# Python Libraries for Statistical Analysis

| DocID | Author | Language | |
|---|---|---|---|
| PYSTATLIB | Kiran VVN | English | 2 |

## Contents

Python Libraries For statistical Analysis ........ 4

- Library: os ........ 4
  - Accessing Environment Variables ........ 4
  - File and Directory Manipulation ........ 4
  - Path Manipulation ........ 4
- Library: pandas ........ 4
  - Creating DataFrames ........ 4
  - Reading and Writing CSV Files ........ 4
  - DataFrame Operations ........ 4
- Library: sklearn.model_selection.train_test_split ........ 4
  - Splitting Data ........ 5
- Library: sklearn.linear_model.LinearRegression ........ 5
  - Fitting a Linear Regression Model ........ 5
- Library: sklearn.metrics.mean_squared_error ........ 5
  - Calculating MSE ........ 5
- Library: concurrent.futures ........ 5
  - Running Tasks in Threads ........ 5
- Library: matplotlib.pyplot & matplotlib.backends.backend_pdf.PdfPages ........ 5
  - Plotting a Graph ........ 5
  - Saving Plots to PDF ........ 6
- Library: pickle ........ 6
  - Serializing Data ........ 6
  - Deserializing Data ........ 6
- Library: argparse ........ 6
  - Command-line Argument Parsing ........ 6
- Library: psutil ........ 6
  - Getting CPU and Memory Usage ........ 6
- Library: threading ........ 6
  - Starting a New Thread ........ 6
- Library: time ........ 7
  - Sleeping for a Duration ........ 7

| DocID | Author | Language | |
|---|---|---|---|
| PYSTATLIB | Kiran VVN | English | 3 |

This document gives an overview of libraries used for stastistical analysis and how to apply them in Python with code samples.

| DocID | Author | Language | |
|---|---|---|---|
| PYSTATLIB | Kiran VVN | English | 4 |

# Python Libraries For statistical Analysis

## Library: os

The os module provides functions to interact with the operating system. It allows for tasks like file and directory manipulation, environment variable access, and process management.

### Accessing Environment Variables

```
import os
home_dir = os.getenv('HOME')
print(f'Home Directory: {home_dir}')
```

### File and Directory Manipulation

```
os.mkdir('test_dir')
files = os.listdir('.');
print(f'Files in current directory: {files}')
os.rmdir('test_dir')
```

### Path Manipulation

```
base_name = os.path.basename('/path/to/file.txt')
print(f'Base name: {base_name}')
```

## Library: pandas

Pandas is used for data manipulation and analysis. It offers data structures like Series and DataFrames to work with structured data.

### Creating DataFrames

```
import pandas as pd
data = {'Name': ['John', 'Anna', 'Peter'], 'Age': [29, 24, 35]}
df = pd.DataFrame(data)
print(df)
```

### Reading and Writing CSV Files

```
# Read a CSV file
df = pd.read_csv('data.csv')
# Write to a CSV file
df.to_csv('output.csv', index=False)
```

### DataFrame Operations

```
print(df['Name'])
print(df[df['Age'] > 25])
grouped = df.groupby('Age').count()
print(grouped)
```

## Library: sklearn.model_selection.train_test_split

train_test_split is used to split datasets into training and testing sets for machine learning.

| DocID | Author | Language | |
|---|---|---|---|
| PYSTATLIB | Kiran VVN | English | 5 |

### Splitting Data

```
from sklearn.model_selection import train_test_split
X = df[['Age']]
y = df['Name']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
print(f'Train set: {X_train}, Test set: {X_test}')
```

## Library: sklearn.linear_model.LinearRegression

Implements linear regression for predictive modeling.

### Fitting a Linear Regression Model

```
from sklearn.linear_model import LinearRegression
model = LinearRegression()
model.fit(X_train, y_train)
predictions = model.predict(X_test)
print(predictions)
```

## Library: sklearn.metrics.mean_squared_error

Provides metrics to evaluate model performance. mean_squared_error calculates the MSE of regression models.

### Calculating MSE

```
from sklearn.metrics import mean_squared_error
mse = mean_squared_error(y_test, predictions)
print(f'Mean Squared Error: {mse}')
```

## Library: concurrent.futures

Simplifies the execution of parallel tasks using thread or process pools.

### Running Tasks in Threads

```
import concurrent.futures
def square(n): return n * n
with concurrent.futures.ThreadPoolExecutor() as executor:
numbers = [1, 2, 3, 4]
results = executor.map(square, numbers)
print(list(results))
```

## Library: matplotlib.pyplot & matplotlib.backends.backend_pdf.PdfPages

matplotlib.pyplot is used for plotting graphs, and PdfPages allows saving multiple plots into a PDF file.

### Plotting a Graph

```
import matplotlib.pyplot as plt
plt.plot([1, 2, 3], [4, 5, 6])
plt.show()
```

### Saving Plots to PDF

```
from matplotlib.backends.backend_pdf import PdfPages
with PdfPages('output.pdf') as pdf:
    plt.plot([1, 2, 3], [4, 5, 6])
    pdf.savefig()
    plt.close()
```

## Library: pickle

Used to serialize and deserialize Python objects.

### Serializing Data

```
import pickle
data = {'Name': 'John', 'Age': 30}
with open('data.pkl', 'wb') as f:
    pickle.dump(data, f)
```

### Deserializing Data

```
with open('data.pkl', 'rb') as f:
    data = pickle.load(f)
print(data)
```

## Library: argparse

Used for parsing command-line arguments in Python programs.

### Command-line Argument Parsing

```
import argparse
parser = argparse.ArgumentParser(description='Process some integers.')
parser.add_argument('--num', type=int, help='An integer number')
args = parser.parse_args()
print(f'Number passed: {args.num}')
```

## Library: psutil

Provides an interface to retrieve information on system utilization (CPU, memory, etc.).

### Getting CPU and Memory Usage

```
import psutil
print(f'CPU Usage: {psutil.cpu_percent(interval=1)}%')
print(f'Memory Usage: {psutil.virtual_memory().percent}%')
```

## Library: threading

Provides support for multi-threading in Python.

### Starting a New Thread

```
import threading
def print_numbers():
    for i in range(5):
        print(i)
thread = threading.Thread(target=print_numbers)
thread.start()
thread.join()
```

## Library: time

Provides time-related functions like sleeping or getting the current time.

### Sleeping for a Duration

```
import time
print('Start')
time.sleep(2)
print('End after 2 seconds')
```