Lab Task 2 (Report)

Task 2: Kaggle Spaceship Titanic Competition **-**

**1. Introduction**

**Objective**: Predict whether passengers were "Transported" (saved) using a machine learning model.

**Code Explanation:**

**Step 1: Importing Libraries**

* + We Import the necessary libraries, such as for data handling (pandas), for splitting data (train\_test\_split), encoding string into numericals (LabelEncoder), and training a classification model (RandomForestClassifier).
  + Libraries simplify complex tasks (e.g., pandas for data tables, sklearn for machine learning).
  + I Use import statements to load modules.

**Step 2: Load Data**

train\_data = pd.read\_csv("/content/train2.csv")

test\_data = pd.read\_csv("/content/test2.csv")

* + Load training and test datasets into structured tables (DataFrames).
  + Data is stored in CSV files, and pandas reads them into a format Python can process.
  + Use pd.read\_csv() to load CSV files from their file paths.

**Data Exploration**

**1. head() Function**

**train\_data.head()**

* + Displays the first 5 rows of the dataset**.**
  + Quickly inspect the structure of the data (columns, sample values).
  + Verify if data is loaded correctly (e.g., no gibberish values).

**train\_data.tail()**

* + Displays the last 5 rows of the dataset.
  + Check if the dataset is loaded completely (no truncation at the end).
  + Identify inconsistencies in the final rows (e.g., unexpected missing values).

**train\_data.info()**

* + Summarizes the dataset’s structure:
    - Number of rows and columns.
    - Column names and data types (e.g., int64, object).
    - Number of non-null values (helps spot missing data).
  + Verify data types (e.g., CryoSleep should be bool, not object).
  + Identify columns with many missing values (e.g., Age has 100 missing values).
  + Counts missing values (NaN) in each column.
  + Quantify missing data to decide how to handle it (e.g., drop rows, impute values).
  + Identify critical columns with high missingness (e.g., Cabin might have 20% missing).

**Insights:**

1. **Data Structure:**
   * Columns like HomePlanet (categorical) and Age (numerical) are identified.
   * Target variable: Transported (boolean).
2. **Missing Values:**
   * Columns like Cabin and Age require imputation.
3. **Data Types:**
   * Ensure categorical columns (e.g., HomePlanet) are encoded later.

**Step 3: Handle Missing Values**

num\_cols = ["Age", "RoomService", "Spa"]

for col in num\_cols:

train\_median = train\_data[col].median()

train\_data[col].fillna(train\_median, inplace=True)

test\_data[col].fillna(train\_median, inplace=True)

cat\_cols = ["HomePlanet", "CryoSleep"]

for col in cat\_cols:

train\_mode = train\_data[col].mode()[0]

train\_data[col].fillna(train\_mode, inplace=True)

test\_data[col].fillna(train\_mode, inplace=True)

* + Replace missing values (NaN) with the median for numbers and the most frequent value (mode) for categories.
  + Models cannot handle missing values. Using median/mode preserves data distribution without bias.
  + For numerical columns: Calculate the median from the **training data** and apply it to both datasets.
  + For categorical columns: Use the mode (most common category) from the training data.

**Step 4: Encode Categorical Data**

python

Copy

encoder = LabelEncoder()

for col in ["HomePlanet", "CryoSleep"]:

combined\_data = list(train\_data[col]) + list(test\_data[col])

encoder.fit(combined\_data)

train\_data[col] = encoder.transform(train\_data[col])

test\_data[col] = encoder.transform(test\_data[col])

* + Convert text-based categories (e.g., "Earth", "Mars") into numerical labels (e.g., 0, 1).
  + Machine learning models require numerical inputs.
  + Combine training and test data for each column to ensure all possible categories are recognized.
  + Fit the LabelEncoder on the combined data and transform both datasets.

**Step 5: Train the Model**

features = ["HomePlanet", "CryoSleep", "Age", "Spa"]

X = train\_data[features]

y = train\_data["Transported"]

X\_train, X\_valid, y\_train, y\_valid = train\_test\_split(X, y, test\_size=0.2)

model = RandomForestClassifier(n\_estimators=100)

model.fit(X\_train, y\_train)

predictions = model.predict(X\_valid)

accuracy = (predictions == y\_valid).mean()

* + Split data into training/validation sets and train a Random Forest model.
  + Validation helps check if the model generalizes to unseen data.
  + Random Forest combines multiple decision trees to reduce overfitting.
  + Use train\_test\_split to reserve 20% of data for validation.
  + Train the model on X\_train/y\_train and test on X\_valid/y\_valid.

**Step 6: Generate Submission File**

test\_pred = model.predict(test\_data[features])

submission = pd.DataFrame({

"PassengerId": test\_data["PassengerId"],

"Transported": test\_pred

})

submission.to\_csv("submission.csv", index=False)

* + Create a CSV file with predictions for the test set.
  + Kaggle requires predictions in a specific format for scoring.
  + Use the trained model to predict on test\_data.
  + Create a DataFrame with the required columns (PassengerId, Transported).
  + Save to CSV without adding an index column.

**3. Results**

**Accuracy**: 73%

* + Handling missing values and encoding categories are critical for model performance.
  + Simple models like Random Forest can achieve decent accuracy with minimal tuning.

### Prediction Score on Kaggle:

