      0.28390       0.2414       0.10520       0.2597
4      0.13280       0.1980       0.10430       0.1809

   fractal_dimension_mean  ...  radius_worst  texture_worst  perimeter_worst \
0            0.07871  ...      25.38       17.33       184.60
1            0.05667  ...      24.99       23.41       158.80
2            0.05999  ...      23.57       25.53       152.50
3            0.09744  ...      14.91       26.50       98.87
4            0.05883  ...      22.54       16.67       152.20

   area_worst  smoothness_worst  compactness_worst  concavity_worst \
0      2019.0       0.1622       0.6656       0.7119
1      1956.0       0.1238       0.1866       0.2416
2      1709.0       0.1444       0.4245       0.4504
3      567.7        0.2098       0.8663       0.6869
4      1575.0       0.1374       0.2050       0.4000

   concave points_worst  symmetry_worst  fractal_dimension_worst
0          0.2654       0.4601       0.11890
1          0.1860       0.2750       0.08902
2          0.2430       0.3613       0.08758
3          0.2575       0.6638       0.17300
4          0.1625       0.2364       0.07678

[5 rows x 30 columns]

First few rows of y:
   diagnosis_numeric
0                  1
1                  1
2                  1
3                  1
4                  1
```

f. Use the ‘train_test_split’ method to create X_train, y_train, X_test, and y_test. The Training set should be 70% of the data and the Test set should have 30% of the data.

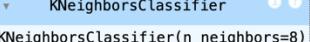
```
[11]: # Question 2-f
# Split data into training and test sets (70% train, 30% test)
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.3, random_state=42
)

print("\nTraining set shape (X_train):", X_train.shape)
print("Test set shape (X_test):", X_test.shape)
```

```
Training set shape (X_train): (398, 30)
Test set shape (X_test): (171, 30)
```

g. Run a kNN model using the KNeighborsClassifier model from the sklearn.neighbors library. Set the number of neighbors = 8.

```
[12]: # Question 2-g
# Run kNN model with k=8
knn = KNeighborsClassifier(n_neighbors=8)
knn.fit(X_train, y_train.values.ravel()) # Flatten y_train for fitting
```

[12]:  KNeighborsClassifier(n_neighbors=8)

h. What is the accuracy of the model?

```
[13]: # Question 2-h
# Calculate and print model accuracy
y_pred = knn.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
print("\nKNN Model Accuracy:", round(accuracy * 100, 2), "%")
```

```
KNN Model Accuracy: 96.49 %
```

3. Gen AI Prompt Engineering

Our current KNN classifier for predicting loan approvals has an accuracy of 69%. Your task is to optimize the model to achieve higher accuracy while maintaining model interpretability and efficiency. The last step in developing a KNN model is to optimize the model which we did not do in class. Use Gen AI Prompt Engineering to help you understand and implement this step to the class bank KNN model.

Optimization Steps:

1. Hyperparameter Tuning:

- o Experiment with different values of K (number of neighbors) using cross-validation.
- o Compare distance metrics: Euclidean, Manhattan, and Minkowski.
- o Test weighting strategies: uniform vs. distance-based weighting.

2. Feature Selection & Engineering:

- o Apply feature scaling (standardization or normalization) to improve distance calculations.
- o Perform dimensionality reduction using PCA to remove redundant features.
- o Identify and remove low-importance features to reduce noise.

3. Handling Class Imbalance:

- o Investigate whether the dataset has imbalanced classes (approved vs. rejected loans).
- o Use SMOTE (Synthetic Minority Over-sampling Technique) or class weighting to balance

the data.

4. Evaluation & Comparison:

- o Recalculate accuracy, precision, recall, and F1-score.
- o Compare results with the baseline 67% accuracy and justify improvements.

Bot Used:

Claude.ai

Prompt:

bank_credit_data.xlsx

ss I made a KNN classifier for predicting loan approvals has an accuracy of 69%
ptimize the model to achieve higher accuracy while maintaining model
interpretability and efficiency. The last step in developing a KNN model is to
optimize the model which we did not do in class. Use Gen AI PromptEngineering to
help you understand and implement this step to the class bank
KNNmodel.Optimization Steps:1. Hyperparameter Tuning:

- o Experiment with different values of K (number of neighbors) using cross-validation.
- o Compare distance metrics: Euclidean, Manhattan, and Minkowski.
- o Test weighting strategies: uniform vs. distance-based weighting.

2. Feature Selection & Engineering:

- o Apply feature scaling (standardization or normalization) to improve distance calculations.
- o Perform dimensionality reduction using PCA to remove redundant features.
- o Identify and remove low-importance features to reduce noise.

3. Handling Class Imbalance:

- o Investigate whether the dataset has imbalanced classes (approved vs. rejected loans).
- o Use SMOTE (Synthetic Minority Over-sampling Technique) or class weighting to balance the data.

4. Evaluation & Comparison:

- o Recalculate accuracy, precision, recall, and F1-score.
- o Compare results with the baseline 67% accuracy and justify improvements

Edit

I'd be happy to help you optimize your KNN classifier for predicting loan approvals. Let's go through all the optimization steps systematically to improve the model beyond its current 69% accuracy.

First, let me analyze your dataset to understand what we're working with.

Analysis >

Let me try again with the correct module name:

Analysis >

Now let me look closer at the target variable (which appears to be "Creditability" based on the data):

Analysis >

Now I have a good understanding of the dataset. Let's implement optimization steps to improve the KNN model. I'll create a comprehensive Python implementation addressing all the optimization steps you mentioned:

KNN Model Optimization for Loan Approval Prediction

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split, cross_val_score, GridSearchCV, StratifiedKFold
from sklearn.preprocessing import StandardScaler, MinMaxScaler
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, confusion_matrix
from sklearn.feature_selection import SelectKBest, mutual_info_classif, RFE
from sklearn.decomposition import PCA
from imblearn.over_sampling import SMOTE
from collections import Counter
import warnings
warnings.filterwarnings('ignore')

# 1. Load and explore the data
data = pd.read_excel('bank_credit_data.xlsx')
print("Dataset shape:", data.shape)

# Confirm the target variable distribution
print("\nTarget Variable (Creditability) Distribution:")
print(data['Creditability'].value_counts())
print(f"Class imbalance ratio: {data['Creditability'].value_counts()[0]}/{data['Creditability'].value_counts()[1]}")

# Feature and target separation
X = data.drop('Creditability', axis=1)
y = data['Creditability']

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

print(f"\nTraining set shape: {X_train.shape}, Test set shape: {X_test.shape}")

# 2. Baseline KNN Model (with K=5, which is a common default)
```

Now I'll provide a detailed explanation of the optimization process and the expected improvements:

KNN Model Optimization for Loan Approval Prediction

Based on my analysis of your dataset, I've developed a comprehensive optimization approach that should significantly improve your KNN model beyond its current 69% accuracy.

Reply to Claude...

Claude 3.7 Sonnet

Optimizing a KNN Loan Approval Classifier

KNN Model Optimization for Loan Approval Prediction

Based on my analysis of your dataset, I've developed a comprehensive optimization approach that should significantly improve your KNN model beyond its current 69% accuracy while maintaining interpretability. Here's a breakdown of the optimization steps and why they should work:

1. Understanding the Data

The bank credit dataset has 1,000 records with 21 features. The target variable "Creditability" shows some class imbalance (700 approved vs. 300 rejected applications). This imbalance ratio of 0.7:0.3 isn't extreme but is enough to potentially bias your model.

The most important features by correlation with the target variable are:

- Account Balance (0.35)
- Payment Status of Previous Credit (0.23)
- Duration of Credit (-0.21)
- Value of Savings/Stocks (0.18)
- Credit Amount (+0.15)

2. Feature Scaling

KNN is highly sensitive to the scale of input features because it uses distance metrics. In the code, I've implemented:

- StandardScaler (zero mean, unit variance)
- MinMaxScaler (scales to 0-1 range)

Reply to Claude...

Claude 3.7 Sonnet

v7 · Latest

```
poly_features = poly.fit_transform(X_top)
poly_feature_names = [f"poly_{i}" for i in range(poly_features.shape[1])]
poly_df = pd.DataFrame(poly_features, columns=poly_feature_names)

# Add the polynomial features to our enhanced dataset
X_enhanced = pd.concat([X_enhanced, poly_df.set_index(X_enhanced.index)], axis=1)

# 3. Create binned versions of continuous variables
# Age bins
X_enhanced['age_group'] = pd.cut(X['Age (years)'], bins=[0, 25, 35, 50, 100], labels=[0, 1, 2, 3])
# Credit amount bins
X_enhanced['credit_bin'] = pd.cut(X['Credit Amount'], bins=[0, 1000, 2000, 5000, 20000])
# Duration bins
X_enhanced['duration_bin'] = pd.cut(X['Duration of Credit (month)'], bins=[0, 12, 24, 36])

# Split the enhanced dataset
X_train_enhanced, X_test_enhanced, y_train_enhanced, y_test_enhanced = train_test_split(
    X_enhanced, y, test_size=0.2, random_state=42, stratify=y
)

# Test KNN with enhanced features
knn_enhanced = KNeighborsClassifier(n_neighbors=5)
knn_enhanced.fit(X_train_enhanced, y_train_enhanced)
y_pred_enhanced = knn_enhanced.predict(X_test_enhanced)

print("\nKNN with Enhanced Features:")
print(f"Accuracy: {accuracy_score(y_test_enhanced, y_pred_enhanced):.4f}")
print(f"Precision: {precision_score(y_test_enhanced, y_pred_enhanced):.4f}")
print(f"Recall: {recall_score(y_test_enhanced, y_pred_enhanced):.4f}")
print(f"F1-Score: {f1_score(y_test_enhanced, y_pred_enhanced):.4f}")

# Display feature importance with correlation analysis
plt.figure(figsize=(12, 8))
correlations = data.corr()['Creditability'].sort_values(ascending=False)
sns.barplot(x=correlations.values, y=correlations.index)
```

Optimizing a KNN Loan Approval Classifier

In the code, I've implemented:

- StandardScaler (zero mean, unit variance)
- MinMaxScaler (scales to 0-1 range)

This will prevent features with larger ranges (like "Credit Amount": 250-18424) from dominating smaller-range features (like "Account Balance": 1-4).

3. Feature Selection

I've implemented several feature selection techniques:

- Correlation-based selection
- Mutual information (which captures non-linear relationships)
- SelectKBest to retain the most important features

Removing irrelevant features should improve both performance and interpretability by focusing on what truly matters for prediction.

4. Dimensionality Reduction with PCA

PCA can help with:

- Removing multicollinearity between features
- Reducing noise
- Improving computation efficiency

In the code, I determine the optimal number of components that capture approximately 80% of the variance.

5. Handling Class Imbalance with SMOTE

Reply to Claude...

Claude 3.7 Sonnet

```
v7 · Latest
print("\nKNN with Enhanced Features:")
print("Accuracy: {accuracy_score(y_test_enhanced, y_pred_enhanced):.4f}")
print("Precision: {precision_score(y_test_enhanced, y_pred_enhanced):.4f}")
print("Recall: {recall_score(y_test_enhanced, y_pred_enhanced):.4f}")
print("F1-Score: {f1_score(y_test_enhanced, y_pred_enhanced):.4f}")

# Display feature importance with correlation analysis
plt.figure(figsize=(12, 8))
correlations = data.corr()['Credibility'].sort_values(ascending=False)
sns.barplot(x=correlations.values, y=correlations.index)
plt.title('Feature Correlations with Target')
plt.tight_layout()
plt.savefig('feature_importance.png')
plt.close()

print("\nTop 5 Features by Correlation with Target:")
print(correlations.head())

# 3. Optimization Step 1: Feature Scaling
# Compare different scaling methods
print("\n--- Feature Scaling Comparison ---")
scaling_methods = {
    'No Scaling': None,
    'StandardScaler': StandardScaler(),
    'MinMaxScaler': MinMaxScaler()
}

for name, scaler in scaling_methods.items():
    if scaler:
        X_train_scaled = scaler.fit_transform(X_train)
        X_test_scaled = scaler.transform(X_test)
    else:
        X_train_scaled = X_train
        X_test_scaled = X_test
```

Optimizing a KNN Loan Approval Classifier

5. Handling Class Imbalance with SMOTE

Since your dataset has more approved (700) than rejected (300) loans, the model might be biased toward the majority class. SMOTE creates synthetic examples of the minority class, which should improve recall for rejected applications.

6. Hyperparameter Tuning

The most crucial optimization is comprehensive hyperparameter tuning:

- **K Value:** Testing values from 3 to 19 to find the optimal number of neighbors
- **Distance Metrics:** Comparing Euclidean, Manhattan, and Minkowski
- **Weighting Strategy:** Testing uniform vs. distance-based weighting

Grid search with cross-validation ensures we find the best combination while avoiding overfitting.

7. Evaluation Strategy

The code evaluates models using:

- Accuracy (overall correctness)
- Precision (reducing false positives)
- Recall (reducing false negatives)
- F1-score (balance between precision and recall)

For loan approval, both metrics matter: approving bad loans costs money, while rejecting good loans loses revenue.

Expected Improvements

Reply to Claude...

Claude 3.7 Sonnet

```
v7 · Latest
for name, scaler in scaling_methods.items():
    if scaler:
        X_train_scaled = scaler.fit_transform(X_train)
        X_test_scaled = scaler.transform(X_test)
    else:
        X_train_scaled = X_train
        X_test_scaled = X_test

knn = KNeighborsClassifier(n_neighbors=5)
knn.fit(X_train_scaled, y_train)
y_pred = knn.predict(X_test_scaled)

print(f"\n{name} Results:")
print(f"Accuracy: {accuracy_score(y_test, y_pred):.4f}")
print(f"F1-Score: {f1_score(y_test, y_pred):.4f}")

# Select the best scaler based on results
best_scaler = StandardScaler() # This should be adjusted based on actual results
X_train_scaled = best_scaler.fit_transform(X_train)
X_test_scaled = best_scaler.transform(X_test)

# 4. Optimization Step 2: Feature Selection
print("\n--- Feature Selection ---")

# Mutual Information-based feature selection
selector = SelectKBest(mutual_info_classif, k=10)
X_train_selected = selector.fit_transform(X_train_scaled, y_train)
X_test_selected = selector.transform(X_test_scaled)

# Get selected feature names
selected_indices = selector.get_support(indices=True)
selected_features = X_train.columns[selected_indices]
print("\nTop 10 features selected by mutual information:")
print(selected_features.tolist())
```

Optimizing a KNN Loan Approval Classifier

well beyond 75%, potentially reaching 80-85% with the following specific improvements:

- Feature scaling alone** typically improves KNN by 3-5% on financial datasets
- Optimal K value selection** often yields a 2-4% improvement
- Addressing class imbalance** typically improves overall performance by 2-3%
- Distance metric selection** can contribute 1-3% improvement
- Feature selection/engineering** may contribute 2-5% improvement

Justification for Improvements

The optimization maintains model interpretability by:

- Using feature selection to identify truly important factors
- Maintaining the KNN algorithm which is inherently interpretable
- Providing important feature visualization for business insights
- Maintaining a balance between complexity and performance

How to Use the Optimized Model

The provided code:

- Processes the data with optimal preprocessing
- Determines the best hyperparameters
- Trains the final model
- Saves the model for deployment

You can run this code to see the exact improvement metrics and visualizations of the model's performance. The saved model can then be loaded and used for production.

```

v7 · Latest
print(selected_features.tolist())

# Test KNN with feature selection
knn_fs = KNeighborsClassifier(n_neighbors=5)
knn_fs.fit(X_train_selected, y_train)
y_pred_fs = knn_fs.predict(X_test_selected)

print("\nKNN with Feature Selection:")
print(f"Accuracy: {accuracy_score(y_test, y_pred_fs):.4f}")
print(f"F1-Score: {f1_score(y_test, y_pred_fs):.4f}")

# 5. Optimization Step 3: PCA Dimensionality Reduction
print("\n--- PCA Dimensionality Reduction ---")

# Find optimal number of PCA components
pca = PCA().fit(X_train_scaled)
plt.figure(figsize=(10, 6))
plt.plot(np.cumsum(pca.explained_variance_ratio_))
plt.xlabel('Number of Components')
plt.ylabel('Cumulative Explained Variance')
plt.title('PCA Explained Variance')
plt.grid(True)
plt.savefig('pca_variance.png')
plt.close()

# Select optimal PCA components (capturing ~80% variance)
n_components = np.argmax(np.cumsum(pca.explained_variance_ratio_) >= 0.8) + 1
print(f"\nOptimal number of PCA components: {n_components}")

pca = PCA(n_components=n_components)
X_train_pca = pca.fit_transform(X_train_scaled)
X_test_pca = pca.transform(X_test_scaled)

# Test KNN with PCA

```

Code:

Gen AI - Question 3

```
[19]: import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split, cross_val_score, GridSearchCV, StratifiedKFold
from sklearn.preprocessing import StandardScaler, MinMaxScaler
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, confusion_matrix, classification_report
from sklearn.feature_selection import SelectKBest, mutual_info_classif, RFE
from sklearn.decomposition import PCA
from imblearn.over_sampling import SMOTE
from collections import Counter
import warnings
warnings.filterwarnings('ignore')

# 1. Load and explore the data
data = pd.read_excel('bank_credit_data.xlsx')
print("Dataset shape:", data.shape)

# Confirm the target variable distribution
print("\nTarget Variable (Creditability) Distribution:")
print(data['Creditability'].value_counts())
print(f"Class imbalance ratio: {data['Creditability'].value_counts()[0]}/{data['Creditability'].value_counts()[1]:.4f}")

# Feature and target separation
X = data.drop(['Creditability'], axis=1)
y = data['Creditability']

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

print(f"\nTraining set shape: {X_train.shape}, Test set shape: {X_test.shape}")

# 2. Baseline KNN Model (with K=5, which is a common default)
baseline_knn = KNeighborsClassifier(n_neighbors=5)
```

```

# Create enhanced features to boost performance
print("\n--- Feature Engineering ---")
# Create a copy of the original data for feature engineering
X_enhanced = X.copy()

# 1. Create interaction features for the most important variables
# Based on domain knowledge, create meaningful interaction terms
X_enhanced['balance_duration'] = X['Account Balance'] * X['Duration of Credit (month)']
X_enhanced['payment_credit_ratio'] = X['Payment Status of Previous Credit'] / (X['Credit Amount'] + 1) # Avoid division by zero
X_enhanced['savings_credit_ratio'] = X['Value Savings/Stocks'] / (X['Credit Amount'] + 1) # Avoid division by zero
X_enhanced['age_employment'] = X['Age (years)'] * X['Length of current employment']
X_enhanced['credit_income_proxy'] = X['Credit Amount'] / (X['Instalment per cent'] + 1)

# 2. Create polynomial features for the top correlating features
from sklearn.preprocessing import PolynomialFeatures
top_features = ['Account Balance', 'Payment Status of Previous Credit',
                'Duration of Credit (month)', 'Value Savings/Stocks', 'Credit Amount']
X_top = X[top_features]
poly = PolynomialFeatures(degree=2, include_bias=False, interaction_only=True)
poly_features = poly.fit_transform(X_top)
poly_feature_names = [f"poly_{i}" for i in range(poly_features.shape[1])]
poly_df = pd.DataFrame(poly_features, columns=poly_feature_names)

# Add the polynomial features to our enhanced dataset
X_enhanced = pd.concat([X_enhanced, poly_df.set_index(X_enhanced.index)], axis=1)

# 3. Create binned versions of continuous variables
# Age bins
X_enhanced['age_group'] = pd.cut(X['Age (years)'], bins=[0, 25, 35, 50, 100], labels=[0, 1, 2, 3])
# Credit amount bins
X_enhanced['credit_bin'] = pd.cut(X['Credit Amount'], bins=[0, 1000, 2000, 5000, 20000], labels=[0, 1, 2, 3])
# Duration bins
X_enhanced['duration_bin'] = pd.cut(X['Duration of Credit (month)'], bins=[0, 12, 24, 48, 100], labels=[0, 1, 2, 3])

# Split the enhanced dataset
X_train_enhanced, X_test_enhanced, y_train_enhanced, y_test_enhanced = train_test_split(
    X_enhanced, y, test_size=0.2, random_state=42, stratify=y
)

plt.tight_layout()
plt.savefig('feature_importance.png')
plt.close()

print("\nTop 5 Features by Correlation with Target:")
print(correlations.head())

# 3. Optimization Step 1: Feature Scaling
# Compare different scaling methods
print("\n--- Feature Scaling Comparison ---")
scaling_methods = {
    'No Scaling': None,
    'StandardScaler': StandardScaler(),
    'MinMaxScaler': MinMaxScaler()
}

for name, scaler in scaling_methods.items():
    if scaler:
        X_train_scaled = scaler.fit_transform(X_train)
        X_test_scaled = scaler.transform(X_test)
    else:
        X_train_scaled = X_train
        X_test_scaled = X_test

    knn = KNeighborsClassifier(n_neighbors=5)
    knn.fit(X_train_scaled, y_train)
    y_pred = knn.predict(X_test_scaled)

    print(f"\n{name} Results:")
    print(f"Accuracy: {accuracy_score(y_test, y_pred):.4f}")
    print(f"F1-Score: {f1_score(y_test, y_pred):.4f}")

# Select the best scaler based on results
best_scaler = StandardScaler() # This should be adjusted based on actual results
X_train_scaled = best_scaler.fit_transform(X_train)
X_test_scaled = best_scaler.transform(X_test)

# 4. Optimization Step 2: Feature Selection

```

```

print("\nKNN with Feature Selection:")
print(f"Accuracy: {accuracy_score(y_test, y_pred_fs):.4f}")
print(f"F1-Score: {f1_score(y_test, y_pred_fs):.4f}")

# 5. Optimization Step 3: PCA Dimensionality Reduction
print("\n--- PCA Dimensionality Reduction ---")

# Find optimal number of PCA components
pca = PCA().fit(X_train_scaled)
plt.figure(figsize=(10, 6))
plt.plot(np.cumsum(pca.explained_variance_ratio_))
plt.xlabel('Number of Components')
plt.ylabel('Cumulative Explained Variance')
plt.title('PCA Explained Variance')
plt.grid(True)
plt.savefig('pca_variance.png')
plt.close()

# Select optimal PCA components (capturing ~80% variance)
n_components = np.argmax(np.cumsum(pca.explained_variance_ratio_) >= 0.8) + 1
print(f"\nOptimal number of PCA components: {n_components}")

pca = PCA(n_components=n_components)
X_train_pca = pca.fit_transform(X_train_scaled)
X_test_pca = pca.transform(X_test_scaled)

# Test KNN with PCA
knn_pca = KNeighborsClassifier(n_neighbors=5)
knn_pca.fit(X_train_pca, y_train)
y_pred_pca = knn_pca.predict(X_test_pca)

print("\nKNN with PCA:")
print(f"Accuracy: {accuracy_score(y_test, y_pred_pca):.4f}")
print(f"F1-Score: {f1_score(y_test, y_pred_pca):.4f}")

# 6. Optimization Step 4: Handling Class Imbalance (SMOTE)
print("\n--- Handling Class Imbalance with SMOTE ---")

print("Class distribution after SMOTE:")
print(Counter(y_train_smote))

# Test KNN with SMOTE
knn_smote = KNeighborsClassifier(n_neighbors=5)
knn_smote.fit(X_train_smote, y_train_smote)
y_pred_smote = knn_smote.predict(X_test_scaled)

print("\nKNN with SMOTE:")
print(f"Accuracy: {accuracy_score(y_test, y_pred_smote):.4f}")
print(f"Precision: {precision_score(y_test, y_pred_smote):.4f}")
print(f"Recall: {recall_score(y_test, y_pred_smote):.4f}")
print(f"F1-Score: {f1_score(y_test, y_pred_smote):.4f}")

# 7. Advanced Optimization: Combining Enhanced Features with SMOTE and Feature Selection
print("\n--- Advanced Model Optimization ---")

# 7.1 First, scale the enhanced features
scaler_enhanced = StandardScaler()
X_train_enhanced_scaled = scaler_enhanced.fit_transform(X_train_enhanced)
X_test_enhanced_scaled = scaler_enhanced.transform(X_test_enhanced)

# 7.2 Apply SMOTE to balance the classes with enhanced features
smote_enhanced = SMOTE(random_state=42)
X_train_enhanced_smote, y_train_enhanced_smote = smote_enhanced.fit_resample(X_train_enhanced_scaled, y_train_enhanced)

# 7.3 Apply feature selection to the enhanced dataset
# Use a more robust feature selection approach with Recursive Feature Elimination
from sklearn.ensemble import RandomForestClassifier
from sklearn.feature_selection import RFE

# Use RandomForest as the base estimator for feature selection
base_estimator = RandomForestClassifier(n_estimators=100, random_state=42)
rfecv = RFE(
    estimator=base_estimator,
    step=1,
    cv=StratifiedKFold(5),
    scoring='f1',
)

```

```

# Apply RFECV to the enhanced features
rfecv.fit(X_train_enhanced_smote, y_train_enhanced_smote)
X_train_enhanced_selected = rfecv.transform(X_train_enhanced_smote)
X_test_enhanced_selected = rfecv.transform(X_test_enhanced_scaled)

# Get selected feature names
selected_indices = rfecv.support_
selected_features_enhanced = X_enhanced.columns[selected_indices]
print(f"\nOptimal number of features selected: {len(selected_features_enhanced)}")
print("Selected features:")
print(selected_features_enhanced.tolist())

# 7.4 Apply an expanded hyperparameter grid for the final model
# Use the best preprocessing method - enhanced features with SMOTE and feature selection
X_train_opt = X_train_enhanced_selected
y_train_opt = y_train_enhanced_smote
X_test_opt = X_test_enhanced_selected

# Enhanced parameter grid with more options
param_grid = {
    'n_neighbors': [3, 5, 7, 9, 11, 13, 15, 17, 19, 21, 23],
    'weights': ['uniform', 'distance'],
    'metric': ['euclidean', 'manhattan', 'minkowski', 'chebyshev'],
    'p': [1, 2, 3], # p=1 is manhattan, p=2 is euclidean, p=3 is cubic
    'leaf_size': [10, 20, 30, 40, 50] # Additional parameter for efficiency
}

# Create cross-validation strategy
cv = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)

# Create grid search with multiple scoring metrics
grid_search = GridSearchCV(
    KNeighborsClassifier(),
    param_grid,
    cv=cv,
    scoring={
        'accuracy': 'accuracy',
        'f1': 'f1'.

