To read a CSV file using **pandas** in Python, follow these steps:

**1. Import pandas**

python

CopyEdit

import pandas as pd

**2. Read the CSV file**

python

CopyEdit

df = pd.read\_csv('your\_file.csv')

Replace 'your\_file.csv' with the actual file path or file name.

**3. View the data**

python

CopyEdit

print(df.head()) # prints the first 5 rows

**Optional Parameters:**

* sep=',': Specify delimiter (e.g., sep=';' for semicolon-separated files).
* encoding='utf-8': Set encoding (e.g., for special characters).
* index\_col=0: Use a column as the index.
* usecols=['col1', 'col2']: Load only specific columns.

Example with options:

python

CopyEdit

df = pd.read\_csv('data.csv', sep=',', encoding='utf-8', usecols=['Name', 'Age'])

Got it! Let's go step by step: **read**, **filter**, **clean**, and **analyze** a CSV using pandas.

**🔹 Step 1: Read the CSV File**

python

CopyEdit

import pandas as pd

df = pd.read\_csv('data.csv') # Replace with your actual file path

print(df.head()) # View first 5 rows

**🔹 Step 2: Filter Specific Columns**

Let’s say you only need the Name, Age, and City columns:

python

CopyEdit

df = df[['Name', 'Age', 'City']]

**🔹 Step 3: Clean the Data**

**✅ Remove rows with missing values:**

python

CopyEdit

df = df.dropna()

**✅ Fill missing values with a default:**

python

CopyEdit

df['Age'] = df['Age'].fillna(0)

**✅ Convert column types (e.g., Age to integer):**

python

CopyEdit

df['Age'] = df['Age'].astype(int)

**✅ Strip whitespace from text columns:**

python

CopyEdit

df['Name'] = df['Name'].str.strip()

**🔹 Step 4: Analyze the Data**

**✅ Basic statistics:**

python

CopyEdit

print(df.describe()) # Gives mean, min, max, std, etc.

**✅ Count unique values in a column:**

python

CopyEdit

print(df['City'].value\_counts())

**✅ Filter rows (e.g., Age > 30):**

python

CopyEdit

adults = df[df['Age'] > 30]

print(adults)

**✅ Group and summarize:**

python

CopyEdit

print(df.groupby('City')['Age'].mean()) # Average age per city

**Step 5: Visualize the Data**

**📊 1. Import plotting libraries**

python

CopyEdit

import matplotlib.pyplot as plt

import seaborn as sns

# Optional: Make plots prettier

sns.set(style="whitegrid")

**📈 2. Bar Chart — Count of people by City**

python

CopyEdit

city\_counts = df['City'].value\_counts()

plt.figure(figsize=(10, 5))

sns.barplot(x=city\_counts.index, y=city\_counts.values, palette='viridis')

plt.title('Number of People by City')

plt.xlabel('City')

plt.ylabel('Count')

plt.xticks(rotation=45)

plt.tight\_layout()

plt.show()

**📊 3. Histogram — Age Distribution**

python

CopyEdit

plt.figure(figsize=(8, 5))

sns.histplot(df['Age'], bins=10, kde=True, color='skyblue')

plt.title('Age Distribution')

plt.xlabel('Age')

plt.ylabel('Frequency')

plt.show()

**📉 4. Boxplot — Age by City (to compare age ranges)**

python

CopyEdit

plt.figure(figsize=(10, 6))

sns.boxplot(x='City', y='Age', data=df, palette='Set2')

plt.title('Age Distribution by City')

plt.xticks(rotation=45)

plt.show()

**🧠 Bonus: Correlation Heatmap (if your CSV has numerical columns)**

python

CopyEdit

plt.figure(figsize=(8, 6))

sns.heatmap(df.corr(numeric\_only=True), annot=True, cmap='coolwarm')

plt.title('Correlation Matrix')

plt.show()

NOW LETS TRY WITH OUR CSV FILE WHICH IS TITANIC-DATASET.CSV

Here's a summary of our Titanic dataset:

**🔹 Overview:**

* **Rows**: 891
* **Columns**: 12
* **Missing Data**:
  + Age: 177 missing
  + Cabin: 687 missing (mostly empty)
  + Embarked: 2 missing

**🔹 Columns in the Dataset:**

* PassengerId: Unique ID for each passenger
* Survived: 0 = No, 1 = Yes
* Pclass: Passenger class (1 = 1st, 2 = 2nd, 3 = 3rd)
* Name, Sex, Age, SibSp (siblings/spouses aboard), Parch (parents/children aboard)
* Ticket, Fare, Cabin, Embarked (port: C = Cherbourg, Q = Queenstown, S = Southampton)

### **Cleaned Dataset Stats:**

* **Rows**: 889
* **Columns**: 11 (no nulls now)

our dataset is now ready for machine learning!

**✅ Preprocessing Summary:**

* **Categorical encoding**: Sex and Embarked converted to numeric.
* **Selected features**:  
  ['Pclass', 'Sex', 'Age', 'SibSp', 'Parch', 'Fare', 'Embarked']
* **Target variable**: Survived

**🔹 Train/Test Split:**

* **Training set**: 711 passengers
* **Test set**: 178 passengers

Lets run this code in your own Python environment to train and evaluate a logistic regression model:

python

CopyEdit

from sklearn.linear\_model import LogisticRegression

from sklearn.metrics import accuracy\_score, classification\_report, confusion\_matrix

# Train the model

model = LogisticRegression(max\_iter=1000)

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

# Predict on the test data

y\_pred = model.predict(X\_test)

# Evaluate the model

print("Accuracy:", accuracy\_score(y\_test, y\_pred))

print("\nClassification Report:\n", classification\_report(y\_test, y\_pred))

print("\nConfusion Matrix:\n", confusion\_matrix(y\_test, y\_pred))

This will give you the model's accuracy, a detailed report on precision/recall/F1-score, and a confusion matrix.

Let's switch things up and train a **Decision Tree** and a **Random Forest** model. These are great for handling nonlinear relationships and giving insights into **feature importance**.

**🌳 1. Decision Tree Classifier**

python

CopyEdit

from sklearn.tree import DecisionTreeClassifier

from sklearn.metrics import accuracy\_score, classification\_report, confusion\_matrix

# Train the Decision Tree

dt\_model = DecisionTreeClassifier(random\_state=42)

dt\_model.fit(X\_train, y\_train)

# Predict and evaluate

dt\_pred = dt\_model.predict(X\_test)

print("Decision Tree Accuracy:", accuracy\_score(y\_test, dt\_pred))

print("\nClassification Report:\n", classification\_report(y\_test, dt\_pred))

print("\nConfusion Matrix:\n", confusion\_matrix(y\_test, dt\_pred))

**🌲 2. Random Forest Classifier**

python

CopyEdit

from sklearn.ensemble import RandomForestClassifier

# Train the Random Forest

rf\_model = RandomForestClassifier(n\_estimators=100, random\_state=42)

rf\_model.fit(X\_train, y\_train)

# Predict and evaluate

rf\_pred = rf\_model.predict(X\_test)

print("Random Forest Accuracy:", accuracy\_score(y\_test, rf\_pred))

print("\nClassification Report:\n", classification\_report(y\_test, rf\_pred))

print("\nConfusion Matrix:\n", confusion\_matrix(y\_test, rf\_pred))

**💡 Bonus: Feature Importance (for Random Forest)**

python

CopyEdit

import matplotlib.pyplot as plt

# Plot feature importances

importances = rf\_model.feature\_importances\_

features = X\_train.columns

plt.figure(figsize=(8, 5))

plt.barh(features, importances, color='teal')

plt.title('Feature Importance (Random Forest)')

plt.xlabel('Importance Score')

plt.show()

you can **plot a Decision Tree** using sklearn.tree.plot\_tree.

**🖼️ Plotting the Decision Tree:**

python

CopyEdit

from sklearn.tree import plot\_tree

import matplotlib.pyplot as plt

# Plot the trained decision tree

plt.figure(figsize=(20, 10)) # Adjust size as needed

plot\_tree(dt\_model,

feature\_names=X\_train.columns,

class\_names=['Not Survived', 'Survived'],

filled=True,

rounded=True)

plt.title("Decision Tree Visualization")

plt.show()

* You can set max\_depth=3 when creating DecisionTreeClassifier to make a smaller, more readable tree.

python

CopyEdit

dt\_model = DecisionTreeClassifier(max\_depth=3, random\_state=42)