The University of Faisalabad

LAB MANUAL

COURSE CODE :	AI - 414
COURSE TITLE:	MACHINE LEARNING
SUBMITTED TO:	SIR SAEED
SUBMITTED BY:	TAIBAH SHAHBAZ
REGISTRATION NO:	2023-BS-AI-024
SECTION:	A
DEPARTMENT:	COMPUTER SCIENCE

TABLE OF CONTENTS

PROJECT 1

PREDICTING SELLING PRICE OF USED CARS USING REGRESSION

ABOUT DATASET	3
SUMMARY	3
OBJECTIVES	3
ABSTRACT	4
EXPLANATION OF STEPS	4
NOTEBOOK CODE SCREENSHOT	6
CONCLUSION	17

PROJECT 2

Classifying Most Streamed Spotify Songs 2023

ABOUT DATASET	18
SUMMARY	18
OBJECTIVES	18
ABSTRACT	19
EXPLANATION OF STEPS	19
NOTEBOOK CODE SCREENSHOT	21
CONCLUSION	28

PROJECT 1

PREDICTING SELLING PRICE OF USED CARS USING REGRESSION

About Dataset

Description:

The dataset in this project consists of detailed information about used cars, aimed at predicting their selling prices. Key features include the car's name, year of manufacture, current showroom price, kilometers driven, fuel type (Petrol, Diesel, or CNG), seller type (Dealer or Individual), transmission type (Manual or Automatic), and the number of previous owners. The target variable is the selling price of the car. These attributes together create a comprehensive picture of each vehicle, enabling robust regression modeling to forecast the car’s market value accurately.

Summary:

This project utilizes regression-based machine learning techniques to predict the selling prices of used cars. The dataset encompasses various attributes, such as brand, model, year of manufacture, mileage, engine details, fuel type, and transmission. The core aim is to develop a regression model that can accurately determine a vehicle's market value based on these features. By applying linear regression and related techniques, this project highlights how supervised learning with continuous target variables can be leveraged for practical, real-world scenarios like vehicle appraisal and pricing strategies. Ultimately, it demonstrates the power of data-driven decision-making in the automotive sector.

Objectives:

1. Predict car selling prices based on available features.
2. Preprocess the data (clean, remove outliers, transform features).
3. Train a regression model to fit the data.
4. Evaluate the model's performance using various metrics.
5. Visualize actual vs. predicted values for better understanding.

Abstract:

We perform data preprocessing steps (cleaning, outlier removal, scaling, label encoding), dimensionality reduction (PCA), and then apply linear regression to predict car prices. Key steps include loading data, EDA, cleaning (removing Car_Name and adjusting for Car_Age), normalization, PCA to retain 95% variance, training the model, and evaluating it using metrics like RMSE and R^2. Finally, we visualize actual vs. predicted prices.

Explanation of Each Step:

1. Importing Libraries

The essential Python libraries (pandas, numpy, seaborn, matplotlib) and scikit-learn tools are imported to handle data manipulation, visualization, scaling, model building, and evaluation.

2. Reading and Exploring the Data

The dataset (car data.csv) is read into a pandas DataFrame, and various commands (head(), tail(), shape, info(), describe()) provide insights into the dataset's structure, data types, and summary statistics.

3. Data Cleaning

Columns like Car_Name are removed, and a new Car_Age column is created based on the car's year. Missing values and duplicates are handled to ensure a clean dataset for analysis.

4. Outlier Detection and Removal

Boxplots are used to visualize outliers in numerical features, and the IQR (Interquartile Range) method is applied to filter out rows that lie outside of typical value ranges, ensuring more robust model performance.

5. Normalization and Encoding

Normalization scales numerical data to a consistent range (like 0-1 or standard scores), improving model performance and convergence.

Numerical features are scaled using StandardScaler to ensure consistency in model training. Categorical features are transformed into numerical values using LabelEncoder, enabling machine learning algorithms to work effectively.

6. Dimensionality Reduction (PCA)

Principal Component Analysis (PCA) is used to reduce feature dimensionality while preserving 95% of variance, which helps improve model efficiency and prevent overfitting.

7. Data Splitting and Model Training

The data is split into training and testing sets (80-20 split). A linear regression model is then trained on the training set, leveraging scikit-learn's LinearRegression class.

8. Prediction and Model Evaluation

Predictions are made on the test set, and actual vs. predicted values are compared. Metrics like MAE, MSE, RMSE, and R^2 are calculated to evaluate the model's performance.

9. Visualization

The final step involves plotting the actual vs. predicted selling prices to visually assess how well the model fits the data and highlight any major discrepancies.

Notebook Code Screenshot:

Importing libraries

[7]:
```
import matplotlib.pyplot as plt
import seaborn as sns
color = sns.color_palette()

import numpy as np
import pandas as pd

from sklearn.preprocessing import StandardScaler, LabelEncoder
from sklearn.decomposition import PCA
from sklearn.feature_selection import SelectKBest, f_regression
from sklearn.model_selection import train_test_split
```

Reading data

[9]:
```
data = pd.read_csv('car data.csv')
data.head()
```

[9]:

	Car_Name	Year	Selling_Price	Present_Price	Driven_kms	Fuel_Type	Selling_type	Transmission	Owner
0	ritz	2014	3.35	5.59	27000	Petrol	Dealer	Manual	0
1	sx4	2013	4.75	9.54	43000	Diesel	Dealer	Manual	0
2	ciaz	2017	7.25	9.85	6900	Petrol	Dealer	Manual	0
3	wagon r	2011	2.85	4.15	5200	Petrol	Dealer	Manual	0
4	swift	2014	4.60	6.87	42450	Diesel	Dealer	Manual	0

[10]:
```
data.tail()
```

[10]:

	Car_Name	Year	Selling_Price	Present_Price	Driven_kms	Fuel_Type	Selling_type	Transmission	Owner
296	city	2016	9.50	11.6	33988	Diesel	Dealer	Manual	0
297	brio	2015	4.00	5.9	60000	Petrol	Dealer	Manual	0
298	city	2009	3.35	11.0	87934	Petrol	Dealer	Manual	0
299	city	2017	11.50	12.5	9000	Diesel	Dealer	Manual	0
300	brio	2016	5.30	5.9	5464	Petrol	Dealer	Manual	0

[11]:
```
data.shape
```

[11]: (301, 9)

[12]:
```
data.sample()
```

[12]:

	Car_Name	Year	Selling_Price	Present_Price	Driven_kms	Fuel_Type	Selling_type	Transmission	Owner
113	Mahindra Mojo XT300	2016	1.15	1.4	35000	Petrol	Individual	Manual	0

[13]:
```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 301 entries, 0 to 300
Data columns (total 9 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   Car_Name       301 non-null    object
 1   Year           301 non-null    int64
 2   Selling_Price  301 non-null    float64
 3   Present_Price  301 non-null    float64
 4   Driven_kms     301 non-null    int64
 5   Fuel_Type      301 non-null    object
 6   Selling_type   301 non-null    object
 7   Transmission   301 non-null    object
 8   Owner          301 non-null    int64
dtypes: float64(2), int64(3), object(4)
memory usage: 21.3+ KB
```

[14]:
```
data.describe()
```

[14]:

	Year	Selling_Price	Present_Price	Driven_kms	Owner
count	301.000000	301.000000	301.000000	301.000000	301.000000
mean	2013.627907	4.661296	7.628472	36947.205980	0.043189
std	2.891554	5.082812	8.642584	38886.883882	0.247915
min	2003.000000	0.100000	0.320000	500.000000	0.000000
25%	2012.000000	0.900000	1.200000	15000.000000	0.000000
50%	2014.000000	3.600000	6.400000	32000.000000	0.000000
75%	2016.000000	6.000000	9.900000	48767.000000	0.000000
max	2018.000000	35.000000	92.600000	500000.000000	3.000000

Checking missing values

[16]:
```
data.isnull().sum()
```

[16]:
```
Car_Name         0
Year             0
Selling_Price    0
Present_Price    0
Driven_kms       0
Fuel_Type        0
Selling_type     0
Transmission     0
Owner            0
dtype: int64
```

Checking unique values

[18]:
```
data.nunique()
```

[18]:
```
Car_Name          98
Year              16
Selling_Price    156
Present_Price    148
Driven_kms       206
Fuel_Type          3
Selling_type       2
Transmission       2
Owner              3
dtype: int64
```

Cleansing Data

[20]:
```
data_cleaned = data.drop(columns=['Car_Name'])

data_cleaned['Car_Age'] = 2025 - data_cleaned['Year']

data_cleaned.drop(columns=['Year'], inplace=True)

data_cleaned.to_csv("car_data_cleaned.csv", index=False)

print("Data cleaning complete. File saved as 'car_data_cleaned.csv'")
```

```
Data cleaning complete. File saved as 'car_data_cleaned.csv'
```

Dropping missing values if any

[22]:
```
data.dropna()
```

[22]:

	Car_Name	Year	Selling_Price	Present_Price	Driven_kms	Fuel_Type	Selling_type	Transmission	Owner
0	ritz	2014	3.35	5.59	27000	Petrol	Dealer	Manual	0
1	sx4	2013	4.75	9.54	43000	Diesel	Dealer	Manual	0
2	ciaz	2017	7.25	9.85	6900	Petrol	Dealer	Manual	0
3	wagon r	2011	2.85	4.15	5200	Petrol	Dealer	Manual	0
4	swift	2014	4.60	6.87	42450	Diesel	Dealer	Manual	0
...	...	...	...	...	...	...	...	...	...
296	city	2016	9.50	11.60	33988	Diesel	Dealer	Manual	0
297	brio	2015	4.00	5.90	60000	Petrol	Dealer	Manual	0
298	city	2009	3.35	11.00	87934	Petrol	Dealer	Manual	0
299	city	2017	11.50	12.50	9000	Diesel	Dealer	Manual	0
300	brio	2016	5.30	5.90	5464	Petrol	Dealer	Manual	0

301 rows × 9 columns

Removing duplicate rows

[24]:
```
data.drop_duplicates()
```

[24]:

	Car_Name	Year	Selling_Price	Present_Price	Driven_kms	Fuel_Type	Selling_type	Transmission	Owner
0	ritz	2014	3.35	5.59	27000	Petrol	Dealer	Manual	0
1	sx4	2013	4.75	9.54	43000	Diesel	Dealer	Manual	0
2	ciaz	2017	7.25	9.85	6900	Petrol	Dealer	Manual	0
3	wagon r	2011	2.85	4.15	5200	Petrol	Dealer	Manual	0
4	swift	2014	4.60	6.87	42450	Diesel	Dealer	Manual	0
...	...	...	...	...	...	...	...	...	...
296	city	2016	9.50	11.60	33988	Diesel	Dealer	Manual	0
297	brio	2015	4.00	5.90	60000	Petrol	Dealer	Manual	0
298	city	2009	3.35	11.00	87934	Petrol	Dealer	Manual	0
299	city	2017	11.50	12.50	9000	Diesel	Dealer	Manual	0
300	brio	2016	5.30	5.90	5464	Petrol	Dealer	Manual	0

299 rows × 9 columns

Outlier Detection and Removing

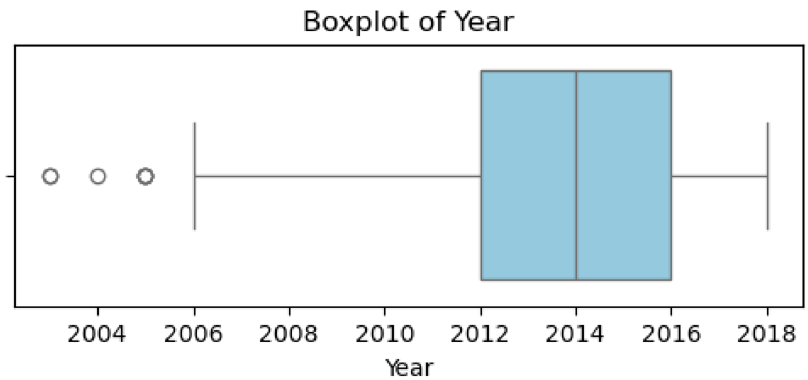
[26]:
```
0.25-1.5*0.5
```

[26]: -0.5

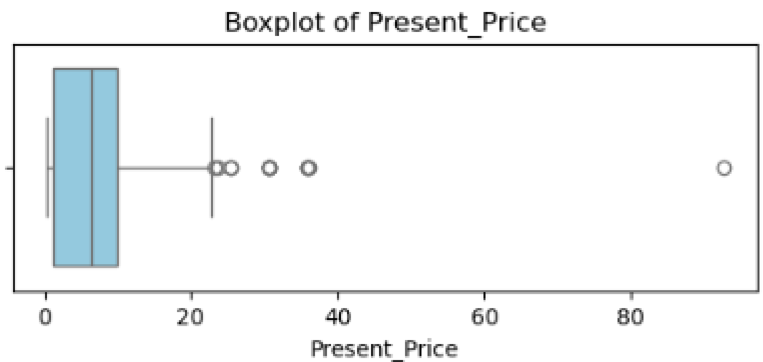
[27]:
```
0.75 + 1.5 * 0.5
```

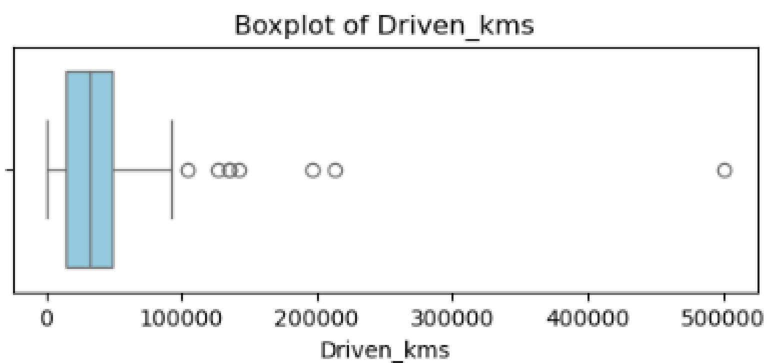
[27]: 1.5

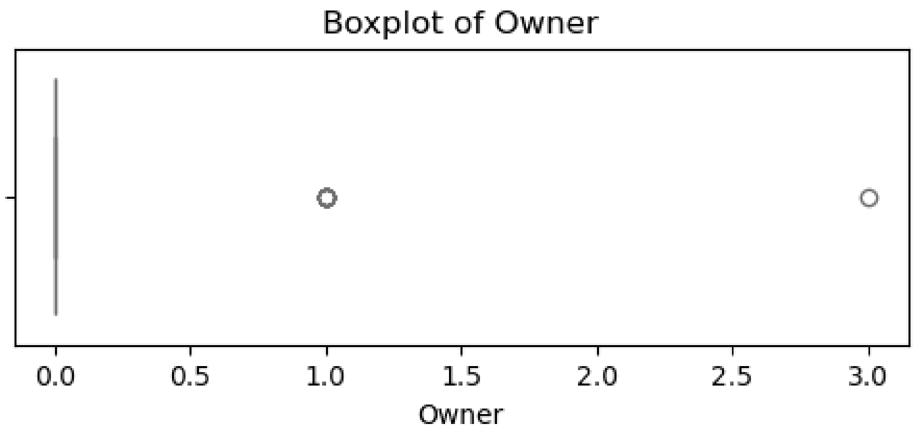
[28]:
```
numerical_cols = data.select_dtypes(include=[np.number]).columns.tolist()

for col in numerical_cols:
    plt.figure(figsize=(6, 2))
    sns.boxplot(x=data[col], color='skyblue')
    plt.title(f'Boxplot of {col}')
    plt.show()
```