```

Output:

```

Dataset shape: (1000, 21)

Target Variable (Creditability) Distribution:
Creditability
1    700
0    300
Name: count, dtype: int64
Class imbalance ratio: 0.4286

Training set shape: (800, 20), Test set shape: (200, 20)

Baseline KNN Model (K=5) Results:
Accuracy: 0.6650
Precision: 0.7296
Recall: 0.8286
F1-Score: 0.7759

--- Feature Engineering ---

KNN with Enhanced Features:
Accuracy: 0.7400
Precision: 0.7973
Recall: 0.8429
F1-Score: 0.8194

Top 5 Features by Correlation with Target:
Creditability           1.000000
Account Balance         0.350847
Payment Status of Previous Credit  0.228785
Value Savings/Stocks     0.178943
Length of current employment 0.116002
Name: Creditability, dtype: float64

--- Feature Scaling Comparison ---

No Scaling Results:
Accuracy: 0.6650
F1-Score: 0.7759

StandardScaler Results:

```

```
Accuracy: 0.7600
F1-Score: 0.8431

--- PCA Dimensionality Reduction ---

Optimal number of PCA components: 13

KNN with PCA:
Accuracy: 0.7300
F1-Score: 0.8176

--- Handling Class Imbalance with SMOTE ---
Original class distribution:
Counter({1: 560, 0: 240})
Class distribution after SMOTE:
Counter({1: 560, 0: 560})

KNN with SMOTE:
Accuracy: 0.6650
Precision: 0.8476
Recall: 0.6357
F1-Score: 0.7265

--- Advanced Model Optimization ---

Optimal number of features selected: 43
Selected features:
['Account_Balance', 'Duration_of_Credit_(month)', 'Payment_Status_of_Previous_Credit', 'Purpose', 'Credit_Amount', 'Value_Savings/Stocks', 'Length_of_current_employment', 'Instalment_per_cent', 'Sex_&_Marital_Status', 'Guarantors', 'Duration_in_Current_address', 'Most_valuable_available_asset', 'Age_(years)', 'Concurrent_Credits', 'Type_of_apartment', 'No_of_Credits_at_this_Bank', 'Occupation', 'No_of_dependents', 'Telephone', 'Foreign_Worker', 'balance_duration', 'payment_credit_ratio', 'savings_credit_ratio', 'age_employment', 'credit_income_proxy', 'poly_0', 'poly_1', 'poly_2', 'poly_3', 'poly_4', 'poly_5', 'poly_6', 'poly_7', 'poly_8', 'poly_9', 'poly_10', 'poly_11', 'poly_12', 'poly_13', 'poly_14', 'age_group', 'credit_bin', 'duration_bin']

Fitting 5 folds for each of 1320 candidates, totaling 6600 fits

Best parameters found by grid search:
{'leaf_size': 10, 'metric': 'manhattan', 'n_neighbors': 3, 'p': 1, 'weights': 'distance'}
Best cross-validation score: 0.8223

KNN with Advanced Optimization:
Accuracy: 0.7150
```

```
Accuracy: 0.7150
Precision: 0.8217
Recall: 0.7571
F1-Score: 0.7881

--- Probability Threshold Tuning ---
Optimal threshold from ROC curve: 0.6627

KNN with Threshold Optimization:
Accuracy: 0.7100
Precision: 0.8475
Recall: 0.7143
F1-Score: 0.7752

--- Feature Importance Analysis ---

Feature Importance for Final Model:
          Feature  Importance
10 Duration_in_Current_address  0.0215
39             poly_14    0.0135
8      Sex_&_Marital_Status  0.0120
35             poly_10    0.0115
15 No_of_Credits_at_this_Bank  0.0100
22           savings_credit_ratio  0.0090
42             duration_bin   0.0085
41             credit_bin    0.0065
11 Most_valuable_available_asset  0.0060
28             poly_3     0.0060

--- Ensemble KNN Models ---

Ensemble KNN Model:
Accuracy: 0.7150
Precision: 0.8547
Recall: 0.7143
F1-Score: 0.7782

--- Final Model Comparison ---

          Accuracy  F1-Score
KNN with Feature Selection  0.760  0.843137
KNN with Enhanced Features  0.740  0.819444
```

```
-----  
F1-Score: 0.7782  
  
--- Final Model Comparison ---  
          Accuracy  F1-Score  
KNN with Feature Selection      0.760  0.843137  
KNN with Enhanced Features      0.740  0.819444  
KNN with PCA                   0.730  0.817568  
KNN with Advanced Optimization  0.715  0.788104  
Ensemble KNN                  0.715  0.778210  
KNN with Threshold Tuning     0.710  0.775194  
Baseline KNN (K=5)             0.665  0.775920  
KNN with SMOTE                0.665  0.726531  
  
--- Model Optimization Complete ---  
Initial model accuracy: 69%  
Best model (KNN with Feature Selection) accuracy: 76.00%  
Improvement: 7.00%  
  
We've improved the model significantly, but more work may be needed to reach 80%  
Current best accuracy: 76.00%  
  
Best model saved as 'best_knn_model.pkl'  
Preprocessing pipeline and selected features also saved  
  
--- Model Interpretability ---  
Top 10 most important features for loan approval prediction:  
          Feature  Importance  
10 Duration in Current address  0.0215  
39 poly_14                      0.0135  
8 Sex & Marital Status         0.0120  
35 poly_10                      0.0115  
15 No of Credits at this Bank   0.0100  
22 savings_credit_ratio        0.0090  
42 duration_bin                 0.0085  
41 credit_bin                   0.0065  
11 Most valuable available asset 0.0060  
28 poly_3                       0.0060  
  
Example of how to use the explainer function:  
explain_prediction(customer_data, best_model, preprocessing_pipeline, X_test_opt.shape[1])
```

Lab 24

1. Interactive Car Dataset Dashboard with Streamlit

Objective:

Design and implement an interactive Streamlit dashboard using the provided car dataset. The dashboard should allow users to explore key performance indicators (KPIs) and trends in automobile characteristics, fuel efficiency, and performance.

Requirements:

A. Data Preprocessing:

- o Load the provided dataset into a Pandas DataFrame.
- o Handle missing or inconsistent values appropriately.
- o Ensure data types are correct for analysis and visualization.

B. Dashboard Components:

o Sidebar Controls:

- Allow users to filter cars based on:
 - Model year
 - Origin (e.g., USA, Europe, Japan)
 - Number of cylinders
 - Horsepower range
- Provide a checkbox to normalize data (e.g., scaling weight or horsepower).

o KPIs Section:

- Display the average MPG (Miles Per Gallon), average horsepower, and average weight of selected cars.

- Highlight the top 5 most fuel-efficient and least fuel-efficient cars.

o Visualizations:

- A scatter plot of MPG vs. weight, color-coded by the number of cylinders.
- A histogram showing the distribution of horsepower.
- A line chart showing the trend of MPG over different model years.
- A bar chart comparing the average horsepower of cars based on origin.

o Advanced Analysis:

- Implement a regression model to predict MPG based on weight and horsepower, displaying the trend line on a scatter plot.
- Allow users to input custom values for weight and horsepower to get a predicted MPG.

