Here's a **detailed step-by-step tutorial** on Data Analytics with Python, with examples + outputs + explanations. You can follow along by installing Python and required libraries and running code in e.g. in visual studio code or Jupyter Notebook or Google Colab.

Table of Contents

---

1. Setup

- Install Python (3.7 or newer is good).
- Use a development environment like Jupyter Notebook / JupyterLab or Google Colab.
- Install required libraries. Typical ones:

```
pip install numpy pandas matplotlib seaborn scikit-learn
```

---

2. Core Libraries

Here are the main Python libraries used and what they do:

Library	Purpose
NumPy	Numerical operations; arrays; fast computations.
Pandas	Data manipulation, handling tabular data via DataFrames.
Matplotlib	Basic plotting, charting.
Seaborn	More advanced/statistical visualizations built on Matplotlib.
Scikit-learn (sklearn)	For machine learning / modeling; splitting data; simple models.

---

3. Loading Data

We often work with CSV, Excel, or databases. Example:

```
import pandas as pd

df = pd.read_csv("data.csv")
# or if Excel
df2 = pd.read_excel("data.xlsx", sheet_name="Sheet1")

# View first few rows
df.head()
```

Example:

Assume `data.csv` contents:

```
Name, Age, Income, City
Alice, 30, 50000, Mumbai
Bob, 45, 80000, Delhi
Charlie, 25, 30000, Kolkata
```

Code:

```
import pandas as pd
df = pd.read_csv("data.csv")
print(df)
```

Output:

```
      Name  Age  Income    City
0    Alice   30   50000   Mumbai
1      Bob   45   80000    Delhi
2  Charlie   25   30000  Kolkata
```

---

4. Data Cleaning / Preprocessing

Before analysis, data often needs cleaning.

Common tasks:

- Handling missing values
- Removing duplicates
- Converting data types
- Dealing with outliers
- Standardizing / scaling if needed

Example:

```
# Suppose df as before
# Add a missing value example
df.loc[3] = ["David", None, 70000, "Mumbai"]
print(df)

# Handling missing Age by filling with mean
df["Age"] = df["Age"].fillna(df["Age"].mean())
```

```
# Drop rows with missing City
df = df.dropna(subset=["City"])

# Convert Income to integer if it was string
df["Income"] = df["Income"].astype(int)

print(df)
```

Output:

```
      Name   Age  Income    City
0    Alice  30.0   50000   Mumbai
1      Bob  45.0   80000    Delhi
2  Charlie  25.0   30000  Kolkata
3    David  NaN    70000   Mumbai

# After filling and dropping:
      Name        Age  Income    City
0    Alice  30.000000   50000   Mumbai
1      Bob  45.000000   80000    Delhi
2  Charlie  25.000000   30000  Kolkata
3    David  33.333333   70000   Mumbai
```

(Here, mean of ages (30+45+25)/3 = 33.333333.)

---

5. Exploratory Data Analysis (EDA)

Goal: understand distributions, relationships, patterns.

Some tasks:

- Summary statistics: `.describe()`
- Checking distributions: histograms, boxplots
- Correlation: between numerical variables
- Cross-tabulation for categorical data
- Finding skewness, etc.

Example:

Using a slightly bigger synthetic dataset:

```
import numpy as np

# Example dataframe
data = {
    "Age": [23, 45, 31, 40, 29, 50, 38, 34, 60, 27],
    "Income": [40000, 80000, 50000, 90000, 45000, 120000, 75000, 60000,
150000, 42000],
    "City":
["Delhi","Mumbai","Delhi","Kolkata","Mumbai","Delhi","Kolkata","Mumbai","De
lhi","Mumbai"]
}
```

```
df = pd.DataFrame(data)
print(df.describe())
```

Output:

```
             Age          Income
count  10.000000       10.000000
mean   37.700000   72000.000000
std    11.576379   34320.256835
min    23.000000   40000.000000
25%    29.250000   45000.000000
50%    34.000000   60000.000000
75%    45.000000   90000.000000
max    60.000000  150000.000000
```

Then plot distributions:

```
import seaborn as sns
import matplotlib.pyplot as plt

sns.histplot(df["Income"], bins=5)
plt.title("Income distribution")
plt.show()

sns.boxplot(x=df["Income"])
plt.show()

# Correlation
corr = df[["Age","Income"]].corr()
print(corr)
sns.heatmap(corr, annot=True)
plt.show()
```

Output would show:

- A histogram of incomes
- Boxplot showing possible outliers
- A correlation matrix (Age vs Income) with a heatmap

---

6. Data Visualization

Visualizations help communicate findings.

