# Python for Data Analysis

## From Fundamentals to Advanced Techniques

A comprehensive guide covering all essential topics for data analysis using Python, structured in a logical flow from basics to advanced concepts. Master the tools and techniques needed to extract insights from data effectively.

Python Basics

Data Analysis

Visualization

Machine Learning

# Python Fundamentals

Essential concepts for data analysis

## {} Data Types & Variables

Python supports various data types:

```
# Basic data types
int_var = 42 # Integer
float_var = 3.14 # Float
string_var = "Data" # String
bool_var = True # Boolean
```

## Operators & Expressions

Mathematical, comparison, and logical operators:

```
# Arithmetic operators
sum = 10 + 5 # Addition
product = 10 * 5 #
Multiplication
# Comparison operators
result = 10 > 5 # True
```

## Control Structures

Conditional statements and loops:

```
# If statement
if x > 0:
  print("Positive")
# For loop
for i in range(5):
  print(i)
```

## Σ Functions & Modules

Reusable code blocks and imports:

```
# Function definition
def add_numbers(a, b):
  return a + b
# Module import
import math
result = math.sqrt(16)
```

## Data Structures

Collections for storing multiple values:

```
# List (ordered, mutable)
my_list = [1, 2, 3]
# Dictionary (key-value pairs)
my_dict = {"name": "John", "age": 30}
# Tuple (ordered, immutable)
my_tuple = (1, 2, 3)
```

# Core Python Libraries (Part 1)

NumPy and Pandas for efficient data manipulation

## NumPy

### Array Creation

Create multi-dimensional arrays efficiently

```
import numpy as np
# Create arrays
arr = np.array([1, 2, 3])
zeros = np.zeros((3, 4))
random = np.random.rand(2, 3)
```

### Array Operations

Perform mathematical operations on arrays

```
# Element-wise operations
result = arr1 + arr2
product = arr1 * arr2
# Mathematical functions
sqrt_arr = np.sqrt(arr)
```

### Broadcasting

Perform operations on arrays of different shapes

```
# Broadcasting example
arr = np.array([[1, 2, 3],
[4, 5, 6]])
scalar = 10
result = arr + scalar
# Adds 10 to each element
```

### Aggregation

Compute summary statistics

```
# Statistical operations
mean_val = np.mean(arr)
max_val = np.max(arr)
sum_val = np.sum(arr, axis=0)
std_dev = np.std(arr)
```

## Pandas

### Data Structures

Series (1D) and DataFrame (2D) data structures

```
import pandas as pd
# Create Series and DataFrame
s = pd.Series([1, 3, 5,
np.nan])
df = pd.DataFrame({
  'A': [1, 2, 3],
  'B': ['a', 'b', 'c']})
```

### Data Selection

Select, filter, and slice data

```
# Selecting data
df['A'] # Column selection
df.loc[0] # Row by label
df.iloc[0] # Row by position
df[df['A'] > 1] # Filtering
```

### Grouping & Aggregation

Group data and compute aggregations

```
# Grouping operations
grouped =
df.groupby('category')
mean_vals = grouped.mean()
count_vals = grouped.count()
# Multiple aggregations
result = grouped.agg(['mean',
'std'])
```

### Handling Missing Data

Detect, remove, or fill missing values

```
# Missing data operations
df.isnull().sum() # Count
nulls
df.dropna() # Drop missing
df.fillna(0) # Fill with 0
df.interpolate() #
Interpolate
```

# Core Python Libraries (Part 2)

Matplotlib and Seaborn for effective data visualization

## Matplotlib

### Basic Plots

Create fundamental plot types

```
import matplotlib.pyplot as
plt
# Line plot
plt.plot(x, y)
# Bar chart
plt.bar(categories, values)
# Scatter plot
plt.scatter(x, y)
```

### Customization

Modify plot appearance and labels

```
# Customizing plots
plt.title("Data Analysis")
plt.xlabel("X-axis")
plt.ylabel("Y-axis")
plt.legend(["Series 1"])
plt.grid(True)
```

### Subplots

Create multiple plots in one figure

```
# Creating subplots
fig, axes = plt.subplots(2,
2)
axes[0, 0].plot(x, y)
axes[0, 1].bar(categories,
values)
axes[1, 0].scatter(x, y)
axes[1, 1].hist(data)
```

### Advanced Plots

Create specialized visualizations

```
# Histogram
plt.hist(data, bins=20)
# Box plot
plt.boxplot([data1, data2])
# Contour plot
plt.contour(X, Y, Z)
```

## Seaborn

### Statistical Plots

Create sophisticated statistical visualizations

```
import seaborn as sns
# Distribution plot
sns.distplot(data)
# Joint plot
sns.jointplot(x='col1',
y='col2', data=df)
```