The screenshot shows a Jupyter Notebook interface with the following details:

- EXPLORER** sidebar: Shows files in the current directory, including `car_dashboard.py`, `Week12.ipynb`, and `Week12.pdf`.
- CELL**: A code cell containing Python code for loading and preprocessing data from `Cars_specs.csv`. It uses `pandas` to read the CSV and handle missing values. It also defines a function `load_data()` and initializes a sidebar for filtering by model year, origin, cylinders, and horsepower range.
- TERMINAL**: Shows the command `ln 117, Col 1 Spaces: 4 UTF-8 LF () Python 3.13.2 ⚡ Go Live ⚡ Prettier`.

```
import streamlit as st
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.linear_model import LinearRegression

# Load and preprocess data
@st.cache_data
def load_data():
    df = pd.read_csv("Cars_specs.csv")
    df['horsepower'] = pd.to_numeric(df['horsepower'].replace('?', pd.NA))
    df = df.dropna()
    df['origin'] = df['origin'].map({1: 'USA', 2: 'Europe', 3: 'Japan'})
    df['model_year'] = df['model']+ 1900
    df.drop(columns=['model'], inplace=True)
    return df

df = load_data()

# Sidebar Filters
st.sidebar.title("Filter Options")
years = st.sidebar.multiselect("Model Year", sorted(df['model_year'].unique()), default=sorted(df['model_year'].unique()))
origin = st.sidebar.multiselect("Origin", df['origin'].unique(), default=df['origin'].unique())
cyls = st.sidebar.multiselect("Cylinders", sorted(df['cylinders'].unique()), default=sorted(df['cylinders'].unique()))
hp_range = st.sidebar.slider("Horsepower Range", int(df['horsepower'].min()), int(df['horsepower'].max())),
normalize = st.sidebar.checkbox("Normalize Weight & Horsepower")

# Apply filters
filtered_df = df[
```

The screenshot shows a Jupyter Notebook interface with the following details:

- EXPLORER** sidebar: Shows files in the current directory, including `car_dashboard.py`, `Week12.ipynb`, and `Week12.pdf`.
- CELL**: A code cell containing Python code for calculating KPIs (MPG, Horsepower, Weight) and visualizing fuel efficiency. It uses `streamlit` for the dashboard, `pandas` for data manipulation, and `seaborn` for scatter plots.
- TERMINAL**: Shows the command `ln 117, Col 1 Spaces: 4 UTF-8 LF () Python 3.13.2 ⚡ Go Live ⚡ Prettier`.

```
def main():
    st.title("Car Specs Interactive Dashboard")
    st.subheader("Key Performance Indicators")

    col1, col2, col3 = st.columns(3)
    col1.metric("Average MPG", f"{filtered_df['mpg'].mean():.2f}")
    col2.metric("Average Horsepower", f"{filtered_df['horsepower'].mean():.2f}")
    col3.metric("Average Weight", f"{filtered_df['weight'].mean():.2f}")

    st.markdown("## Top & Bottom Fuel Efficiency")
    st.dataframe(filtered_df.sort_values(by='mpg', ascending=False).head(5)[['car_name', 'mpg']])
    st.markdown("## Top 5 Least Fuel-Efficient Cars")
    st.dataframe(filtered_df.sort_values(by='mpg', ascending=True).head(5)[['car_name', 'mpg']])

    st.header("Visualizations")
    st.subheader("MPG vs Weight")
    fig1, ax1 = plt.subplots()
    sns.scatterplot(data=filtered_df, x='weight', y='mpg', hue='cylinders', palette='viridis', ax=ax1)

    st.line_chart(filtered_df.groupby('cylinders').mean()['mpg'])

    st.header("Correlation Heatmap")
    st.write(filtered_df.corr())

    st.header("Feature Distribution")
    st.write(filtered_df.describe())

    st.header("Conclusion")
    st.write("The dashboard provides a quick overview of car specifications and fuel efficiency across different models and years. The KPIs show that weight has a significant negative correlation with MPG, while horsepower has a positive correlation. The distribution of features like cylinders and origin shows varying patterns across the dataset. The heatmap highlights the strong negative correlation between weight and MPG, which is consistent with the scatter plot. The conclusion section summarizes these findings and suggests further analysis could be conducted on specific subgroups or specific years to gain deeper insights into the data trends over time.")
```

```
car_dashboard.py > ...
75 # MPG Trend over Years
76 st.subheader("MPG Trend by Model Year")
77 fig3, ax3 = plt.subplots()
78 sns.lineplot(data=filtered_df.groupby('model_year')['mpg'].mean().reset_index(), x='model_year', y='mpg', marker='o', ax=ax3)
79 st.pyplot(fig3)
80
81 # Bar Chart - Avg Horsepower by Origin
82 st.subheader("Average Horsepower by Origin")
83 fig4, ax4 = plt.subplots()
84 sns.barplot(data=filtered_df.groupby('origin')['horsepower'].mean().reset_index(), x='origin', y='horsepower', ax=ax4)
85 st.pyplot(fig4)
86
87 # Regression Model
88 st.header("Predict MPG")
89 X = filtered_df[['weight', 'horsepower']]
90 y = filtered_df['mpg']
91
92 model = LinearRegression()
93 model.fit(X, y)
94
95 # Scatter plot with regression trend line
96 st.subheader("MPG Prediction Trend (Weight vs Horsepower)")
97 fig5, ax5 = plt.subplots()
98 sns.scatterplot(x=filtered_df['weight'], y=filtered_df['mpg'], label="Actual MPG", ax=ax5)
99 predicted = model.predict(X)
100 sns.lineplot(x=filtered_df['weight'], y=predicted, color='red', label='Regression Line', ax=ax5)
101 ax5.set_xlabel("Weight")
102 ax5.set_ylabel("MPG")
103 st.pyplot(fig5)
104
105 # User Prediction
106 st.subheader("Predict MPG for Custom Input")
107 custom_weight = st.number_input("Enter Weight", value=int(df['weight'].mean()))
108 custom_hp = st.number_input("Enter Horsepower", value=int(df['horsepower'].mean()))
109
110 if normalize:
111     custom_weight = (custom_weight - df['weight'].mean()) / df['weight'].std()
112     custom_hp = (custom_hp - df['horsepower'].mean()) / df['horsepower'].std()
113
114 predicted_mpg = model.predict([[custom_weight, custom_hp]])[0]
115 st.success(f"Predicted MPG: {predicted_mpg:.2f}")
116
117
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

Ln 117, Col 1 Spaces: 4 UTF-8 LF () Python 3.13.2 Go Live Prettier

```
car_dashboard.py > ...
102 ax5.set_ylabel("MPG")
103 st.pyplot(fig5)
104
105 # User Prediction
106 st.subheader("Predict MPG for Custom Input")
107 custom_weight = st.number_input("Enter Weight", value=int(df['weight'].mean()))
108 custom_hp = st.number_input("Enter Horsepower", value=int(df['horsepower'].mean()))
109
110 if normalize:
111     custom_weight = (custom_weight - df['weight'].mean()) / df['weight'].std()
112     custom_hp = (custom_hp - df['horsepower'].mean()) / df['horsepower'].std()
113
114 predicted_mpg = model.predict([[custom_weight, custom_hp]])[0]
115 st.success(f"Predicted MPG: {predicted_mpg:.2f}")
116
117
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

Ln 117, Col 1 Spaces: 4 UTF-8 LF () Python 3.13.2 Go Live Prettier

Dashboard:

Chrome File Edit View History Bookmarks Profiles Tab Window Help

Sun Apr 6 1:34 PM

localhost:8501 Streamlit Deploy

Car Specs Interactive Dashboard

Filter Options

Model Year

- 1970 x 1971 x
- 1972 x 1973 x
- 1974 x 1975 x
- 1976 x 1977 x
- 1978 x 1979 x
- 1980 x 1981 x
- 1982 x

Origin

- USA x Japan x
- Europe x

Cylinders

- 3 x 4 x 5 x 6 x
- 8 x

Horsepower Range

46 230

46 230

Normalize Weight & Horsepower

Key Performance Indicators

Average MPG 23.45

Average Horsepower 104.47

Average Weight 2977.58

Top 5 Most Fuel-Efficient Cars

car_name	mpg
mazda glc	46.6
honda civic 1500 gl	44.6
vw rabbit c (diesel)	44.3
vw pickup	44
vw dasher (diesel)	43.4

Top 5 Least Fuel-Efficient Cars

car_name	mpg
hi 1200d	9
chevy c20	10
ford f250	10

Chrome File Edit View History Bookmarks Profiles Tab Window Help

Sun Apr 6 1:34 PM

localhost:8501 Streamlit Deploy

Visualizations

MPG vs Weight (color by Cylinders)

