Python packages

Shucheng Liao*

May 2024

Contents

1 Numpy **2**

2 Scipy **4**

3 Pandas Dataframe **7**

4 Requests **10**

5 Beautifulsoup4 **11**

6 Matplotlib **14**

7 Seaborn **17**

8 Scikit-learn **20**

9 Thank You Note **23**

*NYU Stern, MS in Quantitative Economics

1 Numpy

```python
import numpy as np

## Create a NumPy array
arr = np.array([1, 2, 3, 4, 5])
print(arr)
# Expected output: [1 2 3 4 5]

## Create an array of zeros/ones
zeros = np.zeros((2, 3)) # np.ones((2,3))
print(zeros)
# Expected output:
# [[0. 0. 0.]
#  [0. 0. 0.]]

## Create an array with a range of values
range_arr = np.arange(10)
print(range_arr)
# Expected output: [0 1 2 3 4 5 6 7 8 9]

## Create an array with evenly spaced values
linspace_arr = np.linspace(0, 1, 5)
print(linspace_arr)
# Expected output: [0.   0.25 0.5  0.75 1.  ]

## Reshape an array
reshaped = np.reshape(range_arr, (2, 5))
print(reshaped)
# Expected output:
# [[0 1 2 3 4]
#  [5 6 7 8 9]]

## Basic arithmetic operations
a = np.array([1, 2, 3])
b = np.array([4, 5, 6])

print(a + b)
# Expected output: [5 7 9]

print(a - b)
# Expected output: [-3 -3 -3]

print(a * b)
# Expected output: [4 10 18]

print(a / b)
# Expected output: [0.25 0.4  0.5 ]

## Element-wise square root
sqrt_arr = np.sqrt(a)
print(sqrt_arr)
# Expected output: [1.         1.41421356 1.73205081]

## Dot product of two arrays
dot_product = np.dot(a, b)
print(dot_product)
# Expected output: 32

## Sum of all elements
sum_all = np.sum(a)
print(sum_all)
# Expected output: 6

## Sum along an axis
matrix = np.array([[1, 2], [3, 4]])
sum_axis0 = np.sum(matrix, axis=0)
```

```
print(sum_axis0)
# Expected output: [4 6]

sum_axis1 = np.sum(matrix, axis=1)
print(sum_axis1)
# Expected output: [3 7]

## Transpose of an array
transpose = np.transpose(matrix)
print(transpose)
# Expected output:
# [[1 3]
#  [2 4]]

## Boolean indexing
bool_index = a > 2
print(bool_index)
# Expected output: [False False  True]

filtered = a[a > 2]
print(filtered)
# Expected output: [3]
```

Back to Table of Contents

2 Scipy

1. Optimization

```
from scipy.optimize import minimize

# Define a quadratic function
def func(x):
    return x**2 + 2*x + 1

# Minimize the function
result = minimize(func, 0) # initial guess 0
print(result.x)
# Expected output: [-1.]
```

Finds the minimum of the quadratic function, which is at x = -1.

2. Integration

```
from scipy.integrate import quad

# Define a simple function
def integrand(x):
    return x**2

# Integrate the function from 0 to 1
result, error = quad(integrand, 0, 1)
print(result)
print(error)
# Expected output:
# result: 0.3333333333333333
# error: 3.700743415417189e-15
```

Computes the integral of x^2 from 0 to 1, which is 1/3. The error term gives an estimate of the numerical integration error, which is very small in this case.

3. Interpolation

```
from scipy.interpolate import interp1d
import numpy as np

# Define data points
x = np.array([0, 1, 2, 3])
y = np.array([0, 1, 0, 1])

# 1. Linear Interpolation
f_linear = interp1d(x, y, kind='linear')
print(f_linear(1.5))   # estimate the value of y at x = 1.5
# Expected output: 0.5
```

Linear Interpolation: Connects data points with straight lines, simple and fast. Commonly used for quick approximations.

```
# 2. Nearest-Neighbor Interpolation
f_nearest = interp1d(x, y, kind='nearest')
print(f_nearest(1.5))
# Expected output: 0.0
```

Nearest-Neighbor Interpolation: Uses the nearest data point's value. Useful for categorical data or when a step function is desired.

```
# 3. Zero-Order Hold (Previous)
f_zero = interp1d(x, y, kind='zero')
```

```
print(f_zero(1.5))
# Expected output: 1.0
```

Zero-Order Hold: Uses the previous data point's value (step function). Suitable for systems where the value remains constant until the next data point.

```
# 4. Spline Interpolation of Order 1
f_slinear = interp1d(x, y, kind='slinear')
print(f_slinear(1.5))
# Expected output: 0.5
```

Spline Interpolation of Order 1: Similar to linear interpolation but uses splines for a smoother transition. It can handle more complex data structures better than simple linear interpolation in some cases.

```
# 5. Quadratic/Cubic Spline Interpolation
f_quadratic = interp1d(x, y, kind='quadratic') # 'cubic'
print(f_quadratic(1.5))
# Expected output: Depends on the quadratic fit
```

Quadratic/Cubic Spline Interpolation: Fits a quadratic/cubic polynomial between data points.

4. Linear Algebra

```
from scipy.linalg import eig
import numpy as np

# Define a matrix
A = np.array([[1, 2], [3, 4]])

# Compute eigenvalues and eigenvectors
values, vectors = eig(A)
print(values)
# Expected output: Array of eigenvalues [5.37228132 -0.37228132]
print(vectors)
# Expected output: Matrix of eigenvectors
# [[-0.41597356 -0.82456484]
#  [-0.90937671  0.56576746]]
```

Computes the eigenvalues and eigenvectors of matrix A.

5. Statistical Functions

```
from scipy.stats import norm

# Define a normal distribution
mean, std_dev = 0, 1
dist = norm(mean, std_dev)

# Compute the cumulative distribution function at 0
print(dist.cdf(0))  #(P(X <= 0))  # (dist.logcdf(0))
# Expected output: 0.5

# Compute the probability density function at 0
print(dist.pdf(0))  # (dist.logpdf(0))
# Expected output: 0.3989422804014327
```

Computes the CDF/PDF of the standard normal distribution at x = 0.

7. Spatial Data

```python
from scipy.spatial import distance

# Define two points
point1 = [0, 0]
point2 = [3, 4]

# Compute Euclidean distance
dist = distance.euclidean(point1, point2)
print(dist)
# Expected output: 5.0
```

Computes the Euclidean distance between two points.

Back to Table of Contents

3 Pandas Dataframe

1. Reading Data

```
import pandas as pd

# Reading data from a CSV file
df = pd.read_csv('economic_data.csv')
# or pd.read_excel or pd.read_stata
print(df.head())     # First 5 rows of the CSV file
```

2. Inspecting Data

```
# Inspecting the DataFrame
print(df.info())
# Expected output: Summary of the DataFrame, including column names, non-null
    counts, and data types

print(df.describe())
# Expected output: Descriptive statistics for numerical columns: count, mean, std,
    min, max, percentiles

# Descriptive statistics for categorical data
print(df.describe(include=['object']))
# Expected output: count, unique, top, and frequency for catagorical columns
```

3. Selecting Data

```
# Selecting specific column
gdp_series = df['GDP']
print(gdp_series.head())
# Expected output: First 5 values in the 'GDP' column

# Selecting specific subset
subset = df[['GDP', 'Inflation']]
print(subset.head())
# Expected output: First 5 rows of the DataFrame with 'GDP' and 'Inflation' columns
```

4. Filtering Data

```
# Filtering rows based on a condition
filtered_df = df[df['GDP'] > 10000]
print(filtered_df.head())
# Expected output: Rows where GDP is greater than 10,000
```

5. Adding New Columns

```
# Adding a new column
df['GDP per Capita'] = df['GDP'] / df['Population']
print(df.head())
# Expected output: DataFrame with a new 'GDP per Capita' column
```

6. Grouping Data

```
# Grouping data
grouped_df = df.groupby('Country')['GDP'].sum()
print(grouped_df)
# Expected output: Sum of GDP for each country

# Groupiog data and applying an aggregation function
grouped_df = df.groupby('Country')['GDP'].agg(['mean', 'sum', 'min', 'max'])
# Expected output: DataFrame with mean, sum, min, and max GDP for each country
```

7. Merging DataFrames

```
# Merging two DataFrames
df1 = pd.DataFrame({'Country': ['A', 'B'], 'GDP': [1000, 2000]})
df2 = pd.DataFrame({'Country': ['A', 'B'], 'Population': [10, 20]})
merged_df = pd.merge(df1, df2, on='Country')

# Extension: how='inner'/'outer'/'left'/'right' - includes rows with matched keys
    only/ all rows with unmatched keys marked as 'NaN'/ all rows from left dt and
    matched rows from right dt/ all rows from right dt and matched rows from left dt

print(merged_df)
# Expected output: Merged DataFrame with GDP and Population for each country
```

8. Handling Missing Data

```
# Fill missing values with a specific value or method
merged_df.fillna(0, inplace=True)

# Drop rows with any missing values
merged_df.dropna(inplace=True)

# Handling missing data - filled with the mean GDP
df['GDP'].fillna(df['GDP'].mean(), inplace=True)
print(df.head())
```

9. Time Series Analysis

```
# form of date to be convert (example): ['01-01-2024', '15-02-2024']

# Converting a column to datetime
df['Date'] = pd.to_datetime(df['Date'])

# Extension: format='%d-%m-%Y' - switching the orders

# Extension: errors = 'coerce'/'raise'/ignore' - convert the invalid to NaT/raise an
    error/return original if fail to convert

# Setting it as the index
df.set_index('Date', inplace=True)

print(df.head())
# Expected output: DataFrame with 'Date' column as the index
```

10. Resampling Time Series Data

```
# Resampling time series data to a different frequency
monthly_df = df.resample('M').mean()
print(monthly_df.head())
# Expected output: DataFrame resampled to monthly frequency with mean values
```

11. Exporting DataFrame to Different File Formats

```
# Export to CSV
df.to_csv('output.csv', index=False)
# Saves the DataFrame to a CSV file named 'output.csv' without the index.

# Extension: to_excel/to_stata
```

Back to Table of Contents

4 Requests

```
# Sending a GET request and checking for errors
response = requests.get('https://api.example.com/data')
try:
    response.raise_for_status()
    print('Request was successful')
    # Expected output: 'Request was successful' if the status code indicates success
except requests.HTTPError as e:
    print(f'HTTP error occurred: {e}')
    # Expected output: 'HTTP error occurred: ...' with details of the error
```

Back to Table of Contents

5 Beautifulsoup4

Quick note: Extract text from HTML requests

```
import requests
from bs4 import BeautifulSoup

# Send a GET request
response = requests.get('https://www.example.com')

# Check if the request was successful
if response.status_code == 200:
    # Parse the HTML content
    soup = BeautifulSoup(response.content, 'html.parser')

    # Extract text from the HTML content
    text = soup.get_text()

    print(text)
    # Expected output: All text content extracted from the HTML page
else:
    print(f'Failed to retrieve data: {response.status_code}')
    # Expected output: Failure message with HTTP status code
```

1. Creating a BeautifulSoup Object

```
from bs4 import BeautifulSoup

# Sample HTML content
html_content = '''
<html>
    <head><title>Economic Data</title></head>
    <body>
        <h1>Economic Indicators</h1>
        <p id="gdp">GDP: $20 Trillion</p>
        <p id="inflation">Inflation: 2%</p>
    </body>
</html>
'''

# Creating a BeautifulSoup object
soup = BeautifulSoup(html_content, 'html.parser')
print(soup.prettify())
# Expected output: Formatted HTML content
```

2. Accessing Elements by Tag Name

```
# Accessing elements by tag name
title = soup.title
print(title)
# Expected output: <title>Economic Data</title>

print(title.text)
# Expected output: Economic Data
```

3. Finding Elements by ID

```
# Finding elements by ID
gdp = soup.find(id='gdp')
print(gdp)
```

```
# Expected output: <p id="gdp">GDP: $20 Trillion</p>

print(gdp.text)
# Expected output: GDP: $20 Trillion
```

4. Finding Elements by Class Name

```
# Sample HTML content with class names
html_content = '''
<html>
    <body>
        <div class="economic-indicator">GDP: $20 Trillion</div>
        <div class="economic-indicator">Inflation: 2%</div>
    </body>
</html>
'''

# Creating a BeautifulSoup object
soup = BeautifulSoup(html_content, 'html.parser')

# Finding elements by class name
indicators = soup.find_all(class_='economic-indicator')
for indicator in indicators:
    print(indicator.text)
# Expected output:
# GDP: $20 Trillion
# Inflation: 2%
```

5. Extracting Links (Anchor Tags)

```
# Sample HTML content with links
html_content = '''
<html>
    <body>
        <a href="https://example.com/gdp">GDP Report</a>
        <a href="https://example.com/inflation">Inflation Report</a>
    </body>
</html>
'''

# Creating a BeautifulSoup object
soup = BeautifulSoup(html_content, 'html.parser')

# Extracting all links
links = soup.find_all('a')
for link in links:
    print(link.get('href'))
# Expected output:
# https://example.com/gdp
# https://example.com/inflation
```

6. Navigating the Parse Tree

```
header = soup.body.h1
print(header)
# Expected output: <h1>Economic Indicators</h1>

paragraphs = soup.body.find_all('p')
for para in paragraphs:
    print(para.text)
# Expected output:
```

```
# GDP: $20 Trillion
# Inflation: 2%
```

7. Modifying the HTML Content

```
soup.body.h1.string = "Updated Economic Indicators"
print(soup.body.h1)
# Expected output: <h1>Updated Economic Indicators</h1>

new_tag = soup.new_tag("p", id="unemployment")
new_tag.string = "Unemployment: 5%"
soup.body.append(new_tag)
print(soup.body)
# Expected output: <body>...<p id="unemployment">Unemployment: 5%</p></body>
```

8. Removing Elements

```
for tag in soup.find_all('p'):
    tag.decompose()

print(soup.body)
# Expected output: <body>...</body> without <p> tags
```

Back to Table of Contents

6 Matplotlib

see different shapes of markers

see different named colors

see different line styles

1. Basic Line Plot

```
import matplotlib.pyplot as plt

# Sample economic data
years = [2016, 2017, 2018, 2019, 2020]
gdp = [1.5, 2.3, 2.9, 2.3, -3.5]

# Creating a basic line plot
plt.plot(years, gdp, marker='o')
plt.title('GDP Growth Over Years')
plt.xlabel('Year')
plt.ylabel('GDP Growth (%)')
plt.grid(True)
plt.show()

# Expected output: a basic line plot of GDP growth over the years with markers at
    data points
```

2. Bar Chart

```
import matplotlib.pyplot as plt

# Sample economic data
countries = ['USA', 'China', 'Japan', 'Germany', 'India']
gdp = [21.43, 14.34, 5.08, 3.84, 2.87]

# Creating a bar chart
plt.bar(countries, gdp, color='skyblue')
plt.title('GDP of Top 5 Economies in 2020')
plt.xlabel('Country')
plt.ylabel('GDP (Trillions USD)')
plt.show()

# Expected output: Bar chart showing GDP of the top 5 economies in 2020
```

3. Pie Chart

```
import matplotlib.pyplot as plt

# Sample economic data
sectors = ['Agriculture', 'Industry', 'Services']
gdp_share = [4, 33, 63]

# Creating a pie chart
plt.pie(gdp_share, labels=sectors, autopct='%1.1f%%', startangle=90)
plt.title('GDP Share by Sector')
plt.show()

# Expected output: Pie chart showing GDP share by sector
# '%1.0f%%'/'%1.1f%%'/'%1.2f%%': 0/1/2 decimal place(s)
# startangle=90: This will start the first pie slice from the top
```