### Matrix Plots

Visualize matrix data and correlations

```
# Heatmap
sns.heatmap(df.corr(),
annot=True)
# Clustermap
sns.clustermap(df.corr())
# Pair plot
sns.pairplot(df)
```

### Categorical Plots

Visualize categorical data relationships

```
# Box plot
sns.boxplot(x='category',
y='value', data=df)
# Violin plot
sns.violinplot(x='category',
y='value', data=df)
# Count plot
sns.countplot(x='category',
data=df)
```

### Aesthetic Control

Enhance visual appeal with themes and palettes

```
# Set style
sns.set_style("whitegrid")
# Color palette
sns.set_palette("husl")
# Context scaling
sns.set_context("notebook")
```

# Data Handling

Loading, preprocessing, and transforming data for analysis

## Data Loading

### File Formats

Load data from common file formats

```
import pandas as pd
# CSV files
df = pd.read_csv('data.csv')
# Excel files
df =
pd.read_excel('data.xlsx')
# JSON files
df =
pd.read_json('data.json')
```

### Databases & Web

Connect to databases and scrape web data

```
import sqlite3
import requests
from bs4 import BeautifulSoup
# SQL databases
conn =
sqlite3.connect('db.sqlite')
df = pd.read_sql('SELECT *
FROM table', conn)
```

## Data Preprocessing

### Missing Values

Handle missing data appropriately

```
 # Check for missing values
df.isnull().sum()
# Drop missing values
df.dropna()
# Fill missing values
df.fillna(value=0)
df.fillna(method='ffill')
```

### Outliers & Duplicates

Identify and handle data anomalies

```
 # Remove duplicates
df.drop_duplicates()
# Detect outliers with Z-score
from scipy import stats
z = np.abs(stats.zscore(df))
df = df[(z < 3).all(axis=1)]
```

## Data Transformation

### Scaling & Normalization

Standardize numerical features

```
 from sklearn.preprocessing import StandardScaler, MinMaxScaler
# Standardization
scaler = StandardScaler()
scaled_data = scaler.fit_transform(df)
# Normalization
normalizer = MinMaxScaler()
normalized_data = normalizer.fit_transform(df)
```

### Categorical Encoding

Convert categorical data to numerical

```
 # Label encoding
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
df['category'] = le.fit_transform(df['category'])
# One-hot encoding
pd.get_dummies(df['category'])
```

# Data Visualization Techniques

Choosing the right visualization for your data

## Basic Plots

### Line & Bar Charts

Line charts show trends over time; bar charts compare categories

```
# Line chart for time series
plt.plot(dates, values)
# Bar chart for categories
plt.bar(categories, values)
# Horizontal bar chart
plt.barh(categories, values)
```

### Scatter & Pie Charts

Scatter plots show relationships; pie charts show proportions

```
# Scatter plot with color mapping
plt.scatter(x, y, c=colors)
# Pie chart with percentages
plt.pie(sizes, labels=labels, autopct='%1.1f%%')
# Bubble chart (sized scatter)
plt.scatter(x, y, s=sizes)
```

## Statistical Plots

### Distribution Plots

Visualize data distribution and spread

```
# Histogram for frequency distribution
plt.hist(data, bins=20)
# Box plot for quartiles and outliers
plt.boxplot([data1, data2])
# Violin plot (box + density)
sns.violinplot(x='group', y='value', data=df)
```

### Matrix & Correlation Plots

Visualize relationships between multiple variables

```
# Heatmap for correlations
sns.heatmap(df.corr(), annot=True)
# Pair plot for multiple variables
sns.pairplot(df, hue='category')
# Clustermap for hierarchical clustering
sns.clustermap(df.corr())
```

## Advanced Visualization

### Interactive Plots

Create interactive visualizations with Plotly and Bokeh

```
import plotly.express as px
# Interactive scatter plot
fig = px.scatter(df, x='col1', y='col2', color='category')
fig.show()
# Interactive line chart
fig = px.line(df, x='date', y='value')
```

### Geospatial Visualization

Visualize data on maps and geographical contexts

```
import folium
# Create base map
m = folium.Map(location=[lat, lon], zoom_start=10)
# Add markers
folium.Marker([lat, lon], popup='Location').add_to(m)
# Choropleth map
folium.Choropleth(geo_data, data=df).add_to(m)
```

# Data Analysis Techniques

Methods to extract insights and patterns from data

## Exploratory Data Analysis (EDA)

### Univariate Analysis

Analyze individual variables to understand their distribution and characteristics

```
# Summary statistics
df.describe()
# Distribution visualization
sns.histplot(df['column'])
# Box plot for outliers
sns.boxplot(x=df['column'])
```

### Bivariate & Multivariate Analysis

Examine relationships between multiple variables

```
# Correlation matrix
correlation = df.corr()
# Scatter plot for relationships
sns.scatterplot(x='col1', y='col2', data=df)
# Pair plot for multiple variables
sns.pairplot(df)
```

