

# **LAPORAN Pengerjaan UAS Pemrograman Interpreter**



Dosen Pengampu:

Rahman Taufik, M.Kom

Muhammad Galih Ramaputra, S.Kom., M.T.I.

Tenggat Pengerjaan:

15 Desember 2024

Dikerjakan oleh:


Muhammad Ilham Akbar

**PROGRAM STUDI ILMU KOMPUTER FAKULTAS MATEMATIKA  
DAN ILMU PENGETAHUAN ALAM UNIVERSITAS LAMPUNG  
2024**

## 1. Penjelasan Fungsi

Fungsi-fungsi yang ada tersebar di setiap file python, ada 4 jenis file yaitu Data\_raw, display, Calculator\_py dan Testing. Dimana 4 file itu memiliki fungsi tersendiri.

### 1.1 Data\_raw



```
1 import csv
2 #Membaca file csv
3 def read_csv(filepath):
4     data = []
5     with open(filepath, mode='r', encoding='utf-8') as file:
6         reader = csv.reader(file)
7         for row in reader:
8             try:
9                 # Konversi data ke tipe yang sesuai
10                row[2] = float(row[2]) # Fee
11                row[3] = int(row[3]) # Duration
12                row[4] = float(row[4]) # Percentage Discount
13                if not (0 <= row[4] <= 1):
14                    raise ValueError(f"Invalid discount value: {row[4]}")
15                data.append(row)
16            except (ValueError, IndexError) as e:
17                print(f"Error processing row {row}: {e}")
18    return data
```

#### Fungsi `read_csv(filepath)`

Berguna untuk membaca data dari file csv, dengan parameter filepath sebagai lokasi file csv nya. Membuka file CSV dalam mode baca ('r') dengan encoding UTF-8. Membaca data menggunakan csv.reader, menghasilkan data berbentuk list per baris.

Untuk setiap baris (row):

- Baris ketiga (`row[2]`) diubah menjadi tipe `float` (menunjukkan fee).
- Baris keempat (`row[3]`) diubah menjadi tipe `int` (durasi).
- Baris kelima (`row[4]`) diubah menjadi tipe float (persentase diskon) dan diverifikasi agar nilainya berada di antara 0 dan 1.
- Jika data valid, baris akan ditambahkan ke dalam daftar data. Jika tidak valid (misalnya indeks tidak ada atau tipe data salah), akan menampilkan pesan kesalahan dan melewati baris tersebut.

Output:

- Mengembalikan daftar (list) berisi data dari file CSV dalam bentuk list of lists, di mana setiap list mewakili satu baris.

```

1 def write_csv(data, filepath, headers):
2     with open(filepath, mode='w', newline='', encoding='utf-8') as file:
3         writer = csv.writer(file)
4         writer.writerow(headers)
5         writer.writerows(data)

```

### Fungsi `write_csv(data,filepath,headers)`

Digunakan untuk menulis data ke file CSV. Dengan parameter:

- `data`: Daftar data yang akan ditulis ke file.
- `filepath`: Lokasi file CSV tujuan.
- `headers`: Baris header yang akan ditulis di bagian atas file.

Dimana cara kerjanya membuka file CSV dalam mode tulis ('w') dengan encoding UTF-8, lalu menulis header (headers) sebagai baris pertama file dan dilanjutkan dengan menulis seluruh data dari parameter data ke file.

```

1 def menghitung_total(row):
2     fee_duration = row[2] * row[3] # Fee * Duration
3     discount = fee_duration * (row[4] / 100) # Total Discount
4     if row[1] == "PI":
5         total = fee_duration - discount - (fee_duration * 0.02)
6         # Tambahan potongan 2% untuk kategori PI
7     else:
8         total = fee_duration - discount
9     return total

```

### Fungsi `menghitung_total(row)`

Fungsi ini digunakan untuk menghitung total nilai berdasarkan data pada satu baris. Dengan parameter:

- `row`: Satu baris data (list) yang berisi informasi seperti fee, durasi, kategori, dan diskon.

Fungsi di atas bekerja dengan mengalikan `fee (row[2])` dengan `duration (row[3])` untuk mendapatkan nilai dasar yang menghitung total diskon berdasarkan persentase diskon (`row[4] / 100`). Jika kategori (`row[1]`) adalah

"PI", maka diberi potongan tambahan sebesar 2% dari nilai dasar dan mengembalikan nilai total setelah diskon dan potongan.

```
1 def process_data(data):
2     for row in data:
3         total = menghitung_total(row)
4         row.append(total)
5     return data
```

### Fungsi `process_data(data)`

Fungsi ini digunakan untuk memproses seluruh data, menambahkan hasil perhitungan total ke setiap baris. Dengan parameter:

- `data`: Daftar data yang akan diproses (list of lists).

Untuk setiap baris dalam data menjalankan fungsi `menghitung_total` lalu menambahkan nilai total sebagai elemen baru di baris tersebut dan mengembalikan data yang sudah diproses.

## 1.2 display

```
1 from tabulate import tabulate
2
3 #Menampilkan data ke file main
4 def display_data(data, headers):
5     formatted_data = [
6         [f"{col:.2f}" if isinstance(col, float) else col for col in row]
7         for row in data
8     ]
9     print(tabulate(formatted_data, headers=headers, tablefmt="grid"))
```

### Fungsi `display_data(data, headers)`

Digunakan untuk menampilkan data dalam bentuk tabel yang rapi di terminal menggunakan library `tabulate`. Fungsi ini memformat setiap elemen dalam data (khususnya angka desimal) hingga dua angka di belakang koma sebelum menampilkannya dengan header dan tata letak tabel yang dipilih (dalam hal ini format "grid").

```
1 def display_statistics(statistics, label):
2     """
3     Menampilkan hasil statistik.
4     """
5     print(f"\nStatistics for '{label}':")
6     print(f"Mean: {statistics['mean']}")
7     print(f"Median: {statistics['median']}")
8     print(f"Mode: {statistics['mode']}")
```

### Fungsi `display_statistics(statistics, label)`

Digunakan untuk menampilkan hasil perhitungan statistik (rata-rata, median, dan modus) secara terorganisir dengan label tertentu. Fungsi ini mempermudah pengguna untuk melihat hasil data dan statistik secara langsung dengan format yang mudah dipahami.

## 1.3 Calculator\_py

```
1 from collections import Counter
2 class Statistics:
3     #Mencari rata-rata
4     @staticmethod
5     def mean(data):
6         return sum(data) / len(data)
```

### Fungsi `mean(data)`

Fungsi mean menghitung rata-rata (mean) dari sekumpulan data numerik. Fungsi ini merupakan metode statis, sehingga dapat diakses tanpa membuat instance kelas Statistics. Cara kerjanya adalah dengan menjumlahkan seluruh elemen dalam daftar data menggunakan `sum(data)` dan membagi hasilnya dengan jumlah elemen dalam daftar (`len(data)`). Fungsi ini mengembalikan nilai rata-rata dalam bentuk `float`.

```

1  #mencari median
2  @staticmethod
3  def median(data):
4      sorted_data = sorted(data)
5      n = len(sorted_data)
6      mid = n // 2
7      if n % 2 == 0:
8          return (sorted_data[mid - 1] + sorted_data[mid]) / 2
9      else:
10         return sorted_data[mid]

```

### Fungsi `median(data)`

Fungsi median digunakan untuk menghitung nilai tengah (median) dari sekumpulan data numerik. Data terlebih dahulu diurutkan menggunakan `sorted(data)`, lalu fungsi menentukan apakah jumlah data genap atau ganjil. Jika jumlahnya ganjil, median adalah elemen tengah; jika genap, median dihitung sebagai rata-rata dari dua elemen tengah. Hasil akhirnya dikembalikan dalam bentuk `float`.

```

1  #Mencari modus
2  @staticmethod
3  def mode(data):
4      freq = Counter(data)
5      max_count = max(freq.values())
6      modes = [k for k, v in freq.items() if v == max_count]
7      return modes[0] if len(modes) == 1 else modes

```

### Fungsi `mode(data)`

Fungsi mode menghitung nilai yang paling sering muncul (modus) dalam sekumpulan data. Fungsi menggunakan `Counter` dari library `collections` untuk menghitung frekuensi kemunculan setiap elemen. Kemudian, fungsi mencari nilai dengan frekuensi tertinggi. Jika hanya ada satu nilai modus, fungsi mengembalikannya. Jika ada beberapa nilai modus dengan frekuensi yang sama, fungsi mengembalikan daftar semua nilai modus tersebut.

