LAB 2 : Data Processioning

Student name: Donfack Tsopfack Yves Dylane

INTRODUCTION:

The goal of this project is to demonstrate the process of cleaning, transforming, and modeling data using Logistic Regression and Random Forest. The dataset used is the Titanic dataset from Seaborn, and the final models are evaluated using accuracy, precision, recall, and F1 score.

1. Data Collection

In this step, we load the Titanic dataset from Seaborn. This dataset contains information about passengers on the Titanic, including whether they survived, their age, class, and other features.

```
Lab3.py ×

import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.preprocessing import MinMaxScaler
import pandas as pd
from sklearn.ensemble import RandomForestClassifier

from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score
from sklearn.ensemble import RandomForestClassifier

# Step 1 Data Collection
titanic_data = sns.load_dataset('titanic')
print(titanic_data.head())
```

Figure 1: importing necesary liberies and the data for the work

Result:

The dataset contains the following columns: `survived`, `pclass`, `sex`, `age`, `sibsp`, `parch`, `fare`, `embarked`, `who`, `class`, `alive`, `alone`, and `embark_town`.

```
Run   Lab3 ×

/home/yveane/Documents/school/ai/customer/pythonProject/.venv/bin/python /home/yveane/Documents/school/ai/customer/pythonProject/Lab3/Lab3.py
   survived  pclass     sex   age  ...  deck  embark_town  alive  alone
0         0       3    male  22.0  ...   NaN  Southampton     no  False
1         1       1  female  38.0  ...     C    Cherbourg    yes  False
2         1       3  female  26.0  ...   NaN  Southampton    yes   True
3         1       1  female  35.0  ...     C  Southampton    yes  False
4         0       3    male  35.0  ...   NaN  Southampton     no   True
```

Figure 2: Head values of our data set

2. Data Cleaning

2.1 Inspecting for Missing Values

We checked for any missing values in the dataset to identify the columns that need imputation or removal.

```python
# Step 2 Data Cleaning
missing_values = titanic_data.isnull().sum() # to sum all missing values in the diferent colums
print("Missing values in each column:\n", missing_values)

# Drop columns that have too many missing values (like 'deck')
titanic_data_cleaned = titanic_data.drop(columns=['deck'])

# Impute missing values in 'age' with the median
titanic_data_cleaned['age'] = titanic_data_cleaned['age'].fillna(titanic_data_cleaned['age'].median())

# Impute 'embark_town' with the most frequent value (mode)
titanic_data_cleaned['embark_town'] = titanic_data_cleaned['embark_town'].fillna(titanic_data_cleaned['embark_town'].mode()[0])

# Dropping rows with missing 'embarked' since they are not many
titanic_data_cleaned = titanic_data_cleaned.dropna(subset=['embarked'])

# Verify if all missing values have been remove or replace with approprite values
print("Missing values after cleaning:\n", titanic_data_cleaned.isnull().sum())
```

Figure 3: handling missing values

Result:

The columns `age`, `embarked`, and `deck` contained missing values, which are needed to be handled.

```
Missing values in each column:
 survived         0
pclass          0
sex             0
age           177
sibsp           0
parch           0
fare            0
embarked        2
class           0
who             0
adult_male      0
deck          688
embark_town     2
alive           0
alone           0
dtype: int64
```

Figure 4: missing values

2.2 Handling Missing Values

For columns with missing numerical data like `age`, we used median imputation. For categorical columns like `embarked`, we used the mode (most frequent value) to fill missing entries.

Result:
After imputing missing values, all missing data were handled appropriately.

```
Missing values after cleaning:
 survived      0
pclass         0
sex            0
age            0
sibsp          0
parch          0
fare           0
embarked       0
class          0
who            0
adult_male     0
embark_town    0
alive          0
alone          0
dtype: int64
```

Figure 5: missing values after cleaning

3. Feature Selection and One-Hot Encoding

3.1 One-Hot Encoding Categorical Variables

To prepare the data for machine learning, we performed one-hot encoding on categorical variables like `sex`, `embarked`, and others. This allows the model to work with numerical data. This is done after handling the outliers and help us to plot a correlation graph for the features.

```
# One-hot encoding for categorical variables since corellation only work with numeric values
titanic_data_encoded = pd.get_dummies(titanic_data_cleaned,
                                      columns=['sex', 'embarked', 'class', 'who', 'embark_town', 'alive'],
                                      drop_first=True)
# Correlation analysis to select important features
plt.figure(figsize=(10, 6))
sns.heatmap(titanic_data_encoded.corr(), annot=True, cmap='coolwarm')
plt.title('Feature Correlation Matrix')
plt.show() # show corellation between the features

```

Figure 6: hot encoding categorical values

Result:

The dataset is now encoded with binary columns for categorical features, and the shape of the dataset has increased due to the additional columns. This permit the correlation graph to be plot.

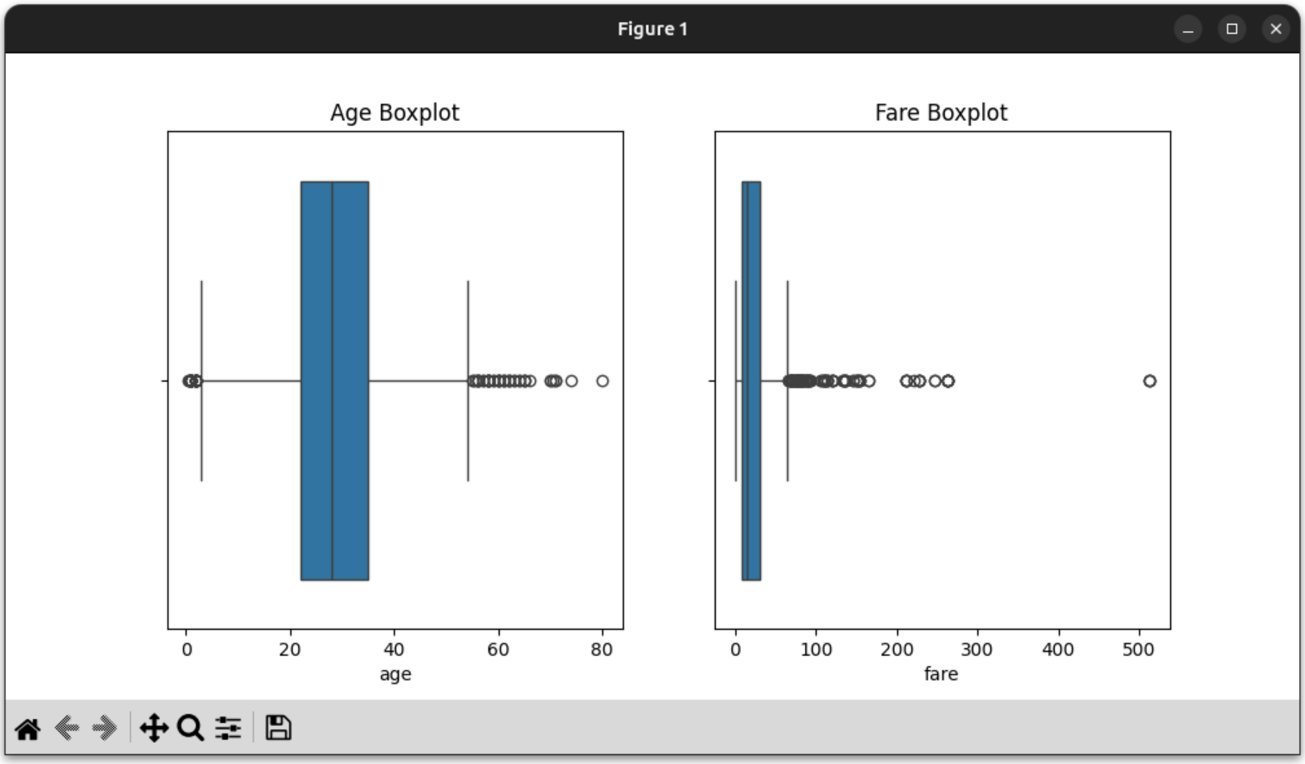

Figure 7: box plot to identify outliers

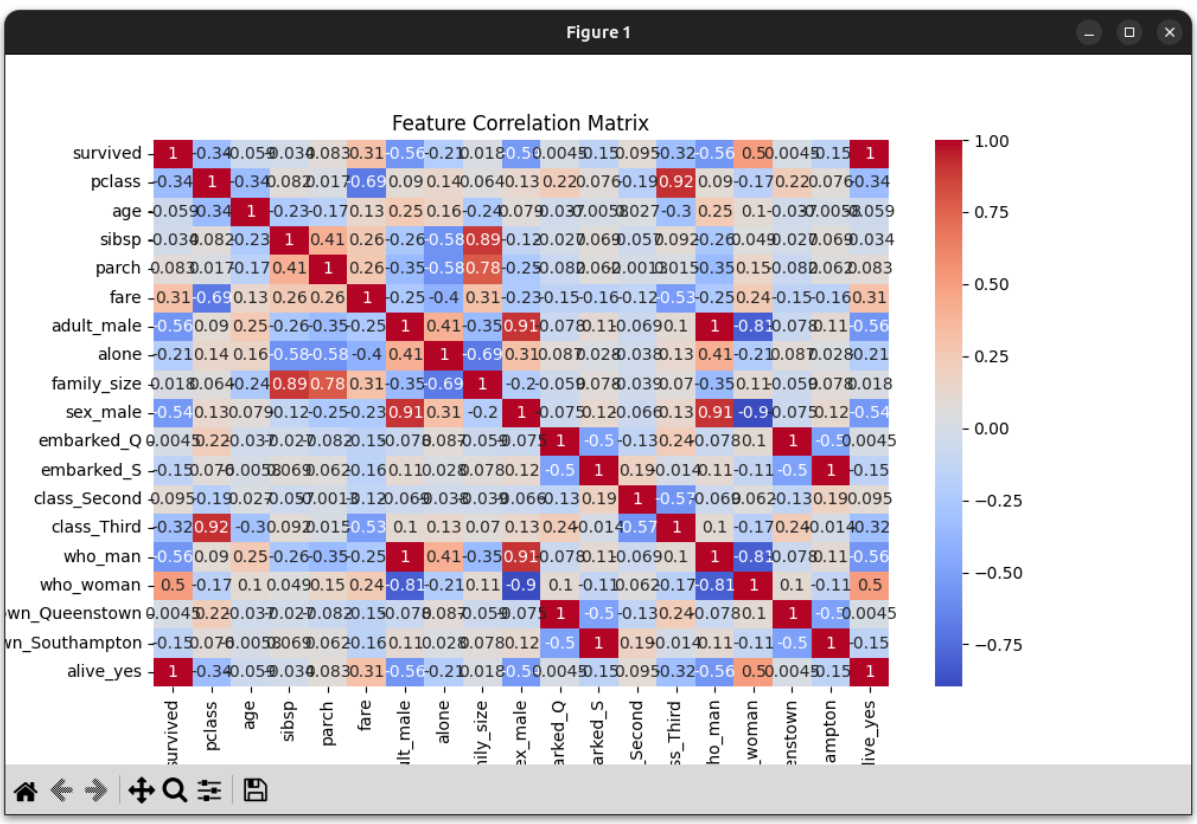

Figure 8: corellation graph of the featuers

4. Data Normalization

We normalized the numerical features such as `age` and `fare` to ensure that the scales of the features did not disproportionately affect the model's performance (i.e we give them the same scale of measurement).

```
scaler = MinMaxScaler() # Min-Max scaling brings 'age' and 'fare' values into a range between 0 and 1 for normalizing it scale

# Normalize 'age' and 'fare'
titanic_data_cleaned[['age', 'fare']] = scaler.fit_transform(titanic_data_cleaned[['age', 'fare']])

# Check normalized values
titanic_data_cleaned[['age', 'fare']].head()
```

Figure 9: normalizing data

Result:
The numerical columns `age` and `fare` are now scaled between 0 and 1.

```
        age      fare
0  0.333333  0.064549
1  0.666667  0.634654
2  0.416667  0.070558
3  0.604167  0.472763
4  0.604167  0.071671
```

Figure 10: normalise data

5. Feature Engineering

We created new features that might provide additional insights for the model and separate features and target.