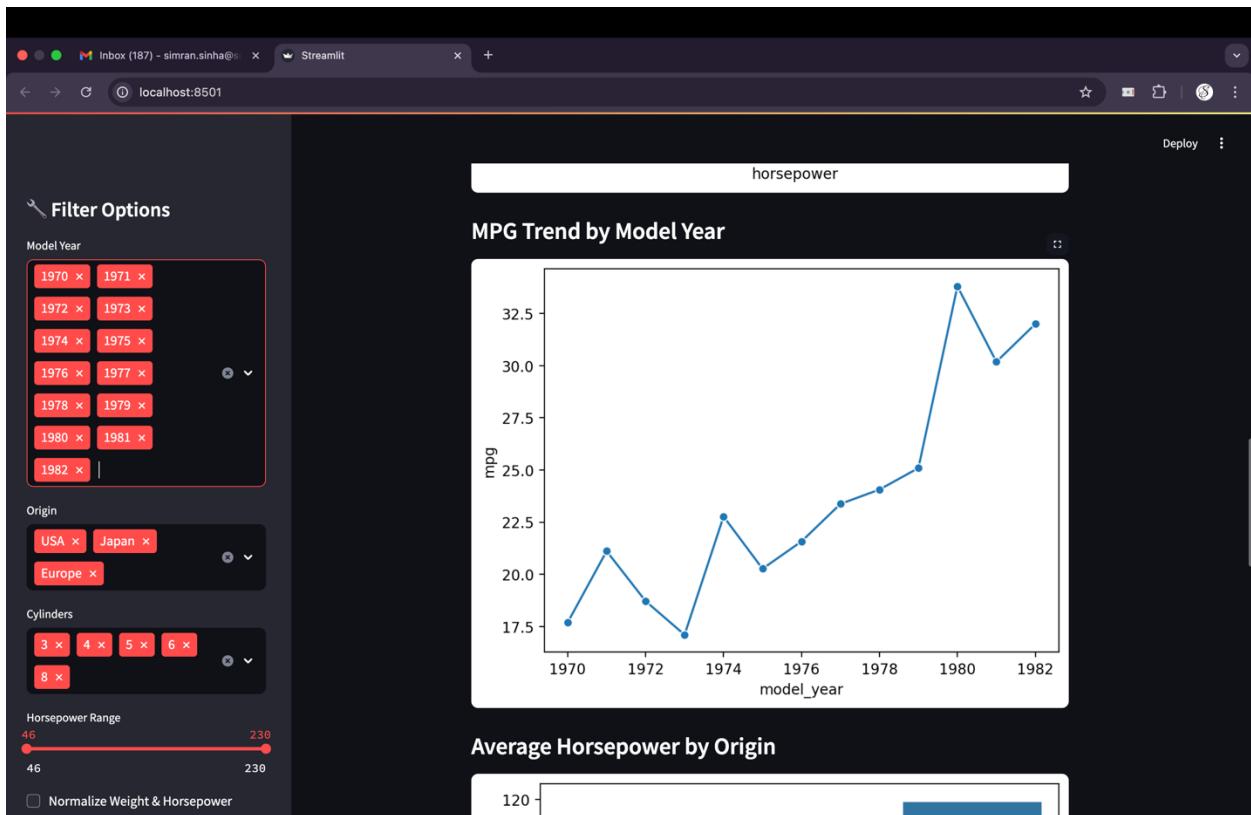
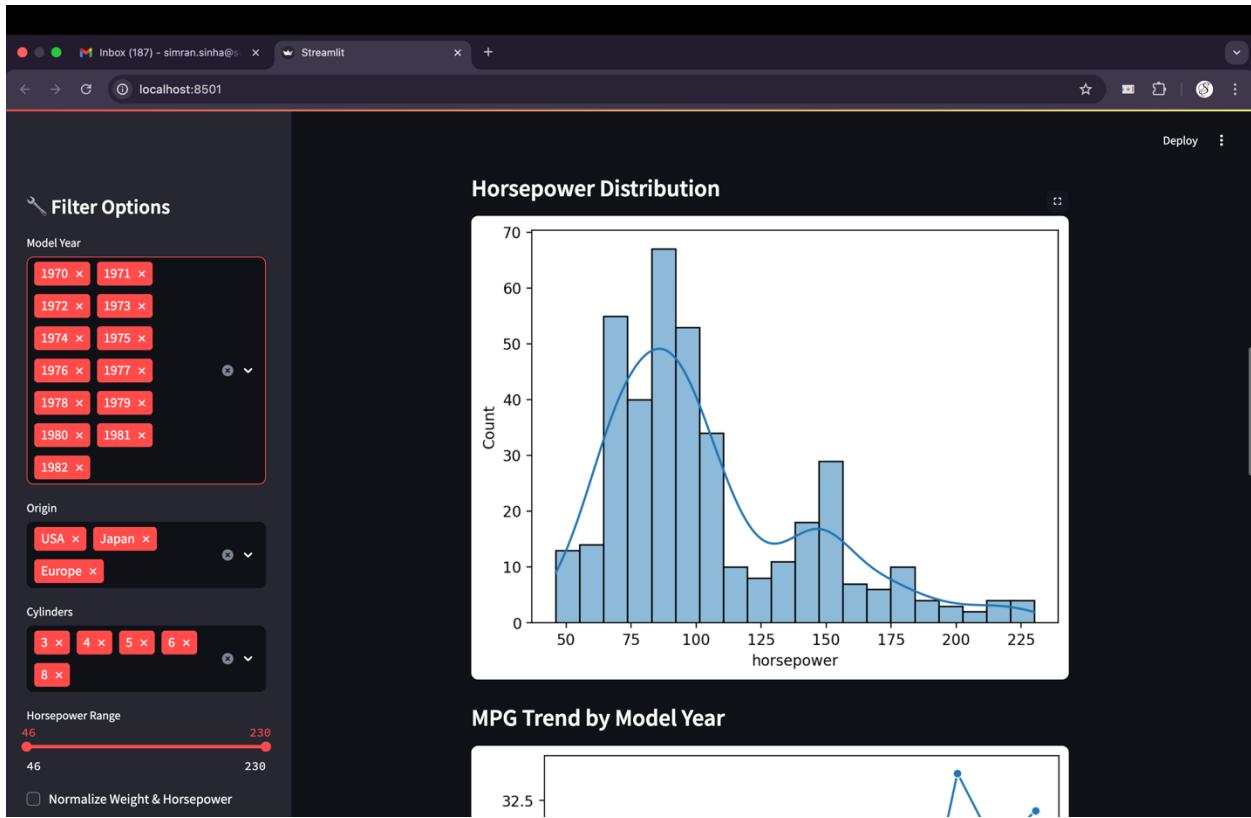
mpg