Boxplot of Selling_Price
0
5
10
15
20
25
30
35
Selling_Price


Boxplot of Present_Price
0
20
40
60
80
Present_Price


Boxplot of Driven_kms
0
100000
200000
300000
400000
500000
Driven_kms



[29]:
```
for col in numerical_cols:
    Q1 = data[col].quantile(0.25)
    Q3 = data[col].quantile(0.75)
    IQR = Q3 - Q1
    lower_bound = Q1 - 1.5 * IQR
    upper_bound = Q3 + 1.5 * IQR
    data = data[(data[col] >= lower_bound) & (data[col] <= upper_bound)]

data.shape
```

[29]:
```
(261, 9)
```

Normalization

[31]:
```
numeric_cols = data.select_dtypes(include=['number']).columns

scaler = StandardScaler()

scaled_numeric_data = scaler.fit_transform(data[numeric_cols])

scaled_numeric_df = pd.DataFrame(scaled_numeric_data, columns=numeric_cols)

non_numeric_data = data.drop(columns=numeric_cols).reset_index(drop=True)

scaled_data = pd.concat([scaled_numeric_df, non_numeric_data], axis=1)

print(scaled_data.shape)
print('*' * 60)
print(scaled_data.head())
```

```
(261, 9)
************************************************************
       Year  Selling_Price  Present_Price  Driven_kms  Owner Car_Name  \
0  0.004882      -0.165821      -0.106661   -0.215719    0.0     ritz
1 -0.419827       0.288044       0.714504    0.580224    0.0      sx4
2  1.279008       1.098519       0.778950   -1.215623    0.0     ciaz
3 -1.269244      -0.327916      -0.406023   -1.300192    0.0  wagon r
4  0.004882       0.239416       0.159438    0.552863    0.0    swift

  Fuel_Type Selling_type Transmission
0    Petrol       Dealer       Manual
1    Diesel       Dealer       Manual
2    Petrol       Dealer       Manual
3    Petrol       Dealer       Manual
4    Diesel       Dealer       Manual
```

Categorical into Numerical

[33]:
```
categorical_cols = data.select_dtypes(include='object').columns.tolist()
categorical_cols
```

[33]:
```
['Car_Name', 'Fuel_Type', 'Selling_type', 'Transmission']
```

[34]:
```
le = LabelEncoder()
for col in categorical_cols:
    data[col] = le.fit_transform(data[col])
```

Dimensionality Reduction (PCA)

[36]:
```
X = data.drop('Selling_Price', axis=1)
y = data['Selling_Price']
```

[37]:
```
pca = PCA(0.95)
X_pca = pca.fit_transform(X)
```

[38]:
```
 X_pca.shape
```

[38]: (261, 1)

[39]:
```
plt.figure(figsize=(6,4))
sns.histplot(y, kde=True, color='green')
plt.title('Distribution of Selling Price')
plt.show()
```

Data Splitting

[41]:
```
X_selected = X
X_train, X_test, y_train, y_test = train_test_split(X_selected, y, test_size=0.2, random_state=42)

print("Training Set Shape:", X_train.shape)
print("Test Set Shape:", X_test.shape)
```

```
Training Set Shape: (208, 8)
Test Set Shape: (53, 8)
```

Train the Linear Regression Model

[43]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error

model = LinearRegression()
model.fit(X_train, y_train)
```

[43]:
```
▾ LinearRegression
LinearRegression()
```

Make Predictions

[76]:
```
y_pred = model.predict(X_test)
```

Compare Actual vs Predicted Values

[79]:
```
df_preds = pd.DataFrame({
    'Actual': y_test.squeeze(),
    'Predicted': y_pred.squeeze()
})

print("Actual vs Predicted:")
print(df_preds.head(10))
```

```
Actual vs Predicted:
      Actual  Predicted
0   5.037904   4.498754
1   8.865488   9.152736
2   7.396518   8.466439
3   7.065746   7.852142
4   6.343712   5.591731
5   6.942462   6.607172
6   5.757981   5.778528
7  11.044395   8.975125
8   5.038909   4.258159
9   6.334288   6.239831
```

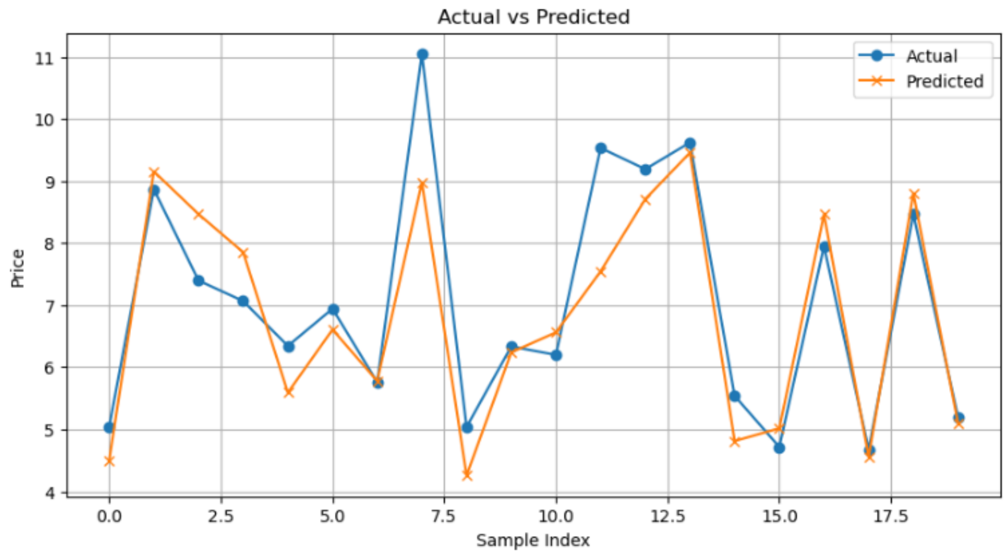
Evaluate the Model

[86]:
```
print("\nModel Evaluation:")
print("MAE:", mean_absolute_error(y_test, y_pred))
print("MSE:", mean_squared_error(y_test, y_pred))
print("RMSE:", mean_squared_error(y_test, y_pred, squared=False))
print("R² Score:", r2_score(y_test, y_pred))
```

```

Model Evaluation:
MAE: 0.5913425779189777
MSE: 0.6536995137170021
RMSE: 0.8085168605026132
R² Score: 0.8072059636181392
```

Plot Actual vs Predicted

```
[93]: plt.figure(figsize=(10, 5))
plt.plot(y_test, label='Actual', marker='o')
plt.plot(y_pred, label='Predicted', marker='x')
plt.title('Actual vs Predicted')
plt.xlabel('Sample Index')
plt.ylabel('Price')
plt.legend()
plt.grid(True)
plt.show()
```



Conclusion:

This project successfully demonstrated how regression-based machine learning techniques, specifically linear regression, can be effectively applied to predict the selling prices of used cars. By leveraging a comprehensive dataset containing various features (such as brand, model, year, mileage, engine details, fuel type, and transmission) we performed thorough data cleaning, transformation, and feature engineering. Principal Component Analysis (PCA) was employed to reduce dimensionality while maintaining data variance, and the final linear regression model was trained and evaluated on the refined dataset. The model's performance, visualizations, and evaluation metrics confirmed its capability to provide valuable insights into used car pricing, showcasing the potential of data-driven decision-making in real-world automotive applications.

PROJECT 2

Classifying Most Streamed Spotify Songs 2023

About Dataset

Description:

This dataset contains a comprehensive list of the most famous songs of 2023 as listed on Spotify. The dataset offers a wealth of features beyond what is typically available in similar datasets. It provides insights into each song's attributes, popularity, and presence on various music platforms. The dataset includes information such as track name, artist(s) name, release date, Spotify playlists and charts, streaming statistics, Apple Music presence, Deezer presence, Shazam charts, and various audio features.

Summary:

The project focuses on classifying Spotify song data using different machine learning models: Random Forest, Logistic Regression, and an Artificial Neural Network (ANN). The data is preprocessed by dropping irrelevant features, handling missing values, and scaling numeric features. Each model's performance is evaluated and compared, concluding with an analysis of confusion matrices.

Objectives:

1. Clean and preprocess Spotify data.
2. Split data into features and labels for training/testing.
3. Train and evaluate a Random Forest classifier.
4. Train and evaluate a Logistic Regression classifier.
5. Build and evaluate an ANN model.
6. Compare model performances using confusion matrices and accuracy scores.

Abstract:

The project utilizes the Spotify 2023 dataset to classify songs based on various features. After data cleaning and preparation, multiple classification models are trained and tested. Random

Forest and Logistic Regression serve as traditional classifiers, while an ANN explores deep learning. Each model's effectiveness is gauged using accuracy metrics and confusion matrices.

Explanation of Each Step:

1. Importing Libraries

Essential libraries like pandas, numpy, and scikit-learn are imported. These libraries provide the tools necessary for data manipulation, visualization, and building machine learning models.

2. Load and Preprocess Dataset

The dataset is loaded using pandas, and irrelevant columns (e.g., song names and artist names) are dropped to remove noise. Missing values are handled to ensure the dataset is clean and consistent. Numeric conversions are also performed for columns like streams.

3. Split Features and Labels

The dataset is divided into features (X) and labels (y). This separation ensures that the models are trained using only relevant features to predict the target variable.

4. Scale Features

StandardScaler from scikit-learn is used to standardize feature values to have zero mean and unit variance. This scaling step ensures that no feature dominates because of its magnitude and helps improve model performance.

5. Train Classification Models

Two classification models, Random Forest and Logistic Regression, are trained using the training dataset. These models are selected for their simplicity and effectiveness in structured classification problems.

6. Evaluate Models

The performance of each model is evaluated using metrics such as accuracy and detailed classification reports. Confusion matrices are also generated to analyze model predictions and understand misclassifications.

7. Artificial Neural Network (ANN)

A basic ANN is built using Keras' Sequential model. The network includes input, hidden, and output layers, and is compiled and trained on the training data to explore deep learning for classification.

8. Compare Results

Results from all models (Random Forest, Logistic Regression, and ANN) are compared using confusion matrices and accuracy scores. This comparison highlights which model performs best for this dataset.

Notebook Code Screenshot:

Importing libraries

```
[1]: import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import classification_report, accuracy_score
```

Load and Preprocess Dataset

[4]:
```
df = pd.read_csv("spotify-2023.csv", encoding="latin1")
```

[5]:
```
df.shape
```

[5]:
```
(953, 24)
```

[6]:
```
df.columns
```

[6]:
```
Index(['track_name', 'artist(s)_name', 'artist_count', 'released_year',
       'released_month', 'released_day', 'in_spotify_playlists',
       'in_spotify_charts', 'streams', 'in_apple_playlists', 'in_apple_charts',
       'in_deezer_playlists', 'in_deezer_charts', 'in_shazam_charts', 'bpm',
       'key', 'mode', 'danceability_%', 'valence_%', 'energy_%',
       'acousticness_%', 'instrumentalness_%', 'liveness_%', 'speechiness_%'],
      dtype='object')
```

[7]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 953 entries, 0 to 952
Data columns (total 24 columns):
 #   Column                Non-Null Count  Dtype
---  ------                --------------  -----
 0   track_name            953 non-null    object
 1   artist(s)_name        953 non-null    object
 2   artist_count          953 non-null    int64
 3   released_year         953 non-null    int64
 4   released_month        953 non-null    int64
 5   released_day          953 non-null    int64
 6   in_spotify_playlists  953 non-null    int64
 7   in_spotify_charts     953 non-null    int64
 8   streams               953 non-null    object
 9   in_apple_playlists    953 non-null    int64
 10  in_apple_charts       953 non-null    int64
 11  in_deezer_playlists   953 non-null    object
 12  in_deezer_charts      953 non-null    int64
 13  in_shazam_charts      903 non-null    object
 14  bpm                   953 non-null    int64
 15  key                   858 non-null    object
 16  mode                  953 non-null    object
```


```
 14  bpm                   953 non-null    int64
 15  key                   858 non-null    object
 16  mode                  953 non-null    object
```


```
 17  danceability_%        953 non-null    int64
 18  valence_%             953 non-null    int64
 19  energy_%              953 non-null    int64
 20  acousticness_%        953 non-null    int64
 21  instrumentalness_%    953 non-null    int64
 22  liveness_%            953 non-null    int64
 23  speechiness_%         953 non-null    int64
dtypes: int64(17), object(7)
memory usage: 178.8+ KB
```

[8]:
```
df.describe()
```

[8]:

	artist_count	released_year	released_month	released_day	in_spotify_playlists	in_spotify_charts	in_apple_playlists	in_apple_charts	in_deezer_charts	bpm	c
count	953.000000	953.000000	953.000000	953.000000	953.000000	953.000000	953.000000	953.000000	953.000000	953.000000	
mean	1.556139	2018.238195	6.033578	13.930745	5200.124869	12.009444	67.812172	51.908709	2.666317	122.540399	
std	0.893044	11.116218	3.566435	9.201949	7897.608990	19.575992	86.441493	50.630241	6.035599	28.057802	
min	1.000000	1930.000000	1.000000	1.000000	31.000000	0.000000	0.000000	0.000000	0.000000	65.000000	
25%	1.000000	2020.000000	3.000000	6.000000	875.000000	0.000000	13.000000	7.000000	0.000000	100.000000	
50%	1.000000	2022.000000	6.000000	13.000000	2224.000000	3.000000	34.000000	38.000000	0.000000	121.000000	
75%	2.000000	2022.000000	9.000000	22.000000	5542.000000	16.000000	88.000000	87.000000	2.000000	140.000000	
max	8.000000	2023.000000	12.000000	31.000000	52898.000000	147.000000	672.000000	275.000000	58.000000	206.000000	

[9]:
```
df.head()
```

[9]:

	track_name	artist(s)_name	artist_count	released_year	released_month	released_day	in_spotify_playlists	in_spotify_charts	streams	in_apple_playlists	...	bpm	k
0	Seven (feat. Latto) (Explicit Ver.)	Latto, Jung Kook	2	2023	7	14	553	147	141381703	43	...	125	
1	LALA	Myke Towers	1	2023	3	23	1474	48	133716286	48	...	92	
2	vampire	Olivia Rodrigo	1	2023	6	30	1397	113	140003974	94	...	138	
3	Cruel Summer	Taylor Swift	1	2019	8	23	7858	100	800840817	116	...	170	
4	WHERE SHE GOES	Bad Bunny	1	2023	5	18	3133	50	303236322	84	...	144	

5 rows × 24 columns

[10]:
```
columns_to_drop = ['track_name', 'artist(s)_name', 'released_year', 'released_month', 'released_day']
df.drop(columns=[col for col in columns_to_drop if col in df.columns], inplace=True)
```

[11]:
```
df.dropna(inplace=True)
```

[12]:
```
df['streams'] = pd.to_numeric(df['streams'].str.replace(',', ''), errors='coerce')
df.columns = df.columns.str.strip().str.lower()
print(df.columns.tolist())
```

```
['artist_count', 'in_spotify_playlists', 'in_spotify_charts', 'streams', 'in_apple_playlists', 'in_apple_charts', 'in_deezer_playlists', 'in_deezer_chart
s', 'in_shazam_charts', 'bpm', 'key', 'mode', 'danceability_%', 'valence_%', 'energy_%', 'acousticness_%', 'instrumentalness_%', 'liveness_%', 'speechine
ss_%']
```

[13]:
```
df.dropna(subset=['streams'], inplace=True)
threshold = df['streams'].quantile(0.75)
df['is_popular'] = (df['streams'] >= threshold).astype(int)
df.drop(columns='streams', inplace=True)

df = pd.get_dummies(df, drop_first=True)

df.dropna(inplace=True)

print(df.columns)
```

```
Index(['artist_count', 'in_spotify_playlists', 'in_spotify_charts',
       'in_apple_playlists', 'in_apple_charts', 'in_deezer_charts', 'bpm',
       'danceability_%', 'valence_%', 'energy_%',
       ...
       'key_B', 'key_C#', 'key_D', 'key_D#', 'key_E', 'key_F', 'key_F#',
       'key_G', 'key_G#', 'mode_Minor'],
      dtype='object', length=503)
```

Split Features and Labels

[15]:
```
from sklearn.model_selection import train_test_split

X = df.drop(columns=['is_popular'])
y = df['is_popular']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

Scale Features

[17]:
```
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

Train Classification Model

[19]:
```
model = RandomForestClassifier(random_state=42)
model.fit(X_train, y_train)
```

[19]:
```
RandomForestClassifier
RandomForestClassifier(random_state=42)
```

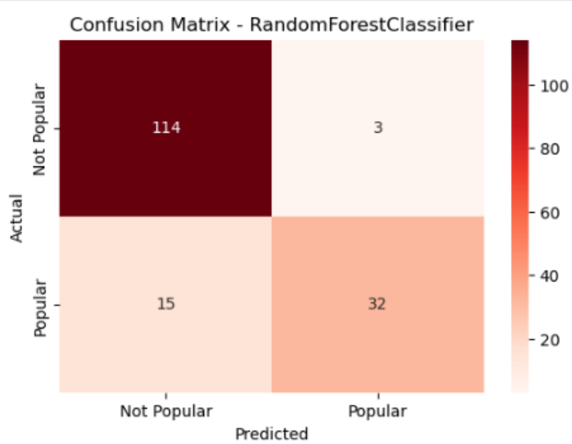
Evaluate Model

[21]:
```
y_pred = model.predict(X_test)

print("Accuracy:", accuracy_score(y_test, y_pred))
print("Classification Report:\n", classification_report(y_test, y_pred))
```

```
Accuracy: 0.8902439024390244
Classification Report:
               precision    recall  f1-score   support

           0       0.88      0.97      0.93       117
           1       0.91      0.68      0.78        47

    accuracy                           0.89       164
   macro avg       0.90      0.83      0.85       164
weighted avg       0.89      0.89      0.88       164
```

Confusion Matrix

[58]:
```
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.metrics import confusion_matrix

cm = confusion_matrix(y_test, y_pred)

plt.figure(figsize=(6, 4))
sns.heatmap(cm, annot=True, fmt='d', cmap='Reds', xticklabels=['Not Popular', 'Popular'], yticklabels=['Not Popular', 'Popular'])
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix - RandomForestClassifier')
plt.show()
```



[24]:
```python
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn.metrics import classification_report, confusion_matrix, accuracy_score
```

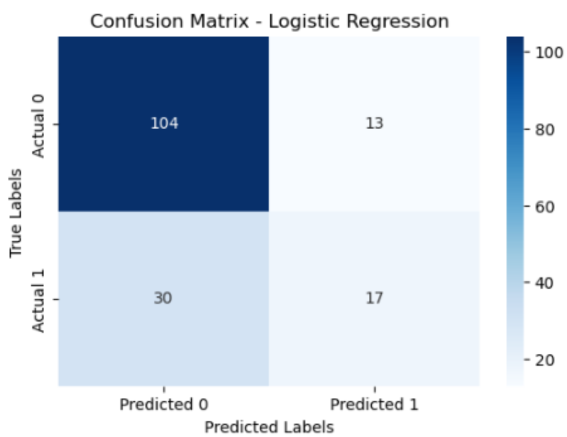
Logistic Regression Classifier

[26]:
```python
log_model = LogisticRegression()
log_model.fit(X_train_scaled, y_train)
log_preds = log_model.predict(X_test_scaled)
```

[27]:
```python
print("Logistic Regression:")
print("Accuracy:", accuracy_score(y_test, log_preds))
print(confusion_matrix(y_test, log_preds))
print(classification_report(y_test, log_preds))
```

```
Logistic Regression:
Accuracy: 0.7378048780487805
[[104  13]
 [ 30  17]]
              precision    recall  f1-score   support

           0       0.78      0.89      0.83       117
           1       0.57      0.36      0.44        47

    accuracy                           0.74       164
   macro avg       0.67      0.63      0.64       164
weighted avg       0.72      0.74      0.72       164
```

[56]:
```python
cm = confusion_matrix(y_test, log_preds)
plt.figure(figsize=(6,4))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=['Predicted 0', 'Predicted 1'], yticklabels=['Actual 0', 'Actual 1'])
plt.xlabel('Predicted Labels')
plt.ylabel('True Labels')
plt.title('Confusion Matrix - Logistic Regression')
plt.show()
```



Artificial Neural Network (ANN)