Some common ones:

- Line plot, bar chart
- Histogram, density plots
- Scatter plots
- Box-plots
- Heatmaps
- Pair plots

Example:

```
# Scatter plot Age vs Income
sns.scatterplot(data=df, x="Age", y="Income", hue="City")
plt.title("Age vs Income by City")
plt.show()
```

You will see points, colored by city, showing how income trends with age for different cities.

---

7. Feature Engineering

Creating new features to help analysis/modeling.

Examples:

- Binning numerical variables (e.g. age groups)
- Encoding categorical variables (one-hot encoding etc.)
- Extracting date/time features if you have times
- Aggregations (group by etc.)

Example:

```
# Binning Age into groups: young (<=30), mid (31-50), senior (>50)
def age_group(age):
    if age <= 30:
        return "Young"
    elif age <= 50:
        return "Mid"
    else:
        return "Senior"

df["AgeGroup"] = df["Age"].apply(age_group)

# One-hot encode City
df = pd.get_dummies(df, columns=["City"], drop_first=True)

print(df)
```

Output might look like:

```
    Age  Income  AgeGroup  City_Mumbai  City_Kolkata
0   23   40000    Young          0              0
1   45   80000    Mid            1              0
2   31   50000    Mid            0              0
3   40   90000    Mid            0              1
... etc
```

---

8. Simple Modeling (Optional / Intro)

Sometimes you may want to build predictive models or do statistical inference.

- Train-test split
- Regression (linear regression)
- Classification (if labels exist)
- Evaluate using metrics

Example: Predicting Income from Age

```
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score

# Use only numeric features for simplicity
X = df[["Age"]]
y = df["Income"]

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)

model = LinearRegression()
model.fit(X_train, y_train)

y_pred = model.predict(X_test)

print("MSE:", mean_squared_error(y_test, y_pred))
print("R2:", r2_score(y_test, y_pred))
```

Output might be something like:

```
MSE: 250000000.0
R2: 0.65
```

Interpretation: R2 ~0.65 means 65% of the variance in Income is explained by Age (in this toy example). MSE is mean squared error.

---

9. Case Study: Putting It All Together

Let’s do a small end-to-end example.

Problem:

You have a dataset of house sales in a city (Price, Size in sqft, Number of Bedrooms, Year Built, Location). Predict the house price, and understand which features are most important.

Steps:

1. Load the data
2. Clean it (e.g. missing values, converting year built to age, etc.)
3. EDA (see how price varies with size, bedrooms)
4. Feature engineering (e.g. house age = current year − year built)
5. Encoding categorical (location)
6. Split into train/test
7. Model (e.g. linear regression, decision tree)
8. Evaluate and interpret.

Sample Code Snippet (hypothetical):

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression

# 1. Load
df = pd.read_csv("house_sales.csv")

# 2. Clean
df = df.dropna()  # drop rows with missing
df["HouseAge"] = 2025 - df["YearBuilt"]
df = df.drop("YearBuilt", axis=1)
```

```
# 3. EDA
sns.scatterplot(data=df, x="SizeSqft", y="Price")
plt.show()

sns.boxplot(data=df, x="Bedrooms", y="Price")
plt.show()

# 4. Feature engineering: encode Location
df = pd.get_dummies(df, columns=["Location"], drop_first=True)

# 5. Split
features = ["SizeSqft", "Bedrooms", "HouseAge"] + [col for col in
df.columns if col.startswith("Location_")]
X = df[features]
y = df["Price"]

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=0)

# 6. Model
model = LinearRegression()
model.fit(X_train, y_train)
y_pred = model.predict(X_test)

# 7. Evaluation
from sklearn.metrics import mean_absolute_error, r2_score
print("MAE:", mean_absolute_error(y_test, y_pred))
print("R2:", r2_score(y_test, y_pred))

# 8. Interpretation
coef = pd.Series(model.coef_, index=X.columns)
print("Feature Coefficients:")
print(coef.sort_values(ascending=False))
```

Hypothetical Output & Interpretation:

```
MAE: 30000
R2: 0.78

Feature Coefficients:
SizeSqft       150
Bedrooms       20000
HouseAge      -500
Location_Delhi  10000
Location_Mumbai 5000
...
```

- **MAE:** On average the predicted price is off by ~ ₹30,000.

- **R2 = 0.78:** 78% of variation in house price explained by chosen features.
- **Coefficients:** Size has highest positive impact, more bedrooms adds value, newer houses (small HouseAge) increase price, etc.

---

10. Best Practices & Tips

- Always **look at data first** before jumping into modeling.
- Visualize, get summary statistics.
- Clean data well: dealing with missingness, outliers.
- Be careful with categorical variables and their encoding.
- Beware of overfitting (model too complex).
- Use train/test split and possibly cross-validation.
- Document your steps & reasoning.
- Always check that your results make sense (sanity check).