Introduction to Python

What is Python and Why Python for Data Science?
Python is a versatile, high-level programming language known for its simplicity and readability. It offers a rich ecosystem of libraries and tools tailored for various tasks, including data science. Python's syntax is straightforward and easy to understand, making it an ideal choice for beginners and experts alike. For data science, Python provides libraries like NumPy, Pandas, Matplotlib, and scikit-learn, which are widely used for data manipulation, visualization, and machine learning tasks.

Python Data Types

Python supports various data types, each serving a different purpose:
Integers: Whole numbers without decimal points.
Floats: Numbers with decimal points.
Strings: Sequences of characters enclosed in quotes.
Lists: Ordered collections of items, mutable (modifiable).
Tuples: Ordered collections of items, immutable (cannot be modified).
Dictionaries: Key-value pairs, unordered and mutable.
Sets: Unordered collections of unique items.

Functions in Python

Defining Functions
Functions are blocks of reusable code that perform a specific task. They help in organizing code and making it more modular.
Parameters and Arguments
Parameters are variables defined in the function header, while arguments are the actual values passed to the function when it's called.
Return Statement
The return statement is used to exit a function and return a value or multiple values back to the caller.

Python Control Structures

Conditional Statements (if, elif, else)
Conditional statements allow the execution of different code blocks based on certain conditions.
if: Executes a block of code if a condition is true.
elif: Used for additional conditions if the initial condition is false.
else: Executes a block of code if none of the preceding conditions are true.
Loops (for loops, while loops)
Loops are used to repeatedly execute a block of code.
for loops: Iterate over a sequence (e.g., list, tuple) or other iterable objects.
while loops: Execute a block of code as long as a condition is true.

Python Libraries for Data Science

NumPy: NumPy is a fundamental package for scientific computing with Python. It provides support for multidimensional arrays and matrices, along with a collection of mathematical functions to operate on these arrays.
Pandas: Pandas is a powerful library for data manipulation and analysis. It offers data structures like DataFrame and Series, which make working with structured data easy and intuitive.
Matplotlib: Matplotlib is a plotting library for creating static, interactive, and animated visualizations in Python. It provides a wide range of plotting functions to create various types of plots.
Seaborn: Seaborn is built on top of Matplotlib and provides a high-level interface for drawing attractive and informative statistical graphics. It simplifies the process of creating complex visualizations.

Basic Machine Learning Concepts with Python

Introduction to Scikit-Learn
Scikit-Learn is a machine learning library in Python that provides simple and efficient tools for data mining and data analysis. It includes various algorithms for classification, regression, clustering, dimensionality reduction, and model selection.
Train-Test Split
Train-test split is a technique used to evaluate the performance of machine learning models. It involves splitting the dataset into two subsets: one for training the model and the other for testing its performance.

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

Model Training and Evaluation
Model training involves fitting a machine learning algorithm to the training data, while evaluation assesses its performance on unseen data. Common metrics for evaluation include accuracy, precision, recall, F1-score, and ROC-AUC score.

Data Import and Export

Reading Data from CSV, Excel, JSON, etc.

```
import pandas as pd
# Read CSV data
data = pd.read_csv("data.csv")
# Read Excel data (assuming 'Sheet1' is the sheet name)
data = pd.read_excel("data.xlsx", sheet_name="Sheet1")
# Read JSON data (assuming 'data' is the key holding the data)
import json
with open("data.json", "r") as f:
    data = json.load(f)
```

Writing Data to CSV, Excel, JSON, etc.

```
import pandas as pd
# Write DataFrame to CSV
data.to_csv("output.csv", index=False)
# Set index=False to avoid saving the index column
# Write DataFrame to Excel (assuming 'Sheet1' is the sheet name)
data.to_excel("output.xlsx", sheet_name="Sheet1")
# Write data to JSON (assuming 'data' is the key to store the
data)
import json
with open("output.json", "w") as f:
    json.dump(data.to_dict(orient="records"), f)
# Convert DataFrame to a list of dictionaries
```

Data Cleaning and Preprocessing

Handling Missing Values
Missing values are common in real-world datasets. Techniques like imputation (replacing missing values with a meaningful estimate) or removal can be used to handle missing data.

```
import pandas as pd
# Drop rows with missing values (be cautious, may lose data)
data.dropna(inplace=True)
# Fill missing values with a specific value (e.g., mean)
data["Age"].fillna(data["Age"].mean(), inplace=True)
```

Data Transformation
Data transformation involves converting raw data into a format suitable for analysis. This may include encoding categorical variables into numerical format or scaling data to ensure all features have a similar scale.

```
import LabelEncoder, StandardScaler
encoder = LabelEncoder()
df['category_encoded'] = encoder.fit_transform(df['category'])
scaler = StandardScaler()
df_scaled = scaler.fit_transform(df[['feature1', 'feature2']])
```

Basic Statistical Analysis with Python

Descriptive Statistics
Descriptive statistics summarize and describe the features of a dataset. Common measures include mean, median, mode, variance, and standard deviation.

```
import pandas as pd
data = pd.DataFrame({"Scores": [80, 95, 72, 90, 88]})
# Calculate mean, median, standard deviation
mean_score = data["Scores"].mean()
median_score = data["Scores"].median()
std_dev = data["Scores"].std()
print(f"Mean score: {mean_score}")
print(f"Median score: {median_score}")
print(f"Standard deviation: {std_dev}")
```

Correlation Analysis
Correlation analysis measures the strength and direction of the linear relationship between two quantitative variables. It helps in understanding how variables are related to each other.

```
import pandas as pd
data = pd.DataFrame({"Height": [170, 180, 165, 175, 182],
"Weight": [65, 72, 58, 70, 75]})
correlation = data["Height"].corr(data["Weight"])
print(f"Correlation between height and weight: {correlation}")
```