## 1.4 Testing

```
1 import unittest
2 from Data_raw import read_csv, process_data, write_csv
3 from statistics import Statistics
4
5 class TestDataProcessing(unittest.TestCase):
6     #Menyiapkan unit test
7     def setUp(self):
8         self.test_file = "data.csv" # File CSV yang diunggah
9         self.processed_file = "processed_data.csv"
10        self.headers = ["Charity", "Categories", "Fee", "Duration", "Percentage Discount",
11        , "Total"]
12        self.data = [
13            ["Spark", "DS", 20000, 30, 0.05],
14            ["Hadoop", "DS", 25000, 40, 0.1],
15            ["Pandas", "PI", 30000, 35, 0.05]
16        ]
```

### Fungsi `setUp` pada `TestDataProcessing`

Fungsi `setUp` adalah metode yang dipanggil sebelum setiap unit test dijalankan untuk menginisialisasi data dan file yang akan digunakan. Dalam konteks ini, fungsi ini menentukan lokasi file CSV uji (`test_file` dan `processed_file`), mendefinisikan header untuk file yang diproses, dan membuat data contoh yang akan digunakan dalam pengujian. Data ini terdiri dari beberapa baris dengan atribut seperti nama organisasi amal, kategori, biaya, durasi, dan persentase diskon.

```
1 #Melakukan testing
2 def test_read_csv(self):
3     try:
4         data = read_csv(self.test_file)
5         self.assertIsInstance(data, list)
6         self.assertGreater(len(data), 0) # Data tidak boleh kosong
7     except Exception as e:
8         self.fail(f"read_csv raised an exception: {e}")
```

### Fungsi `test_read_csv`

Fungsi ini menguji apakah fungsi `read_csv` dapat membaca file CSV dengan benar. Fungsi memanggil `read_csv` dengan file contoh (`self.test_file`) dan

memverifikasi bahwa hasilnya adalah sebuah list dan tidak kosong. Jika `read_csv` mengembalikan hasil yang tidak valid atau terjadi pengecualian, tes akan gagal dan menampilkan pesan kesalahan.

```
1 #Menguji fungsi process data
2 def test_process_data(self):
3     try:
4         processed_data = process_data(self.data)
5         for row in processed_data:
6             self.assertEqual(len(row), len(self.headers))
7     except Exception as e:
8         self.fail(f"process_data raised an exception: {e}")
9
```

### Fungsi `test_process_data`

Fungsi ini menguji apakah fungsi `process_data` memproses data dengan benar. Data contoh yang disiapkan di `setUp` diproses menggunakan `process_data`, dan setiap baris hasilnya diperiksa apakah jumlah kolomnya sesuai dengan header yang didefinisikan. Jika terjadi kesalahan dalam proses atau struktur data hasil tidak sesuai, tes akan gagal.

```
1 #Menguji pembacaan data
2 def test_write_csv(self):
3     try:
4         processed_data = process_data(self.data)
5         write_csv(processed_data, self.processed_file, self.headers)
6     except Exception as e:
7         self.fail(f"write_csv raised an exception: {e}")
```

### Fungsi `test_write_csv`

Fungsi ini menguji apakah `write_csv` dapat menulis data yang telah diproses ke file CSV dengan benar. Data contoh diproses terlebih dahulu menggunakan `process_data`, lalu ditulis ke file menggunakan `write_csv`. Jika terjadi pengecualian selama proses penulisan, tes akan gagal dan mencatat pesan kesalahan.



```
1 class TestStatistics(unittest.TestCase):
2
3     #Memberikan sampel
4     def setUp(self):
5         self.data = [35, 20, 20, 30, 40]
```

### Fungsi `setUp` pada `TestStatistics`

Fungsi `setUp` di kelas `TestStatistics` digunakan untuk menyiapkan data numerik contoh yang akan digunakan dalam pengujian statistik. Data yang disiapkan adalah daftar angka `[35, 20, 20, 30, 40]`, yang nantinya akan diuji untuk fungsi rata-rata, median, dan modus.

```
1 #Menguji nilai mean
2 def test_mean(self):
3     stat = Statistics.mean(self.data)
4     self.assertEqual(stat, 24)
5
6 #Menguji nilai median atau nilai tengah
7 def test_median(self):
8     stat = Statistics.median(self.data)
9     self.assertEqual(stat, 20)
10
11 #Menguji nilai modus
12 def test_mode(self):
13     stat = Statistics.mode(self.data)
14     self.assertEqual(stat, 20)
```

### Fungsi `test_mean`

Fungsi ini menguji apakah `Statistics.mean` menghitung rata-rata dengan benar. Data contoh diproses menggunakan `Statistics.mean`, dan hasilnya diperiksa apakah sama dengan nilai yang diharapkan (24). Jika hasilnya tidak sesuai, tes akan gagal.