## Σ Statistical Analysis

### Correlation & Covariance

Measure relationships between variables

```
# Pearson correlation
corr = df['col1'].corr(df['col2'])
# Covariance matrix
cov_matrix = df.cov()
# Heatmap of correlations
sns.heatmap(df.corr(), annot=True)
```

### Hypothesis Testing

Test assumptions about population parameters

```
from scipy import stats
# T-test
t_stat, p_val = stats.ttest_ind(sample1, sample2)
# ANOVA
f_val, p_val = stats.f_oneway(group1, group2, group3)
# Chi-square test
chi2, p_val = stats.chi2_contingency(observed)
```

## Dimensionality Reduction

### Principal Component Analysis (PCA)

Reduce dimensions while preserving variance

```
from sklearn.decomposition import PCA
# Apply PCA
pca = PCA(n_components=2)
principal_components = pca.fit_transform(X)
# Explained variance
print(pca.explained_variance_ratio_)
```

### Feature Selection

Select most relevant features for analysis

```
from sklearn.feature_selection import SelectKBest, f_classif
# Select best features
selector = SelectKBest(score_func=f_classif, k=5)
X_new = selector.fit_transform(X, y)
# Get selected feature indices
selected_features = selector.get_support(indices=True)
```

# Machine Learning Fundamentals

Key concepts and algorithms for data analysis

## Supervised Learning

### Regression

Predict continuous numerical values

```
from sklearn.linear_model import LinearRegression
# Linear regression
model = LinearRegression()
model.fit(X_train, y_train)
predictions = model.predict(X_test)
# Polynomial regression
from sklearn.preprocessing import PolynomialFeatures
```

### Classification

Predict discrete categorical labels

```
from sklearn.ensemble import RandomForestClassifier
# Random Forest
model = RandomForestClassifier(n_estimators=100)
model.fit(X_train, y_train)
# Logistic Regression
from sklearn.linear_model import LogisticRegression
```

## Unsupervised Learning

### Clustering

Group similar data points together

```
from sklearn.cluster import
KMeans
# K-means clustering
kmeans = KMeans(n_clusters=3)
clusters =
kmeans.fit_predict(X)
# Hierarchical clustering
from sklearn.cluster import
AgglomerativeClustering
```

### Dimensionality Reduction

Reduce features while preserving information

```
from sklearn.decomposition
import PCA
# Principal Component
Analysis
pca = PCA(n_components=2)
X_pca = pca.fit_transform(X)
# t-SNE for visualization
from sklearn.manifold import
TSNE
```

## Model Evaluation

### Cross-Validation

Assess model performance robustly

```
from sklearn.model_selection
import cross_val_score
# K-fold cross-validation
scores =
cross_val_score(model, X, y,
cv=5)
# Stratified K-fold
from sklearn.model_selection
import StratifiedKFold
```

### Evaluation Metrics

Quantify model performance

```
from sklearn.metrics import
accuracy_score,
precision_score
# Classification metrics
accuracy =
accuracy_score(y_true,
y_pred)
precision =
precision_score(y_true,
y_pred)
# Regression metrics
```

# Advanced Topics and Resources

Expanding your Python data analysis capabilities

## Deep Learning Basics

### Neural Networks

Multi-layered networks for complex pattern recognition

```
import tensorflow as tf
# Simple neural network
model = tf.keras.Sequential([
  tf.keras.layers.Dense(128, activation='relu'),
  tf.keras.layers.Dense(10, activation='softmax')
])
```

### Frameworks

Popular libraries for deep learning implementation

```
# TensorFlow example
model.compile(optimizer='adam',
  loss='sparse_categorical_crossentropy',
  metrics=['accuracy'])
# PyTorch example
import torch
import torch.nn as nn
```

## Big Data Handling

### PySpark

Python API for Apache Spark distributed computing

```
from pyspark.sql import SparkSession
# Create Spark session
spark = SparkSession.builder.appName('DataAnalysis').getOrCreate()
# Load data
df = spark.read.csv('large_dataset.csv', header=True)
```

### Distributed Operations

Perform operations across distributed datasets

```
# PySpark transformations
filtered_df = df.filter(df['age'] > 25)
grouped_df = df.groupBy('category').count()
# SQL operations
df.createOrReplaceTempView('data')
result = spark.sql('SELECT * FROM data WHERE value > 100')
```

## Web Frameworks & Resources

### Flask & Django

Build web applications for data visualization and analysis

```
# Flask example
from flask import Flask, render_template
app = Flask(__name__)
@app.route('/')
def home():
  return render_template('dashboard.html')
```

### Learning Resources

Continue your Python data analysis journey

- Python Data Science Handbook
- Coursera Data Science Courses
- Kaggle Learn
- PyData Community
- DataCamp
- Stack Overflow