|  |  |
| --- | --- |
| **1. Python Basics**  1.1 Variables and Data Types  x = 10 # Integer  y = 3.14 # Float  name = "Alice" # String  is\_valid = True # Boolean  Key Points:   * Variables store data values. * Common data types: int, float, str, bool   1.2 Operators  Arithmetic Operators:  sum = 5 + 3 # Addition  sub = 10 - 4 # Subtraction  mul = 2 \* 6 # Multiplication  div = 10 / 2 # Division  mod = 10 % 3 # Modulus  Comparison Operators:  x == y # Equal  y > x # Greater than  x <= y # Less than or equal  **2. Data Structures**  2.1 Lists  Lists are ordered collections of elements.  fruits = ["apple", "banana", "cherry"]  print(fruits[0]) # Output: apple  fruits.append("orange") # Add element  2.2 Tuples  Tuples are immutable ordered collections.  days = ("Monday", "Tuesday", "Wednesday")  print(days[1]) # Output: Tuesday  2.3 Dictionaries  Dictionaries store key-value pairs.  student = {"name": "John", "age": 20, "grade": "A"}  print(student["name"]) # Output: John  2.4 Sets  Sets store unique elements.  numbers = {1, 2, 3, 4, 5}  numbers.add(6)  print(numbers) | **3. Control Structures**  3.1 If-Else Statements  age = 18  if age >= 18:  print("Adult")  else:  print("Minor")  3.2 Loops  For Loop:  for i in range(5):  print(i)  While Loop:  x = 0  while x < 5:  print(x)  x += 1  **4. Functions**  Functions allow code reuse.  def greet(name):  return "Hello, " + name  print(greet("Alice"))  **5. File Handling**  Reading and writing files in Python.  with open("data.txt", "w") as file:  file.write("Hello, world!") |

**Exploratory Data Analysis (EDA)**

**Introduction**

Exploratory Data Analysis (EDA) is an essential step in data analysis that helps understand the structure and patterns within a dataset. This process involves summarizing main characteristics, handling missing values, detecting outliers, and visualizing data. This document provides a step-by-step guide on performing EDA using the Pandas library in Python.

**1. Importing Necessary Libraries**

Before starting with EDA, we need to import the required libraries. These libraries help in data manipulation, numerical calculations, and visualization.

import pandas as pd # For data handling and manipulation

import numpy as np # For numerical computations

import seaborn as sns # For data visualization

import matplotlib.pyplot as plt # For plotting graphs

**2. Loading the Dataset**

Pandas provides functions to load datasets from various formats, such as CSV, Excel, and databases. The dataset should be loaded into a DataFrame to enable easy analysis.

df = pd.read\_csv("data.csv") # Replace with your dataset file

**3. Understanding the Dataset**

Before conducting any analysis, it's important to understand the dataset structure.

**3.1 Display First Few Rows**

The .head() function helps preview the first five rows of the dataset.

print(df.head())

**3.2 Dataset Shape**

The .shape attribute shows the number of rows and columns in the dataset.

print(df.shape) # (rows, columns)

**3.3 Data Types**

This step helps in identifying whether the dataset has numerical or categorical values.

print(df.dtypes)

**3.4 Checking for Missing Values**

Missing values can affect the accuracy of models, so they need to be handled properly.

print(df.isnull().sum())

**4. Handling Missing Data**

There are different ways to handle missing data, depending on the type of variable.

**4.1 Filling Missing Values**

For numerical columns, missing values can be replaced with the mean, while categorical columns are filled with the most frequent value (mode).

df.fillna(df.mean(), inplace=True) # Filling numerical missing values with mean

df.fillna(df.mode().iloc[0], inplace=True) # Filling categorical missing values with mode

**4.2 Dropping Missing Values**

Alternatively, if the missing values are too many, the rows can be removed entirely.

df.dropna(inplace=True) # Removes rows with missing values

**5. Handling Outliers**

Outliers can significantly impact the analysis. The Interquartile Range (IQR) method is commonly used to detect and remove them.

Q1 = df.quantile(0.25)

Q3 = df.quantile(0.75)

IQR = Q3 - Q1

# Removing outliers that fall outside 1.5 \* IQR range

df\_cleaned = df[~((df < (Q1 - 1.5 \* IQR)) | (df > (Q3 + 1.5 \* IQR))).any(axis=1)]

**6. Encoding Categorical Data**

Categorical variables must be converted into numerical format before they can be used in machine learning models. Label Encoding is a simple method where unique category values are assigned numerical labels.

from sklearn.preprocessing import LabelEncoder

encoder = LabelEncoder()

df["Category"] = encoder.fit\_transform(df["Category"])

**7. Feature Scaling**

Feature scaling is essential to ensure that all numerical variables have the same weight in machine learning models. Min-Max scaling scales values between 0 and 1.

from sklearn.preprocessing import MinMaxScaler

scaler = MinMaxScaler()

df["Scaled Feature"] = scaler.fit\_transform(df[["Numerical Feature"]])

**8. Data Visualization**

Visualization helps in understanding the data distribution and relationships between variables.

**8.1 Histogram**

A histogram shows the distribution of numerical data.

sns.histplot(df['Age'], bins=30, kde=True)

plt.show()

**8.2 Boxplot for Outliers**

A boxplot visually represents outliers and distribution spread.

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

plt.show()

**8.3 Correlation Heatmap**

A heatmap shows the correlation between numerical variables. Higher correlation values indicate stronger relationships.

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

plt.show()

**FEATURE ENGINEERING**

**Introduction**

Feature engineering is the process of transforming raw data into meaningful features that improve the performance of machine learning models. This guide covers essential feature engineering techniques, including handling missing data, encoding categorical variables, scaling features, and extracting new features.

**1. Importing Necessary Libraries**

import pandas as pd

import numpy as np

from sklearn.preprocessing import LabelEncoder, OneHotEncoder, MinMaxScaler, StandardScaler

**2. Loading the Dataset**

df = pd.read\_csv("data.csv") # Replace with your dataset file

**3. Handling Missing Data**

**3.1 Filling Missing Values**

df.fillna(df.mean(), inplace=True) # Filling numerical missing values with mean

df.fillna(df.mode().iloc[0], inplace=True) # Filling categorical missing values with mode

**3.2 Dropping Missing Values**

df.dropna(inplace=True) # Removes rows with missing values

**4. Encoding Categorical Variables**

**4.1 Label Encoding**

encoder = LabelEncoder()

df["Category"] = encoder.fit\_transform(df["Category"])

**4.2 One-Hot Encoding**

df = pd.get\_dummies(df, columns=["Category"], drop\_first=True)

**5. Feature Scaling**

**5.1 Min-Max Scaling**

scaler = MinMaxScaler()

df[["Numerical Feature"]] = scaler.fit\_transform(df[["Numerical Feature"]])

**5.2 Standardization**

scaler = StandardScaler()

df[["Numerical Feature"]] = scaler.fit\_transform(df[["Numerical Feature"]])

**6. Feature Extraction**

**6.1 Creating New Features**

df["Age\_Group"] = pd.cut(df["Age"], bins=[0, 18, 35, 60, np.inf], labels=["Teen", "Young Adult", "Adult", "Senior"])

**6.2 Date-Based Feature Extraction**

df["Year"] = pd.to\_datetime(df["Date"]).dt.year

df["Month"] = pd.to\_datetime(df["Date"]).dt.month

df["Day"] = pd.to\_datetime(df["Date"]).dt.day