**1. Install Python & Set Up Your Project Folder**

1. **Install Python 3.10+** (if you haven’t):
   * Windows/macOS: download from <https://python.org/downloads>
2. **Create your project directory** and enter it:

mkdir data\_analysis\_app

cd data\_analysis\_app

1. **nitialize a Git repo** (optional but recommended):

git init

**2. Create and Activate a Virtual Environment**

1. **Create the venv**:

python3 -m venv venv

**Activate it**: Windows (PowerShell): venv\Scripts\Activate.ps1

1. **Confirm** you’re in the venv: which python # should point into ./venv/

**3. Create a requirements.txt with all the required dependencies:**

# Create the file and open it in Notepad:

notepad requirements.txt

then add requirements:

pandas

scikit-learn

matplotlib # (if you’ll plot)

tk

save and close .

**5.Install them**:

pip install -r requirements.txt

after installation you will get a message similar 

**6. Structure Your Code into Modules**

**data\_analysis\_app/**

**├── analyses/**

**│ ├── \_\_init\_\_.py**

**│ ├── descriptive\_stats.py**

**│ ├── linear\_regression.py**

**│ └── clustering.py**

**└── main.py**

6.1 analyses/descriptive\_stats.py

import pandas as pd

from sklearn.cluster import KMeans

def run(df: pd.DataFrame):

    """

    Prompts for number of clusters, runs KMeans on numeric cols,

    and prints cluster assignments.

    """

    nums = df.select\_dtypes(include='number')

    if nums.empty:

        print("No numeric columns to cluster.")

        return

    k = int(input("Number of clusters? ").strip() or 3)

    km = KMeans(n\_clusters=k, random\_state=42)

    labels = km.fit\_predict(nums)

    df\_with = df.copy()

    df\_with['cluster'] = labels

    print(df\_with[['cluster']].value\_counts())

6. 2. analyses/linear\_regression.py

import pandas as pd

from sklearn.linear\_model import LinearRegression

from sklearn.preprocessing import LabelEncoder

from pandas.api.types import is\_numeric\_dtype

def run(df: pd.DataFrame):

    """

    1) Show a preview of the data

    2) Ask user to pick a target column by index

    3) If the target is non-numeric, label-encode it

    4) Fit LinearRegression on numeric features & the (possibly encoded) target

    """

    # 1) preview

    print("=== DATA PREVIEW ===")

    print(df.head(), "\n")

    # 2) list columns

    print("=== COLUMNS ===")

    for idx, col in enumerate(df.columns):

        print(f"{idx}: {col}")

    print()

    # 3) choose target

    sel = input("Enter the index of the target column: ").strip()

    try:

        i = int(sel)

        target\_col = df.columns[i]

    except (ValueError, IndexError):

        print(f"Invalid selection '{sel}'. Defaulting to last column.")

        target\_col = df.columns[-1]

    print(f"\nUsing '{target\_col}' as target.\n")

    # extract y, encoding if needed

    y\_series = df[target\_col]

    if not is\_numeric\_dtype(y\_series):

        le = LabelEncoder()

        y = le.fit\_transform(y\_series)

        print(f"Label-encoded '{target\_col}' categories → {list(le.classes\_)}\n")

    else:

        y = y\_series.values

    # build X from all other numeric columns

    feature\_df = df.drop(columns=[target\_col]).select\_dtypes(include='number')

    if feature\_df.shape[1] == 0:

        raise ValueError("No numeric feature columns found after dropping the target.")

    X = feature\_df.values

    # 4) fit and report

    model = LinearRegression()

    model.fit(X, y)

    print("=== MODEL RESULTS ===")

    print("Features used:", list(feature\_df.columns))

    print("Coefficients:", model.coef\_)

    print("Intercept:", model.intercept\_)

    print("R² score:", model.score(X, y))

6. 3. Analyses/clustering.py

import pandas as pd

def run(df: pd.DataFrame):

    """

    Print basic descriptive statistics for numeric columns.

    """

    desc = df.describe()

    print(desc)

**7 Analyses/clustering.py**

import pandas as pd

from analyses.descriptive\_stats import run as run\_stats

from analyses.linear\_regression import run as run\_lr

from analyses.clustering import run as run\_clust

import sys

def print\_menu():

    print("""

Choose an analysis:

  1) Descriptive statistics

  2) Linear regression

  3) K-Means clustering

  0) Exit

""")

    return input("Enter choice: ").strip()

def main():

    path = input("Path to CSV file: ").strip()

    try:

        df = pd.read\_csv(path)

    except Exception as e:

        print(f"Failed to load CSV: {e}")

        sys.exit(1)

    while True:

        choice = print\_menu()

        if choice == "1":

            run\_stats(df)

        elif choice == "2":

            run\_lr(df)

        elif choice == "3":

            run\_clust(df)

        elif choice == "0":

            print("Goodbye!")

            break

        else:

            print("Invalid choice; please try again.")

if \_\_name\_\_ == "\_\_main\_\_":

    main()

**8.** **Test Your App in the Console**

python main.py

* Enter the path to a sample CSV.
* Try each menu option and confirm results.

**9.Add a GUI with Tkinter**

Create gui.py alongside main.py:

import tkinter as tk

from tkinter import filedialog, messagebox

import pandas as pd

from analyses.linear\_regression import run as run\_lr

from analyses.descriptive\_stats import run as run\_stats

from analyses.clustering import run as run\_clust

class App(tk.Tk):

    def \_\_init\_\_(self):

        super().\_\_init\_\_()

        self.title("Data Analysis App")

        self.df = None

        tk.Button(self, text="Load CSV", command=self.load\_csv).pack(pady=5)

        tk.Button(self, text="Descriptive Stats", command=self.do\_stats).pack(pady=5)

        tk.Button(self, text="Linear Regression", command=self.do\_lr).pack(pady=5)

        tk.Button(self, text="K-Means Clustering", command=self.do\_clust).pack(pady=5)

    def load\_csv(self):

        path = filedialog.askopenfilename(filetypes=[("CSV", "\*.csv")])

        if not path:

            return

        try:

            self.df = pd.read\_csv(path)

            messagebox.showinfo("Loaded", f"{len(self.df)} rows loaded.")

        except Exception as e:

            messagebox.showerror("Error", str(e))

    def ensure\_df(self):

        if self.df is None:

            messagebox.showwarning("No Data", "Please load a CSV first.")

            return False

        return True

    def do\_stats(self):

        if self.ensure\_df():

            run\_stats(self.df)

    def do\_lr(self):

        if self.ensure\_df():

            run\_lr(self.df)

    def do\_clust(self):

        if self.ensure\_df():

            run\_clust(self.df)

if \_\_name\_\_ == "\_\_main\_\_":

    App().mainloop()

**10.Run** it:

python gui.py

**11.Package into a Standalone Executable**

**11.1.Install PyInstaller:** pip install pyinstaller

**11.2. Generate one-file build:** python -m PyInstaller --onefile main.py

**11.3. (For GUI) instead package gui.py:** python -m PyInstaller --onefile gui.py

**11.4. Distribute :**the single binary in dist/ — no Python install required on the user’s machine.