4. Scatter Plot

```
import matplotlib.pyplot as plt

# Sample economic data
education_spending = [4.1, 3.5, 3.8, 3.2, 3.0]
gdp_growth = [2.9, 1.6, 2.2, 2.0, 1.8]

# Creating a scatter plot
plt.scatter(education_spending, gdp_growth, color='red')
plt.title('Education Spending vs GDP Growth')
plt.xlabel('Education Spending (% of GDP)')
plt.ylabel('GDP Growth (%)')
plt.grid(True)
plt.show()

# Expected output: Scatter plot showing the relationship between education spending
    and GDP growth
```

5. Histogram

```
import matplotlib.pyplot as plt
import numpy as np

# Sample economic data
inflation_rates = np.random.normal(3, 1, 1000)  # Generate 1000 random data points

# Creating a histogram
plt.hist(inflation_rates, bins=20, color='green', edgecolor='black')
plt.title('Inflation Rate Distribution')
plt.xlabel('Inflation Rate (%)')
plt.ylabel('Frequency')
plt.show()
# Expected output: Histogram showing the distribution of inflation rates
```

6. Time Series Plot

```
import matplotlib.pyplot as plt
import pandas as pd

# Sample time series data
date_range = pd.date_range(start='1/1/2020', periods=12, freq='M')
unemployment_rate = [5.1, 5.2, 5.0, 4.9, 5.3, 5.5, 5.6, 5.4, 5.3, 5.2, 5.1, 5.0]

# Creating a time series plot
plt.plot(date_range, unemployment_rate, marker='o', linestyle='-')
plt.title('Monthly Unemployment Rate in 2020')
plt.xlabel('Month')
plt.ylabel('Unemployment Rate (%)')
plt.grid(True)
plt.show()
# Expected output: Time series plot showing the monthly unemployment rate in 2020
```

7. Adding Annotations

```
import matplotlib.pyplot as plt

# Sample economic data
years = [2016, 2017, 2018, 2019, 2020]
gdp = [1.5, 2.3, 2.9, 2.3, -3.5]

```

```
# Creating a line plot with annotations
plt.plot(years, gdp, marker='o')
plt.title('GDP Growth Over Years')
plt.xlabel('Year')
plt.ylabel('GDP Growth (%)')
plt.grid(True)

# Adding annotation
plt.annotate('COVID-19 Impact', xy=(2020, -3.5), xytext=(2018, 0),
             arrowprops=dict(facecolor='black', shrink=0.05))
plt.show()
# Expected output: Line plot with an annotation highlighting the impact of COVID-19
    in 2020
```

8. Adding More Stuff

```
import matplotlib.pyplot as plt

# Sample economic data
years = [2016, 2017, 2018, 2019, 2020]
gdp = [1.5, 2.3, 2.9, 2.3, -3.5]

# Creating a basic line plot
plt.plot(years, gdp, marker='o', label='GDP Growth')

# Adding a horizontal line at y=0 with a dashed line style
plt.axhline(y=0, color='gray', linestyle='--')

# Adding a vertical line at x=2018 with a dotted line style
plt.axvline(x=2018, color='blue', linestyle=':')

# Adding a marker at a specific coordinate (2019, 2.3)
plt.plot(2019, 2.3, marker='x', markersize=10, color='red')

# Adding a 45-degree line (y=x)
x_values = range(2016, 2021)
plt.plot(x_values, x_values, color='green', linestyle='-.', label='45-degree line')

# Adding a legend
plt.legend()

plt.title('GDP Growth Over Years with Enhancements')
plt.xlabel('Year')
plt.ylabel('GDP Growth (%)')
plt.grid(True)
plt.show()
```

Back to Table of Contents

7 Seaborn

see different color palettes

1. Importing Seaborn and Matplotlib

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

2. Setting Aesthetic Style

```
sns.set(style="whitegrid")
# Sets the aesthetic style of the plots to whitegrid, which is useful for economic
    data visualizations.
```

3. Line Plot

```
# Sample economic data
years = [2016, 2017, 2018, 2019, 2020]
gdp = [1.5, 2.3, 2.9, 2.3, -3.5]

# Creating a line plot
sns.lineplot(x=years, y=gdp)
plt.title('GDP Growth Over Years')
plt.xlabel('Year')
plt.ylabel('GDP Growth (%)')
plt.show()
# Expected output: Line plot showing GDP growth over the years.
```

4. Bar Plot

```
# Sample economic data
countries = ['USA', 'China', 'Japan', 'Germany', 'India']
gdp = [21.43, 14.34, 5.08, 3.84, 2.87]

# Creating a bar plot
sns.barplot(x=countries, y=gdp, palette="Blues_d")
plt.title('GDP of Top 5 Economies in 2020')
plt.xlabel('Country')
plt.ylabel('GDP (Trillions USD)')
plt.show()
# Expected output: Bar plot showing GDP of the top 5 economies in 2020.
```

5. Histogram

```
import numpy as np

# Sample economic data
inflation_rates = np.random.normal(3, 1, 1000)  # Generate 1000 random data points

# Creating a histogram
sns.histplot(inflation_rates, bins=20, kde=True, color="green")
plt.title('Inflation Rate Distribution')
plt.xlabel('Inflation Rate (%)')
plt.ylabel('Frequency')
plt.show()
```

```
# Expected output: Histogram showing the distribution of inflation rates with KDE:
    this parameter adds a KDE plot over the histogram, providing a smooth estimate
    of the distribution of the data, especially when dealing with continuous data
```

6. Scatter Plot

```
# Sample economic data
education_spending = [4.1, 3.5, 3.8, 3.2, 3.0]
gdp_growth = [2.9, 1.6, 2.2, 2.0, 1.8]

# Creating a scatter plot
sns.scatterplot(x=education_spending, y=gdp_growth, color="red")
plt.title('Education Spending vs GDP Growth')
plt.xlabel('Education Spending (% of GDP)')
plt.ylabel('GDP Growth (%)')
plt.show()
# Expected output: Scatter plot showing the relationship between education spending
    and GDP growth.
```

7. Pair Plot

```
# Sample economic data
data = {
    'GDP': [1.5, 2.3, 2.9, 2.3, -3.5],
    'Inflation': [2.1, 1.8, 2.5, 1.9, 0.5],
    'Unemployment': [5.1, 4.9, 4.7, 5.2, 6.1]
}
df = pd.DataFrame(data, index=[2016, 2017, 2018, 2019, 2020])

# Creating a pair plot
sns.pairplot(df)
plt.show()
# Expected output: Pair plot showing relationships between GDP, inflation, and
    unemployment.
```

8. Heatmap

```
# Sample economic data
data = np.array([[1.5, 2.3, 2.9], [2.1, 1.8, 2.5], [5.1, 4.9, 4.7]])
years = ['2018', '2019', '2020']
indicators = ['GDP Growth', 'Inflation', 'Unemployment']

# Creating a heatmap
sns.heatmap(data, annot=True, xticklabels=indicators, yticklabels=years,
    cmap="YlGnBu")
plt.title('Economic Indicators Heatmap')
plt.show()
# Expected output: Heatmap showing economic indicators over different years.
# cmap="YlGnBu"- Sequential Colormaps representing data that has a natural order
    (e.g., low to high values).
```

9. Box Plot

```
# Sample economic data
data = {
    'Country': ['USA', 'China', 'Japan', 'Germany', 'India'] * 5,
    'Year': [2016, 2017, 2018, 2019, 2020] * 5,
    'GDP Growth': [1.5, 2.3, 2.9, 2.3, -3.5, 6.7, 6.9, 6.6, 6.1, 2.3, 1.0, 0.3, 0.8,
        0.5, 1.8, 2.2, 2.5, 2.0, 2.1, 1.1, 7.1, 6.9, 6.5, 6.0, 5.5]
```

```
}
df = pd.DataFrame(data)

# Creating a box plot
sns.boxplot(x='Year', y='GDP Growth', data=df, palette="Set3")
plt.title('GDP Growth Distribution by Year')
plt.xlabel('Year')
plt.ylabel('GDP Growth (%)')
plt.show()
# Expected output: Box plot showing GDP growth distribution by year.
```

Back to Table of Contents

8 Scikit-learn

1. Importing Scikit-Learn Modules

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

2. Loading and Splitting Data

```
from sklearn.datasets import load_boston

# Load dataset
data = load_boston()
X = data.data  # feature matrix (independent variables, array/pd-dtf)
y = data.target # target vector (dependent variable, array/pd-dtf)

# Split dataset into training and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
    random_state=42)

# test_size: proportion of the dataset to include in the test split, and the
    remaining will be used for training (use interger for absolute number of test
    samples)

# random_state: Using a fixed number allows you to get the same split every time you
    run the code: ensuring reproducibility of the split.

# Extension: shuffle=True/False - shuffle the data before splitting
```

3. Standardizing Data

```
# Standardize features
# Create an instance of the StandardScaler
# The StandardScaler standardizes features by removing the mean and scaling to unit
    variance.

scaler = StandardScaler()

# Fit the StandardScaler to the training data (X_train) and transform it
# This calculates the mean and standard deviation for scaling using the training data
# and then scales the training data based on these calculations.

X_train_scaled = scaler.fit_transform(X_train)

# Transform the test data (X_test) using the same mean and standard deviation
    calculated from the training data
# This ensures that the test data is scaled in the same way as the training data.

X_test_scaled = scaler.transform(X_test)
```

4. Training a Linear Regression Model and Predict

```
# Train a linear regression model
model = LinearRegression()
model.fit(X_train_scaled, y_train)

# Make predictions on the test set
y_pred = model.predict(X_test_scaled)
```

```
print(y_prob)
# Expected output: Array of predicted probabilities for each class, e.g., [[0.7,
    0.3], [0.2, 0.8], ...]
```

5. Evaluating the Model

```
# Evaluate the model
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)
print(f'Mean Squared Error: {mse}')
print(f'R^2 Score: {r2}')
```

6. Cross-Validation

```
from sklearn.model_selection import cross_val_score

# Perform cross-validation
cv_scores = cross_val_score(model, X, y, cv=5)
print(f'Cross-Validation Scores: {cv_scores}')
print(f'Average CV Score: {cv_scores.mean()}')
```

7. Hyperparameter Tuning with Grid Search

```
from sklearn.model_selection import GridSearchCV
from sklearn.ensemble import RandomForestRegressor

# Define the model and parameter grid
rf = RandomForestRegressor(random_state=42)
param_grid = {
    'n_estimators': [50, 100, 200],
    'max_depth': [None, 10, 20, 30]
}

# Perform grid search
grid_search = GridSearchCV(rf, param_grid, cv=5, scoring='neg_mean_squared_error')
grid_search.fit(X_train, y_train)

# Best parameters and score
print(f'Best Parameters: {grid_search.best_params()}')
print(f'Best Score: {grid_search.best_score_}')
```

8. Feature Importance

```
# Feature importance from RandomForestRegressor
best_rf = grid_search.best_estimator_
importances = best_rf.feature_importances_
for feature, importance in zip(data.feature_names, importances):
    print(f'{feature}: {importance}')
```

9. Pipeline for Workflow Automation

```
from sklearn.pipeline import Pipeline

# Create a pipeline with scaling and model training
pipeline = Pipeline([
    ('scaler', StandardScaler()),
    ('regressor', LinearRegression())
```

```
])

# Train the pipeline
pipeline.fit(X_train, y_train)

# Make predictions and evaluate
y_pred_pipeline = pipeline.predict(X_test)
mse_pipeline = mean_squared_error(y_test, y_pred_pipeline)
r2_pipeline = r2_score(y_test, y_pred_pipeline)
print(f'Pipeline Mean Squared Error: {mse_pipeline}')
print(f'Pipeline R^2 Score: {r2_pipeline}')
```

Back to Table of Contents

9 Thank You Note

Thank you, ChatGPT 4o, for helping me writing this guide.