The Superior University, Lahore

Assignment-I (Fall 2023)

Course Title:	Programming for AI				Course Code:	CAI601410	Credit Hours: 4
Instructor:	Prof. Rasikh Ali				Programme Name:	BSDS	
Semester:	4th	Batch:	F23	Section:	BSDSM-4A	Date:	1st February, 2025
Time Allowed:					Maximum Marks:		
Student's Name:	Asaad Iqbal				Reg. No.	SU92-BSDSM-F23-020	

Lab-Task 2

1: Spaceship titanic

Task 1

Spaceship Titanic Model Documentation

Introduction

This document provides a step-by-step guide for the Spaceship Titanic Kaggle competition machine learning model. The goal of this competition is to predict whether a passenger was "Transported" based on a set of features such as demographic information and passenger details. The model uses **Random Forest Classifier** to predict whether a passenger was transported based on these features. The approach avoids using a validation split and instead focuses on preprocessing data, handling missing values, encoding categorical variables, and training the model on the full dataset.

Step 1: Importing Libraries

```
[7]: import pandas as pd
     from sklearn.preprocessing import StandardScaler, OneHotEncoder
     from sklearn.compose import ColumnTransformer
     from sklearn.pipeline import Pipeline
     from sklearn.ensemble import RandomForestClassifier
```

- **pandas**: For handling data and reading CSV files.
- **StandardScaler** and **OneHotEncoder**: For preprocessing numerical and categorical data, respectively.
- **ColumnTransformer**: Allows applying different preprocessing steps to different subsets of features.
- **Pipeline**: Combines the preprocessing steps with the machine learning model in a streamlined workflow.

- **RandomForestClassifier**: A classification model used for predicting the target variable (Transported).

Step 2: Loading dataset

```
[8]: df = pd.read_csv('sample_submission.csv')
     df1 = pd.read_csv('train.csv')
     df2 = pd.read_csv('test.csv')
     print("All datasets are loaded")

     All datasets are loaded
```

- **train.csv**: The training dataset containing both features and target labels (Transported).
- **test.csv**: The test dataset with passenger details but without the target labels.
- **sample_submission.csv**: Provides a template for the submission file.
- The .head() method is used to inspect the first five rows of each dataset:
- df.head(), df1.head(), and df2.head() display the first five rows.
- .info() provides details about each dataset's columns, data types, and missing values.

Copying data for preprocessing

```
[9]: encoded_df1 = df1.copy()
     encoded_df2 = df2.copy()
```

Step 3: Handle missing values

```
[12]: def fill_missing_values(data):
          for col in data.columns:
              if data[col].dtype == 'O':
                  mode_value = data[col].mode()[0]
                  data[col] = data[col].fillna(mode_value).astype(str)
              elif data[col].dtype == 'float64':
                  data[col] = data[col].fillna(data[col].mean())
              elif data[col].dtype == 'int64':
                  data[col] = data[col].fillna(data[col].median())
```

```
[13]: fill_missing_values(encoded_df1)
      fill_missing_values(encoded_df2)
```

The **fill_missing_values()** function is defined to fill missing values in the dataset:

- **Categorical columns** (dtype 'O') are filled with the **mode** (most frequent value).
- **Float columns** (float64) are filled with the **mean** of the column.
- **Integer columns** (int64) are filled with the **median** of the column.

The function is applied to both the training (df1) and test (df2) datasets to ensure all missing values are handled appropriately.

Step 4: Selecting features and target variables

```
[14]: target = 'Transported'
      drop_cols = ['PassengerId', 'Name', target]
```

```
[15]: X = encoded_df1.drop(columns=drop_cols)
      y = encoded_df1[target]
```

- The target variable is **Transported**, which indicates whether the passenger was transported.
- The following columns are dropped from the dataset as they do not contribute to the prediction:
    - **PassengerId**: The unique identifier for each passenger.
    - **Name**: Passenger names are irrelevant to the prediction.

The remaining columns in the training dataset are stored in X, while the target variable is stored in y.

- For the test dataset (df2), columns such as PassengerId and Name are dropped, and the dataset is prepared for predictions.

Step 5: Identify categorical and numerical columns

```
[16]: X_test = encoded_df2.drop(columns=['PassengerId', 'Name'])
```

```
[17]: categorical_cols = X.select_dtypes(include=['object']).columns
      numerical_cols = X.select_dtypes(include=['int64', 'float64']).columns
```

The dataset contains categorical variables that need to be encoded into numerical format.
OneHotEncoder is used to encode categorical variables in the X and test datasets.

The **ColumnTransformer** is used to apply different preprocessing steps:

- **Numerical columns** are scaled using **StandardScaler**.
- **Categorical columns** are encoded using **OneHotEncoder** with the parameter handle_unknown='ignore' to handle unseen categories in the test set.

Step 6: Making a pipeline

Preprocessing pipeline

```
[23]: preprocessor = ColumnTransformer(
          transformers=[
              ('num', StandardScaler(), numerical_cols),
              ('cat', OneHotEncoder(handle_unknown='ignore'), categorical_cols)
          ])
```

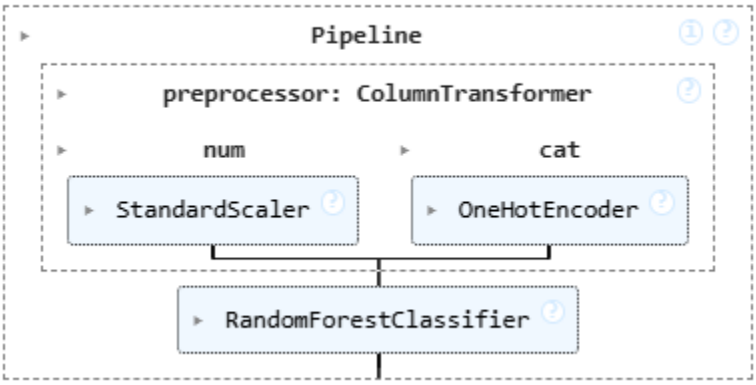
Model in a pipeline

```
[29]: model = Pipeline([
          ('preprocessor', preprocessor),
          ('classifier', RandomForestClassifier(random_state=42))
      ])
```

Step 7: Training the model on that pipeline

```
[30]: model.fit(X, y)
```

[30]:



After preprocessing, the model is trained using **RandomForestClassifier**, which is a robust classifier using **100 trees (estimators)**.

The model is trained on the entire dataset without splitting it into training and validation sets. This decision avoids the need for a validation split and instead trains on all available data.

Step 8: Preparing test data for prediction

```
[32]: test_predictions = model.predict(X_test)
```

The test dataset is prepared by selecting the same features as the training dataset. The **ExterCond** categorical feature is transformed using **LabelEncoder** to ensure compatibility with the trained model.

Step 9: Saving Predictions to csv

```
[33]: submission = pd.DataFrame({
          'PassengerId': df2['PassengerId'],
          'Transported': test_predictions
      })
      submission['Transported'] = submission['Transported'].map({True: 'True', False: 'False'})
      submission.to_csv('submission.csv', index=False)
      print("Submission file saved as 'submission.csv'")

      Submission file saved as 'submission.csv'
```

After making predictions, a new **DataFrame** is created containing the following columns:

- **PassengerId**: The unique passenger identifier from the test dataset.
- **Transported**: The predicted transport status (True or False).

The predictions are saved to a CSV file called **submission1.csv**, which adheres to the competition's submission format.

Kaggle Competition Result