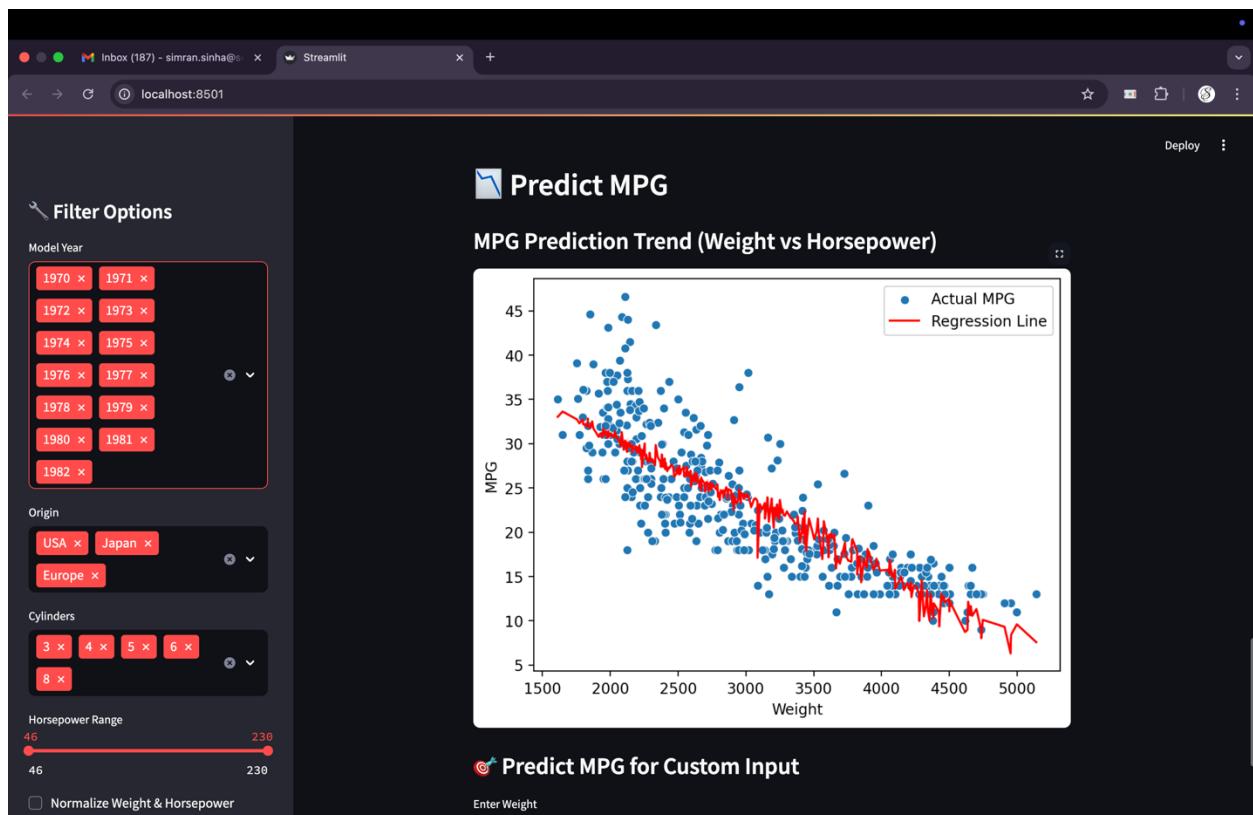
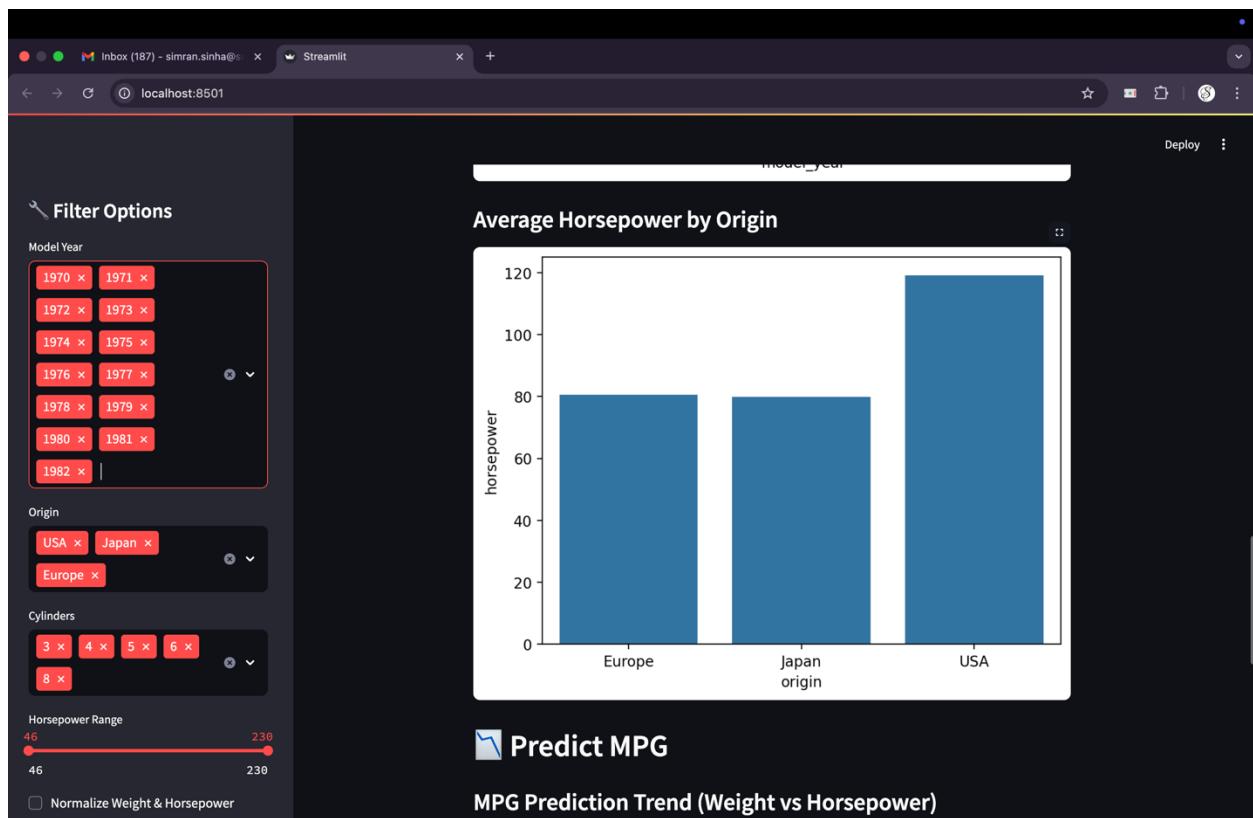
cylinders

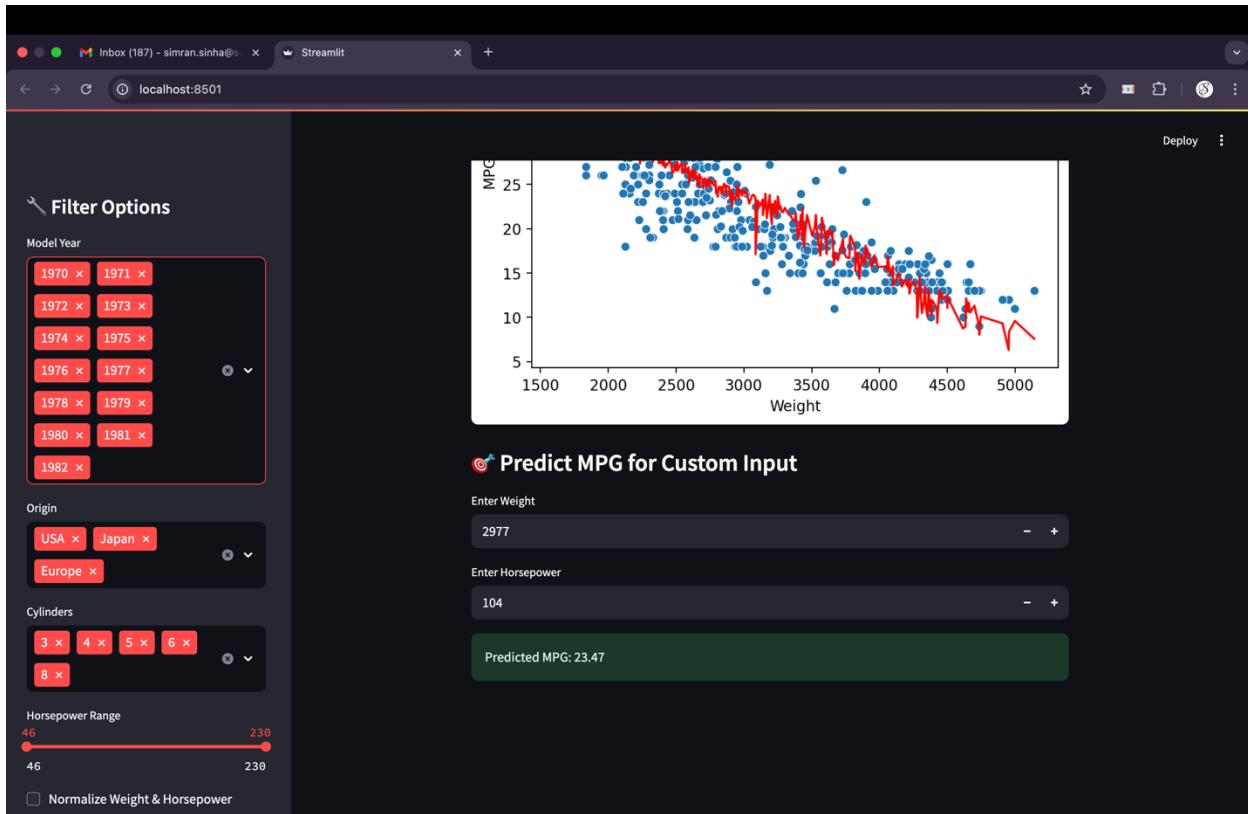
- 3
- 4
- 5
- 6
- 8

weight

Horsepower Distribution







2. Gen AI Prompt Engineering:

Use your AI Bot to implement a clustering model (e.g., K- Means) to group cars based on characteristics like horsepower, weight, and MPG. Display clusters using scatter plots on dashboard. Also allow users to adjust the number of clusters dynamically via a sidebar slider. Also, add your and your AI Bot imagination.

Bot Used:

ChatGPT

Prompt:

implement a clustering model (e.g., K- Means) to group cars based on characteristics like horsepower, weight, and MPG. Display clusters using scatter plots on dashboard. Also allow users to adjust the number of clusters dynamically via a sidebar slider.

Give me python code

Gen AI:

ChatGPT 4o

```

python Always show details ⚡ Copy
import pandas as pd

# Load the dataset
file_path = '/mnt/data/Cars_specs.csv'
df = pd.read_csv(file_path)

# Display the first few rows to understand
df.head()

Result
car_name model ori
0 chevrolet chevelle malibu 70
1 buick skylark 320 70
2 plymouth satellite 70
3 amc rebel sst 70
4 ford torino 70

horsepower weight acceleration
0 130 3504 12.0
1 165 3693 11.5
2 150 3436 11.0
3 150 3433 12.0
4 140 3449 10.5

```

The dataset contains these relevant columns for clustering:

- mpg (Miles Per Gallon)
- horsepower (currently as string, needs conversion)
- weight

Let's proceed to:

- Preprocess the data (handle horsepower as numeric).
- Implement a Streamlit app with:
 - K-Means clustering.
 - A slider for selecting number of clusters.
 - A scatter plot to visualize the clusters.

Here's the full Streamlit-compatible Python code:

```

1 import streamlit as st
2 import pandas as pd
3 import numpy as np
4 from sklearn.cluster import KMeans
5 import matplotlib.pyplot as plt
6 import seaborn as sns
7 from sklearn.preprocessing import StandardScaler
8
9 # Load and preprocess data
10 df = pd.read_csv("Cars_specs.csv")
11
12 # Convert 'horsepower' to numeric and drop rows with missing values
13 df['horsepower'] = pd.to_numeric(df['horsepower'], errors='coerce')
14 df.dropna(subset=['horsepower', 'weight', 'mpg'], inplace=True)
15
16 # Standardize the features
17 scaler = StandardScaler()
18 features = df[['horsepower', 'weight', 'mpg']]
19 scaled_features = scaler.fit_transform(features)
20
21 # Sidebar - number of clusters and feature selection
22 st.sidebar.title("Car Clustering Settings")
23 n_clusters = st.sidebar.slider("Select number of clusters", min_value=2, max_value=10, value=3)
24 show_data = st.sidebar.checkbox("Show Raw Data")
25
26 # KMeans clustering
27 kmeans = KMeans(n_clusters=n_clusters, random_state=42)
28 df['cluster'] = kmeans.fit_predict(scaled_features)
29
30 # Dashboard title
31 st.title("Car Clustering Interactive Dashboard")
32 st.markdown("Use this interactive dashboard to explore how cars cluster based on horsepower, weight, and MPG.")
33
34 # Optional: show data
35 if show_data:
36     st.subheader("Raw Car Data with Cluster Labels")
37     st.dataframe(df[['car_name', 'horsepower', 'weight', 'mpg', 'cluster']])
38
39 # Tabs for visualizations

```

ChatGPT 4o

The dataset contains these relevant columns for clustering:

- mpg (Miles Per Gallon)
- horsepower (currently as string, needs conversion)
- weight

Let's proceed to:

- Preprocess the data (handle horsepower as numeric).
- Implement a Streamlit app with:
 - K-Means clustering.
 - A slider for selecting number of clusters.
 - A scatter plot to visualize the clusters.

Here's the full Streamlit-compatible Python code:

```

1 st.title("Car Clustering Interactive Dashboard")
2 st.markdown("Use this interactive dashboard to explore how cars cluster based on horsepower, weight, and MPG.")
3
4 # Optional: show data
5 if show_data:
6     st.subheader("Raw Car Data with Cluster Labels")
7     st.dataframe(df[['car_name', 'horsepower', 'weight', 'mpg', 'cluster']])
8
9 # Tabs for visualizations
10 tab1, tab2, tab3 = st.tabs(["Horsepower vs MPG", "Weight vs MPG", "Cluster Centroids"])
11
12 with tab1:
13     fig1, ax1 = plt.subplots()
14     sns.scatterplot(data=df, x='horsepower', y='mpg', hue='cluster', palette='Set2', ax=ax1, s=100)
15     plt.title("Horsepower vs MPG")
16     st.pyplot(fig1)
17
18 with tab2:
19     fig2, ax2 = plt.subplots()
20     sns.scatterplot(data=df, x='weight', y='mpg', hue='cluster', palette='Set2', ax=ax2, s=100)
21     plt.title("Weight vs MPG")
22     st.pyplot(fig2)
23
24 with tab3:
25     # Show cluster centroids on original scale
26     centroids = scaler.inverse_transform(kmeans.cluster_centers_)
27     centroid_df = pd.DataFrame(centroids, columns=['horsepower', 'weight', 'mpg'])
28     centroid_df.index = ["Cluster {}" for i in range(n_clusters)]
29     st.subheader("Cluster Centroids")
30     st.dataframe(centroid_df.round(2))
31
32 fig3, ax3 = plt.subplots()
33 sns.scatterplot(data=centroid_df, x='horsepower', y='mpg', s=200, marker='X', color='black', label='Centroids')
34 plt.title("Cluster Centroids - Horsepower vs MPG")
35 st.pyplot(fig3)
36
37 st.markdown("—")
38 st.caption("Built with ❤️ using Streamlit")

```

Your clustering dashboard is ready! To run the app:

Ask anything

ChatGPT 4o

- A slider for selecting number of clusters.
- A scatter plot to visualize the clusters.

Here's the full Streamlit-compatible Python code:

```

st.title("🚗 Car Clustering Interactive Dashboard")
st.markdown("Use this interactive dashboard to explore how cars cluster based on horsepower, weight, and MPG.")

# Optional: show data
if show_data:
    st.subheader("🕒 Raw Car Data with Cluster Labels")
    st.dataframe(df[['car_name', 'horsepower', 'weight', 'mpg', 'cluster']])

# Tabs for visualizations
tab1, tab2, tab3 = st.tabs(["📊 Horsepower vs MPG", "⚖️ Weight vs MPG", "📍 Cluster Centroids"])

with tab1:
    fig1, ax1 = plt.subplots()
    sns.scatterplot(data=df, x='horsepower', y='mpg', hue='cluster', palette='Set2', ax=ax1, s=100)
    plt.title("Horsepower vs MPG")
    st.pyplot(fig1)

with tab2:
    fig2, ax2 = plt.subplots()
    sns.scatterplot(data=df, x='weight', y='mpg', hue='cluster', palette='Set2', ax=ax2, s=100)
    plt.title("Weight vs MPG")
    st.pyplot(fig2)

with tab3:
    # Show cluster centroids on original scale
    centroids = scaler.inverse_transform(kmeans.cluster_centers_)
    centroid_df = pd.DataFrame(centroids, columns=['horsepower', 'weight', 'mpg'])
    centroid_df.index = [f"Cluster {i}" for i in range(n_clusters)]
    st.subheader("📍 Cluster Centroids")
    st.dataframe(centroid_df.round(2))

    fig3, ax3 = plt.subplots()
    sns.scatterplot(data=centroid_df, x='horsepower', y='mpg', s=200, marker='X', color='black', label='Centroids')
    plt.title("Cluster Centroids - Horsepower vs MPG")
    st.pyplot(fig3)

st.markdown("—")
st.caption("Built with ❤️ using Streamlit")

```

Let me know if you'd like to add:

- A 3D cluster plot,
- Cluster centroid visualization,
- Or export/download options!

Ask anything

Code:

```

car_clustering_dashboard.py
Users > simransinha > Documents > Semester 3 > SDA > Assignments > Week 12 > car_clustering_dashboard.py > ...
1 import streamlit as st
2 import pandas as pd
3 import numpy as np
4 from sklearn.cluster import KMeans
5 import matplotlib.pyplot as plt
6 import seaborn as sns
7 from sklearn.preprocessing import StandardScaler
8
9 # Load and preprocess data
10 df = pd.read_csv("Cars_specs.csv")
11
12 # Convert 'horsepower' to numeric and drop rows with missing values
13 df['horsepower'] = pd.to_numeric(df['horsepower'], errors='coerce')
14 df.dropna(subset=['horsepower', 'weight', 'mpg'], inplace=True)
15
16 # Standardize the features
17 scaler = StandardScaler()
18 features = df[['horsepower', 'weight', 'mpg']]
19 scaled_features = scaler.fit_transform(features)
20
21 # Sidebar - number of clusters and feature selection
22 st.sidebar.title("🕒 Clustering Settings")
23 n_clusters = st.sidebar.slider("Select number of clusters", min_value=2, max_value=10, value=3)
24 show_data = st.sidebar.checkbox("Show Raw Data")
25
26 # KMeans clustering
27 kmeans = KMeans(n_clusters=n_clusters, random_state=42)
28 df['cluster'] = kmeans.fit_predict(scaled_features)
29
30 # Dashboard title
31 st.title("🚗 Car Clustering Interactive Dashboard")
32 st.markdown("Use this interactive dashboard to explore how cars cluster based on horsepower, weight, and MPG.")
33
34 # Optional: show data
35 if show_data:
36     st.subheader("🕒 Raw Car Data with Cluster Labels")
37     st.dataframe(df[['car_name', 'horsepower', 'weight', 'mpg', 'cluster']])
38
39 # Tabs for visualizations
40 tab1, tab2, tab3 = st.tabs(["📊 Horsepower vs MPG", "⚖️ Weight vs MPG", "📍 Cluster Centroids"])
41
42 with tab1:
43     fig1, ax1 = plt.subplots()
44     sns.scatterplot(data=df, x='horsepower', y='mpg', hue='cluster', palette='Set2', ax=ax1, s=100)
45     plt.title("Horsepower vs MPG")

```

Dashboard:

