# Chapter 2: Python Fundamentals, Machine Learning, and AWS Integration

## Part 1: Python Fundamentals

### 1.1 Setting Up Python

1. Visit <https://www.python.org/downloads/>
2. Download and install the latest version of Python for your operating system
3. Verify installation by opening a terminal/command prompt and typing:

Copy

python --version

### 1.2 Python Basics

## Variables and Data Types

python

Copy

x = 5 *# integer*

y = 3.14 *# float*

name = "Alice" *# string*

is\_student = True *# boolean*

print(f"x: {x}, y: {y}, name: {name}, is\_student: {is\_student}")

## Control Structures

##### If Statements

python

Copy

age = 20

if age < 18:

print("Minor")

elif age >= 18 and age < 65:

print("Adult")

else:

print("Senior")

##### Loops

###### For Loops

python

Copy

*# Iterating over a list*

fruits = ["apple", "banana", "cherry"]

for fruit in fruits:

print(fruit)

*# Using range*

for i in range(5):

print(i)

*# Iterating over a dictionary*

person = {"name": "Bob", "age": 30, "city": "New York"}

for key, value in person.items():

print(f"{key}: {value}")

###### While Loops

python

Copy

count = 0

while count < 5:

print(count)

count += 1

## Functions

python

Copy

def greet(name):

return f"Hello, {name}!"

print(greet("World"))

*# Function with default parameter*

def power(base, exponent=2):

return base \*\* exponent

print(power(3)) *# 9*

print(power(3, 3)) *# 27*

### 1.3 Data Structures

## Lists

python

Copy

fruits = ["apple", "banana", "cherry"]

print(fruits[0]) *# apple*

fruits.append("date")

print(fruits) *# ['apple', 'banana', 'cherry', 'date']*

fruits.remove("banana")

print(fruits) *# ['apple', 'cherry', 'date']*

## Tuples

python

Copy

coordinates = (3, 4)

x, y = coordinates

print(f"x: {x}, y: {y}")

## Dictionaries

python

Copy

person = {"name": "Alice", "age": 30, "city": "New York"}

print(person["name"]) *# Alice*

person["job"] = "Engineer"

print(person)

## Sets

python

Copy

fruits = {"apple", "banana", "cherry"}

fruits.add("date")

print(fruits)

fruits.remove("banana")

print(fruits)

## Part 2: Python for Data Science and Machine Learning

### 2.1 Introduction to NumPy

NumPy is a fundamental package for scientific computing in Python. It provides support for large, multi-dimensional arrays and matrices, along with a collection of mathematical functions to operate on these arrays efficiently.

First, install NumPy:

Copy

pip install numpy

Basic NumPy operations:

python

Copy

import numpy as np

*# Create arrays*

arr1 = np.array([1, 2, 3, 4, 5])

arr2 = np.array([6, 7, 8, 9, 10])

print("Array 1:", arr1)

print("Array 2:", arr2)

*# Basic operations*

print("Sum:", arr1 + arr2)

print("Multiplication:", arr1 \* arr2)

*# Statistical operations*

print("Mean of arr1:", np.mean(arr1))

print("Standard deviation of arr2:", np.std(arr2))

*# Reshaping*

matrix = arr1.reshape(5, 1)

print("Reshaped array:\n", matrix)

*# Linear algebra*

A = np.array([[1, 2], [3, 4]])

B = np.array([[5, 6], [7, 8]])

print("Matrix multiplication:\n", np.dot(A, B))

### 2.2 Introduction to Pandas

Pandas is a powerful data manipulation and analysis library for Python. It provides data structures like Series (1-dimensional) and DataFrame (2-dimensional) that allow you to work with structured data efficiently.

Install Pandas:

Copy

pip install pandas

Let's explore Pandas with more detailed examples:

python

Copy

import pandas as pd

import numpy as np

*# Creating a DataFrame*

df = pd.DataFrame({

'Name': ['John', 'Anna', 'Peter', 'Linda'],

'Age': [28, 34, 29, 32],

'City': ['New York', 'Paris', 'Berlin', 'London'],

'Salary': [50000, 60000, 55000, 75000]

})

print("Original DataFrame:")

print(df)

*# Basic information about the DataFrame*

print("\nDataFrame Info:")

df.info()

*# Statistical summary*

print("\nStatistical Summary:")

print(df.describe())

*# Selecting columns*

print("\nNames and Ages:")

print(df[['Name', 'Age']])

*# Filtering rows*

print("\nPeople older than 30:")

print(df[df['Age'] > 30])

*# Adding a new column*

df['Experience'] = [3, 8, 4, 10]

print("\nDataFrame with new 'Experience' column:")

print(df)

*# Group by and aggregate*

print("\nAverage Salary by City:")

print(df.groupby('City')['Salary'].mean())

*# Sorting*

print("\nSorted by Age (descending):")

print(df.sort\_values('Age', ascending=False))

*# Handling missing values*

df.loc[1, 'Salary'] = np.nan

print("\nDataFrame with missing value:")

print(df)

print("\nDropping rows with missing values:")

print(df.dropna())

print("\nFilling missing values with mean:")

print(df.fillna(df.mean()))

*# Date handling*

df['Date'] = pd.date\_range('2023-01-01', periods=4)

print("\nDataFrame with dates:")

print(df)

print("\nExtracting year from date:")

print(df['Date'].dt.year)

*# Reading and writing data*

df.to\_csv('employees.csv', index=False)

df\_read = pd.read\_csv('employees.csv')

print("\nData read from CSV:")

print(df\_read)

This example demonstrates various Pandas operations including creating DataFrames, selecting and filtering data, adding columns, grouping and aggregating, sorting, handling missing values, and working with dates.

### 2.3 Introduction to Scikit-learn

Scikit-learn is a machine learning library for Python. It features various classification, regression and clustering algorithms including support vector machines, random forests, gradient boosting, k-means and DBSCAN, and is designed to interoperate with the Python numerical and scientific libraries NumPy and SciPy.

Install Scikit-learn:

Copy

pip install scikit-learn

Let's explore Scikit-learn with more detailed examples:

python

Copy

from sklearn.model\_selection import train\_test\_split

from sklearn.preprocessing import StandardScaler

from sklearn.linear\_model import LogisticRegression

from sklearn.tree import DecisionTreeClassifier

from sklearn.ensemble import RandomForestClassifier

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

from sklearn.datasets import load\_iris

import pandas as pd

*# Load a dataset*

iris = load\_iris()

X = iris.data

y = iris.target

*# Convert to DataFrame for better visualization*

df = pd.DataFrame(X, columns=iris.feature\_names)

df['target'] = y

print("Iris Dataset:")

print(df.head())

*# Split the data*

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

*# Preprocessing*

scaler = StandardScaler()

X\_train\_scaled = scaler.fit\_transform(X\_train)

X\_test\_scaled = scaler.transform(X\_test)

*# Logistic Regression*

lr\_model = LogisticRegression(random\_state=42)

lr\_model.fit(X\_train\_scaled, y\_train)

lr\_pred = lr\_model.predict(X\_test\_scaled)

print("\nLogistic Regression Results:")

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

print("Classification Report:")

print(classification\_report(y\_test, lr\_pred, target\_names=iris.target\_names))

*# Decision Tree*

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

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

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

print("\nDecision Tree Results:")

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

print("Classification Report:")

print(classification\_report(y\_test, dt\_pred, target\_names=iris.target\_names))

*# Random Forest*

rf\_model = RandomForestClassifier(random\_state=42)

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

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

print("\nRandom Forest Results:")

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

print("Classification Report:")

print(classification\_report(y\_test, rf\_pred, target\_names=iris.target\_names))

*# Feature Importance (for Random Forest)*

feature\_importance = pd.DataFrame({

'feature': iris.feature\_names,

'importance': rf\_model.feature\_importances\_

}).sort\_values('importance', ascending=False)

print("\nFeature Importance:")

print(feature\_importance)

This example demonstrates:

1. Loading a dataset (Iris)
2. Splitting the data into training and test sets
3. Preprocessing the data with StandardScaler
4. Training and evaluating multiple models (Logistic Regression, Decision Tree, Random Forest)
5. Comparing model performances
6. Extracting feature importance from the Random Forest model

These examples provide a more comprehensive introduction to Pandas and Scikit-learn, showcasing their capabilities in data manipulation, analysis, and machine learning.

## Part 3: AWS Integration

### 3.1 Setting Up AWS SDK for Python (Boto3)

1. Open a terminal/command prompt
2. Install Boto3 using pip:

Copy

pip install boto3

1. Create a new Python file named aws\_example.py

### 3.2 Configuring AWS Credentials

1. Create a file named credentials in ~/.aws/ (Linux/Mac) or C:\Users\YOUR\_USERNAME\.aws\ (Windows)
2. Add your AWS credentials to this file:

Copy

[default]

aws\_access\_key\_id = YOUR\_ACCESS\_KEY

aws\_secret\_access\_key = YOUR\_SECRET\_KEY

Replace YOUR\_ACCESS\_KEY and YOUR\_SECRET\_KEY with your actual AWS credentials.

### 3.3 Basic AWS Operations with Python

## Listing S3 Buckets

Add the following code to aws\_example.py:

python

Copy

import boto3

*# Create an S3 client*

s3 = boto3.client('s3')

*# List S3 buckets*

response = s3.list\_buckets()

print("S3 Buckets:")

for bucket in response['Buckets']:

print(f"- {bucket['Name']}")

Run the script:

Copy

python aws\_example.py

## Uploading a File to S3

Add this function to aws\_example.py:

python

Copy

def upload\_file(file\_name, bucket, object\_name=None):

if object\_name is None:

object\_name = file\_name

s3\_client = boto3.client('s3')

try:

s3\_client.upload\_file(file\_name, bucket, object\_name)

print(f"File {file\_name} uploaded successfully to {bucket}/{object\_name}")

except Exception as e:

print(f"Error uploading file: {e}")

return False

return True

*# Usage*

upload\_file('sample.txt', 'your-bucket-name')

Replace 'your-bucket-name' with your actual bucket name.

## Launching an EC2 Instance

Add this function to aws\_example.py:

python

Copy

def launch\_ec2\_instance():

ec2 = boto3.resource('ec2')

instances = ec2.create\_instances(

ImageId='ami-0aa7d40eeae50c9a9', *# Amazon Linux 2 AMI ID, may vary by region*

MinCount=1,

MaxCount=1,

InstanceType='t2.micro',

KeyName='your-key-pair-name' *# Replace with your key pair name*

)

print(f"New instance created: {instances[0].id}")

*# Usage*

launch\_ec2\_instance()

Replace 'your-key-pair-name' with the name of your EC2 key pair.

### 3.4 Using AWS Data Wrangler

AWS Data Wrangler is an open-source Python library that extends the power of Pandas to AWS, making it easier to integrate dataframes with AWS data-related services. It's designed to be a Swiss Army Knife for data engineering, data science, and analytics in AWS.

Install AWS Data Wrangler:

Copy

pip install awswrangler

Let's explore some examples of using AWS Data Wrangler with various AWS services:

## 3.4.1 Working with Amazon S3

Amazon S3 (Simple Storage Service) is an object storage service offering industry-leading scalability, data availability, security, and performance.

python

Copy

import awswrangler as wr

import pandas as pd

*# Create a sample DataFrame*

df = pd.DataFrame({

'id': [1, 2, 3],

'name': ['Alice', 'Bob', 'Charlie'],

'age': [25, 30, 35]

})

*# Write DataFrame to S3 as a CSV file*

wr.s3.to\_csv(

df=df,

path='s3://your-bucket-name/employees.csv'

)

print("Data written to S3")

*# Read CSV from S3*

df\_read = wr.s3.read\_csv('s3://your-bucket-name/employees.csv')

print("Data read from S3:")

print(df\_read)

*# List objects in an S3 bucket*

objects = wr.s3.list\_objects('s3://your-bucket-name/')

print("Objects in bucket:")

for obj in objects:

print(obj)

## 3.4.2 Working with Amazon Athena

Amazon Athena is an interactive query service that makes it easy to analyze data in Amazon S3 using standard SQL. It's serverless, so there's no infrastructure to manage.

python

Copy

*# Write DataFrame to an AWS Glue Data Catalog / Amazon Athena*

wr.s3.to\_parquet(

df=df,

path='s3://your-bucket-name/employees\_parquet/',

dataset=True,

database='your\_database',

table='employees'

)

print("Data written to Glue Data Catalog")

*# Read data from Athena*

df\_athena = wr.athena.read\_sql\_query(

"SELECT \* FROM employees WHERE age > 30",

database='your\_database'

)

print("Data read from Athena:")

print(df\_athena)

*# Get the query execution details*

query\_execution\_id = wr.athena.start\_query\_execution(

sql="SELECT \* FROM employees",

database='your\_database'

)

details = wr.athena.get\_query\_execution(query\_execution\_id)

print("Query execution details:", details)

## 3.4.3 Working with AWS Glue

AWS Glue is a fully managed extract, transform, and load (ETL) service that makes it easy to prepare and load data for analytics.

python

Copy

*# List all databases in Glue Data Catalog*

databases = wr.catalog.databases()

print("Glue databases:", databases)

*# List all tables in a specific database*

tables = wr.catalog.tables(database='your\_database')

print("Tables in your\_database:", tables)

*# Get the schema of a specific table*

schema = wr.catalog.table(database='your\_database', table='employees')

print("Schema of employees table:", schema)

## 3.4.4 Working with Amazon Redshift

Amazon Redshift is a fully managed, petabyte-scale data warehouse service in the cloud.

python

Copy

*# Assuming you have a Redshift cluster set up*

connection\_string = "redshift+psycopg2://username:password@host:port/database"

*# Write DataFrame to Redshift*

wr.redshift.to\_sql(

df=df,

table='employees',

schema='public',

con=connection\_string,

mode='overwrite'

)

print("Data written to Redshift")

*# Read data from Redshift*

df\_redshift = wr.redshift.read\_sql\_query(

sql="SELECT \* FROM public.employees",

con=connection\_string

)

print("Data read from Redshift:")

print(df\_redshift)

## 3.4.5 Working with Amazon QuickSight

Amazon QuickSight is a scalable, serverless, embeddable, machine learning-powered business intelligence (BI) service.

python

Copy

*# Create a QuickSight dataset from an Athena table*

response = wr.quicksight.create\_athena\_dataset(

name="EmployeesDataset",

database='your\_database',

table='employees',

account\_id='your-aws-account-id',

region='your-aws-region'

)

print("QuickSight dataset created:", response)

*# List QuickSight datasets*

datasets = wr.quicksight.list\_datasets(account\_id='your-aws-account-id')

print("QuickSight datasets:", datasets)

These examples demonstrate how AWS Data Wrangler can be used to interact with various AWS services, making it easier to work with data across the AWS ecosystem. Remember to replace placeholders like 'your-bucket-name', 'your\_database', 'your-aws-account-id', and 'your-aws-region' with your actual AWS resource names and identifiers.

## Conclusion

In this comprehensive tutorial, we've covered:

1. Python fundamentals, including data structures and control flow
2. Introduction to machine learning libraries: NumPy, Pandas, and Scikit-learn
3. AWS integration using Boto3 and AWS Data Wrangler

We've explored how to use AWS Data Wrangler with various AWS services, including:

* Amazon S3 for object storage
* Amazon Athena for querying data in S3
* AWS Glue for ETL operations and data catalog management
* Amazon Redshift for data warehousing
* Amazon QuickSight for business intelligence

These skills provide a solid foundation for developing Python applications, performing data analysis, and working with AWS services. AWS Data Wrangler simplifies many data operations across AWS services, making it an invaluable tool for data engineers and analysts working in the AWS ecosystem.

Remember to always clean up your AWS resources after you're done experimenting to avoid unnecessary charges. This includes deleting S3 objects, stopping Athena queries, and terminating Redshift clusters if they're no longer needed.

As you continue to work with these tools, refer to the official documentation for AWS Data Wrangler and the respective AWS services for more advanced usage and best practices.