[29]:
```
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
```

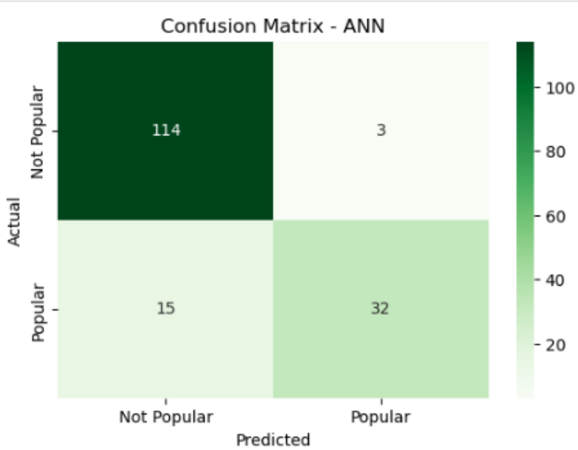
[30]:
```
import numpy as np

X_train = np.array(X_train).astype('float32')
y_train = np.array(y_train).astype('int32')

X_test_scaled = np.array(X_test_scaled).astype('float32')
y_test = np.array(y_test).astype('int32')

model = Sequential()
model.add(Dense(64, activation='relu'))
model.add(Dense(32, activation='relu'))
model.add(Dense(1, activation='sigmoid'))

model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
model.fit(X_train, y_train, epochs=20, batch_size=32, validation_split=0.2)

y_pred_ann = (model.predict(X_test_scaled) > 0.5).astype(int)

print("\nArtificial Neural Network")
print("Accuracy:", accuracy_score(y_test, y_pred_ann))
print(confusion_matrix(y_test, y_pred_ann))
print(classification_report(y_test, y_pred_ann))
```

```
Epoch 1/20
17/17 ━━━━━━━━━━━━━━━━━━━━ 5s 36ms/step - accuracy: 0.5108 - loss: 53.3049 - val_accuracy: 0.7328 - val_loss: 14.8772
Epoch 2/20
17/17 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - accuracy: 0.7752 - loss: 10.9919 - val_accuracy: 0.8702 - val_loss: 1.4928
Epoch 3/20
17/17 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - accuracy: 0.8725 - loss: 2.3329 - val_accuracy: 0.8855 - val_loss: 1.6298
```

```
Epoch 6/20
17/17 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - accuracy: 0.9006 - loss: 0.9760 - val_accuracy: 0.8931 - val_loss: 0.9897
Epoch 7/20
17/17 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - accuracy: 0.8799 - loss: 0.8245 - val_accuracy: 0.8321 - val_loss: 1.3237
Epoch 8/20
17/17 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - accuracy: 0.8746 - loss: 1.1064 - val_accuracy: 0.7328 - val_loss: 2.2332
Epoch 9/20
17/17 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - accuracy: 0.8435 - loss: 1.2625 - val_accuracy: 0.8855 - val_loss: 0.7527
Epoch 10/20
17/17 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - accuracy: 0.8503 - loss: 1.1166 - val_accuracy: 0.6183 - val_loss: 6.6870
Epoch 11/20
17/17 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - accuracy: 0.7663 - loss: 3.0710 - val_accuracy: 0.8855 - val_loss: 1.4928
Epoch 12/20
17/17 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - accuracy: 0.8532 - loss: 1.8900 - val_accuracy: 0.9008 - val_loss: 1.0511
Epoch 13/20
17/17 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - accuracy: 0.8961 - loss: 0.9759 - val_accuracy: 0.9008 - val_loss: 0.8516
Epoch 14/20
17/17 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - accuracy: 0.9154 - loss: 0.6262 - val_accuracy: 0.8168 - val_loss: 1.4550
Epoch 15/20
17/17 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - accuracy: 0.8121 - loss: 2.5317 - val_accuracy: 0.6718 - val_loss: 4.2064
Epoch 16/20
17/17 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - accuracy: 0.8053 - loss: 2.6396 - val_accuracy: 0.8779 - val_loss: 1.0991
Epoch 17/20
17/17 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - accuracy: 0.8835 - loss: 1.3866 - val_accuracy: 0.9008 - val_loss: 1.0481
Epoch 18/20
17/17 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - accuracy: 0.8765 - loss: 1.3000 - val_accuracy: 0.8931 - val_loss: 0.8742
Epoch 19/20
17/17 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 0.8667 - loss: 1.2941 - val_accuracy: 0.8855 - val_loss: 0.7707
Epoch 20/20
17/17 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - accuracy: 0.8579 - loss: 1.0737 - val_accuracy: 0.8626 - val_loss: 0.8384
6/6 ━━━━━━━━━━━━━━━━━━━━ 0s 24ms/step

Artificial Neural Network
Accuracy: 0.5304878048780488
[[55 62]
 [15 32]]
              precision    recall  f1-score   support

           0       0.79      0.47      0.59       117
           1       0.34      0.68      0.45        47

    accuracy                           0.53       164
   macro avg       0.56      0.58      0.52       164
weighted avg       0.66      0.53      0.55       164
```

Confusion Matrix - ANN

[60]:
```python
cm = confusion_matrix(y_test, y_pred)

plt.figure(figsize=(6, 4))
sns.heatmap(cm, annot=True, fmt='d', cmap='Greens', xticklabels=['Not Popular', 'Popular'], yticklabels=['Not Popular', 'Popular'])
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix - ANN')
plt.show()
```



Conclusion

The project effectively demonstrates how to clean, preprocess, and classify song data using various machine learning models. Random Forest and Logistic Regression yield competitive accuracy, while the ANN offers a deep learning perspective. Comparative confusion matrices reveal classification performance, showcasing strengths and limitations of each approach.