Perfect 👍 Let's go **cell by cell** and I'll explain each in **very simple and easy language** so you can clearly understand what's happening in your *ML Practical 1 – Data Preparation* notebook.

▯ CELL 1 (Markdown)

Content:
```
ML Practical 1. Data preparation
```

Explanation:
This is a **heading** written in Markdown (not code).
It simply gives the **title** of the practical — *Machine Learning Practical 1: Data Preparation.*

▯ CELL 2

```
#Import libraries
import pandas as pd
import numpy as np
```

Explanation:

1. `#Import libraries` — This is a **comment**; it's ignored by Python.
   It just tells what the next lines do.
2. `import pandas as pd` — This loads the **pandas** library and gives it the short name **pd** (used for data handling and analysis).
3. `import numpy as np` — This loads the **NumPy** library and gives it the short name **np** (used for numerical calculations).

✅ *Purpose:* These two libraries are commonly used for handling data in ML.

▯ CELL 3

```
#Load dataset
df = pd.read_csv("Heart.csv")
```

Explanation:

1. Comment — says that we are loading the dataset.
2. `pd.read_csv("Heart.csv")` — Reads the **Heart.csv** file into a pandas DataFrame.
3. The result is stored in variable **df**, which acts like a table in memory.

✅ *Purpose:* Load the heart dataset into the program for further analysis.

▯ CELL 4

```
#Display first few records
print("First 5 Rows of the Dataset:\n")
print(df.head(), "\n")
```

Explanation:

1. Comment — we are going to see the first few rows.
2. `print("First 5 Rows of the Dataset:\n")` — Prints a heading message.
3. `df.head()` — Shows the **first 5 rows** of the dataset.
   This helps to understand what kind of data and columns are present.

✅ *Purpose:* Quick look at the data to understand its structure.

▯ CELL 5 (Markdown)

```
a) Find Shape of Data
```

Explanation:
Just a text note telling the next step is to find the shape (number of rows and columns).

▯ CELL 6

```
#Display dataset shape
print("Shape of Dataset (Rows, Columns):", df.shape)
```

Explanation:

1. Comment — tells we're finding the shape of the dataset.
2. `df.shape` — Returns a tuple `(rows, columns)` showing how many rows and columns the dataset has.
3. The result is printed.

✅ *Purpose:* To know how large the dataset is.

▯ CELL 7 (Markdown)

```
b) Find Missing Values
```

Explanation:
A note saying next step is to check if any values are missing (NaN) in the dataset.

▯ CELL 8

```
#Check for null values in the dataset
print("\nMissing Values in Each Column:\n")
print(df.isnull().sum())
```

Explanation:

1. Comment — we're checking for missing data.
2. `df.isnull()` — Creates a table of **True/False** values (True if data is missing).
3. `.sum()` — Counts how many True (missing) values are there per column.
4. `print()` — Displays those counts.

✅ *Purpose:* To find columns that have missing (empty) values.

▯ CELL 9 (Markdown)

```
c) Find data type of each column
```

Explanation:
Tells that the next cell will check what kind of data (int, float, string, etc.) each column has.

▯ CELL 10

```
#Display data types of each column
print("\nData Types of Each Column:\n")
print(df.dtypes)
```

Explanation:

1. Comment — describes the action.
2. `df.dtypes` — Shows the **data type** (integer, float, object/string) for every column.
3. `print()` — Displays it nicely.

✅ *Purpose:* To know which columns have numeric values and which have text values.

▯ CELL 11 (Markdown)

```
d) Finding out Zero's
```

Explanation:
Indicates that we will now count how many **zero (0)** values are in each column.

▯ CELL 12

```
#Check how many zero values exist in each column
zero_counts = (df == 0).sum()

print("\nCount of Zero Values in Each Column:\n")
print(zero_counts)
```

Explanation:

1. `(df == 0)` — Creates a table of True/False values for every cell, checking if the value is zero.
2. `.sum()` — Counts how many True (i.e., zeros) in each column.
3. Stored in variable **zero_counts**.
4. `print()` — Shows the count for every column.

✅ *Purpose:* To identify columns with zero values (sometimes zero can mean missing or invalid).

▯ CELL 13 (Markdown)

```
e) Find Mean age of patients
```

Explanation:
Tells that the next step is to calculate the average age.