5.1 Family Size Feature

We created a `family_size` feature by summing `sibsp` and `parch` columns, which represent the number of siblings/spouses and parents/children aboard.

```
titanic_data_cleaned['family_size'] = titanic_data_cleaned['sibsp'] + titanic_data_cleaned['parch'] + 1 # create family_size column
# Check the new feature by printing it values
print(titanic_data_cleaned[['family_size']].head())
```

Figure 11: creating family size

Result:

The `family_size` feature is now added to the dataset.

```
   family_size
0            2
1            2
2            1
3            2
4            1
```

Figure 12: head of the family size column

6. Model Building

6.1 Splitting Data into Train and Test Sets

We split the dataset into features (`X`) and target (`y`), where `y` is the `survived` column, and `X` contains the rest of the features. Then, we split the data into training and testing sets.

```python
# Split the data into train and test sets (same for both models)
X_train, X_test, y_train, y_test = train_test_split(*arrays: X, y, test_size=0.3, random_state=42)
```

Figure 13: spliting data

The data has being successfully split into training and testing sets with 30% of the data reserved for testing.

6.2 Logistic Regression Model

We first trained a Logistic Regression model and evaluated its performance using accuracy, precision, recall, and F1 score.

```python
# ----- Logistic Regression -----
# Initialize and fit Logistic Regression model
logreg = LogisticRegression(max_iter=1000)
logreg.fit(X_train, y_train)

# Make predictions
y_pred_logreg = logreg.predict(X_test)

# Evaluate the Logistic Regression model
accuracy_logreg = accuracy_score(y_test, y_pred_logreg)
precision_logreg = precision_score(y_test, y_pred_logreg)
recall_logreg = recall_score(y_test, y_pred_logreg)
f1_logreg = f1_score(y_test, y_pred_logreg)
```

Figure 14: logistic regression

Result:

The Logistic Regression model achieves the following performance metrics:

```
Logistic Regression Model Results:
Accuracy: 0.79
Precision: 0.72
Recall: 0.72
F1 Score: 0.72
```

Figure 15: logistic model result

6.3 Random Forest Model

We then trained a Random Forest model and compared its performance to that of Logistic Regression.

```python
rf_model = RandomForestClassifier(n_estimators=100, random_state=42)

# Fit the model on the training data
rf_model.fit(X_train, y_train)

# Make predictions
y_pred_rf = rf_model.predict(X_test)

# Evaluate the Random Forest model
accuracy_rf = accuracy_score(y_test, y_pred_rf)
precision_rf = precision_score(y_test, y_pred_rf)
recall_rf = recall_score(y_test, y_pred_rf)
f1_rf = f1_score(y_test, y_pred_rf)
```

Figure 16: random forest model

Result:

The Random Forest model achieves the following performance metrics:

```
Run    Lab3

Random Forest Model Results:
Accuracy: 0.78
Precision: 0.74
Recall: 0.65
F1 Score: 0.69
----------------------------------------
```

Figure 17: random forest result

7. Model Comparison

The following table compares the performance of Logistic Regression and Random Forest:

Metric	Logistic Regression	Random Forest
Accuracy	0.79	0.78
Precision	0.72	0.74
Recall	0.72	0.65
F1 Score	0.72	0.69

Conclusion:

- Logistic Regression performed better in terms of recall and F1 score, meaning it captured more true churners and balanced both precision and recall more effectively.
- Random Forest had slightly higher precision but lower recall, meaning it was better at avoiding false positives but missed some true churners.

Conclusion

This lab demonstrated the steps of processioning data, including cleaning, encoding, and normalizing, as well as building machine learning models for classification. Both Logistic Regression and Random Forest models were trained and evaluated, providing insights into their performance on the Titanic dataset.