▯ CELL 14

```
#display mean age of patients
mean_age = df["Age"].mean()
print(f"\nMean Age of Patients: {mean_age:.2f} years")
```

Explanation:

1. `df["Age"]` — Selects the Age column.
2. `.mean()` — Calculates the average value of Age.
3. Stores it in **mean_age**.
4. `print(f"...")` — Displays the average age with 2 decimal places.

✅ *Purpose:* To find the average age of patients in the dataset.

▯ CELL 15 (Markdown)

```
f) Now extract only Age, Sex, ChestPain, RestBP, Chol. Randomly divide
dataset in training (75%) and testing (25%)
```

Explanation:
A note describing the next steps — selecting specific columns and splitting into training and test data.

▯ CELL 16

```
#Select only the given columns
subset = df[["Age", "Sex", "ChestPain", "RestBP", "Chol"]]
print("\nSelected Columns:\n")
print(subset.head())
```

Explanation:

1. `df[["Age", "Sex", "ChestPain", "RestBP", "Chol"]]` — Selects only these 5 columns.
2. Stores in **subset**.
3. `subset.head()` — Shows first 5 rows.

✓ *Purpose:* To create a smaller dataset with only selected columns.

▯ CELL 17

```
#Split into Training (75%) and Testing (25%)
from sklearn.model_selection import train_test_split
```

Explanation:

1. Comment — describes the purpose.
2. Imports the function **train_test_split** from **sklearn.model_selection**.
   It helps to split data into training and testing parts randomly.

✓ *Purpose:* Prepare for splitting the dataset.

▯ CELL 18

```
train, test = train_test_split(subset, test_size=0.25, random_state=0)
```

Explanation:

- `train_test_split()` divides the `subset` DataFrame into:
    - **75% training data** (used to train models)
    - **25% testing data** (used to test model performance)
- `random_state=0` ensures the same split each time (for reproducibility).

✅ *Purpose:* Create separate training and testing datasets.

▯ CELL 19

```
#Display shapes
print("\nTraining Set Shape :", train.shape)
print("Testing Set Shape  :", test.shape)
```

Explanation:

1. `train.shape` — Shows how many rows and columns in training data.
2. `test.shape` — Shows the same for testing data.
3. `print()` displays them.

✅ *Purpose:* Verify that the split worked correctly (usually 75%-25%).

▯ CELL 20 (Markdown)

Description:
A small story about a COVID prediction test, then asks to:

- Create confusion matrix
- Find Accuracy, Precision, Recall, and F1 Score.

Explanation:
It gives a real-world example to calculate performance metrics.

▯ CELL 21

```
# Import libraries
import matplotlib.pyplot as plt
import seaborn as sns

# Given data
TP = 45   # True Positives
FP = 55   # False Positives
FN = 5    # False Negatives
TN = 395  # True Negatives
```

```
total = 500

#Create confusion matrix
cm = [[TP, FP],
      [FN, TN]]

#Calculate performance metrics
accuracy = (TP + TN) / total
precision = TP / (TP + FP)
recall = TP / (TP + FN)
f1 = 2 * (precision * recall) / (precision + recall)

#Print results
print("Confusion Matrix:")
print(f"TP={TP}, FP={FP}, FN={FN}, TN={TN}\n")
print(f"Accuracy : {accuracy:.2f}")
print(f"Precision: {precision:.2f}")
print(f"Recall   : {recall:.2f}")
print(f"F1 Score : {f1:.2f}")

#Visualize confusion matrix
labels = ['Actual Positive', 'Actual Negative']
sns.heatmap(cm, annot=True, fmt='d', cmap='coolwarm',
            xticklabels=labels,
            yticklabels=['Predicted Positive', 'Predicted Negative'])
plt.title('Confusion Matrix for COVID Prediction')
plt.xlabel('Actual')
plt.ylabel('Predicted')
plt.show()
```

Explanation (step-by-step):

1. Import plotting libraries: **matplotlib** (plt) and **seaborn** (sns) for graphs.
2. Assign the test results — TP, FP, FN, TN values.
3. Store them in a **2×2 confusion matrix (cm)**.
4. Calculate:
   - **Accuracy** = (TP + TN) / total
   - **Precision** = TP / (TP + FP)
   - **Recall** = TP / (TP + FN)
   - **F1 Score** = 2 × (Precision × Recall) / (Precision + Recall)
5. Print the values.
6. Draw a **heatmap** using seaborn to visualize the confusion matrix with colors.

✓ *Purpose:* To evaluate model performance using all main metrics.

Would you like me to create a **PDF or Word file** containing this full “cell-by-cell easy explanation” so you can keep it for revision or submission?