### Fungsi `test_median`

Fungsi ini menguji apakah `Statistics.median` menghitung median (nilai tengah) dengan benar. Data contoh diproses menggunakan `Statistics.median`,

dan hasilnya diperiksa apakah sama dengan nilai yang diharapkan (20). Jika ada kesalahan dalam perhitungan median, tes akan gagal.

## Fungsi `test_mode`

Fungsi ini menguji apakah `Statistics.mode` dapat menentukan nilai yang paling sering muncul (modus) dengan benar. Data contoh diproses menggunakan `Statistics.mode`, dan hasilnya diperiksa apakah sesuai dengan nilai yang diharapkan (20). Jika fungsi tidak mengembalikan nilai yang benar, tes akan gagal.

## 2. Class dan Method

### 2.1 Data\_raw

```
1 import csv
2 #Membaca file csv
3 def read_csv(filepath):
4     data = []
5     with open(filepath, mode='r', encoding='utf-8') as file:
6         reader = csv.reader(file)
7         for row in reader:
8             try:
9                 # Konversi data ke tipe yang sesuai
10                row[2] = float(row[2]) # Fee
11                row[3] = int(row[3]) # Duration
12                row[4] = float(row[4]) # Percentage Discount
13                if not (0 <= row[4] <= 1):
14                    raise ValueError(f"Invalid discount value: {row[4]}")
15                data.append(row)
16            except (ValueError, IndexError) as e:
17                print(f"Error processing row {row}: {e}")
18    return data
19 #Menulis data
20 def write_csv(data, filepath, headers):
21     with open(filepath, mode='w', newline='', encoding='utf-8') as file:
22         writer = csv.writer(file)
23         writer.writerow(headers)
24         writer.writerows(data)
25 #Menghitung total
26 def menghitung_total(row):
27     fee_duration = row[2] * row[3] # Fee * Duration
28     discount = fee_duration * (row[4] / 100) # Total Discount
29     if row[1] == "PI":
30         total = fee_duration - discount - (fee_duration * 0.02)
31         # Tambahan potongan 2% untuk kategori PI
32     else:
33         total = fee_duration - discount
34     return total
35 #Melakukan pemrosesan data
36 def process_data(data):
37     for row in data:
38         total = menghitung_total(row)
39         row.append(total)
40     return data
41
```

**Kelas DataProcessor** adalah kelas yang bertugas mempermudah pengelolaan data dalam file CSV.

- **Method `read_csv(filepath)`:** Metode ini membaca file CSV dari lokasi yang diberikan (`filepath`) dan mengembalikan data dalam bentuk list of lists. Data dalam file diproses dengan mengkonversi tipe data kolom tertentu: `Fee` menjadi `float`, `Duration` menjadi `int`, dan `Percentage Discount` menjadi `float`. Validasi dilakukan untuk memastikan diskon berada di antara 0 dan 1. Jika terdapat baris dengan format yang salah, baris tersebut dilewati dengan menampilkan pesan kesalahan.
- **Method `write_csv(data, filepath, headers)`:** Metode ini bertugas menulis data yang telah diproses ke file CSV baru. Header kolom ditulis sebagai baris pertama, diikuti oleh seluruh data dalam format tabular yang rapi.
- **Method `menghitung_total(row)`:** Metode ini digunakan untuk menghitung total biaya setelah memperhitungkan diskon dan potongan tambahan jika kategori adalah "PI". Metode ini mengalikan `Fee` dengan `Duration` untuk mendapatkan nilai awal, lalu mengurangi total diskon. Untuk kategori "PI", ada tambahan potongan 2% dari total nilai awal.
- **Method `process_data(data)`:** Metode ini memproses setiap baris data dengan menghitung total biaya menggunakan `menghitung_total` dan menambahkan hasil perhitungan tersebut sebagai kolom baru dalam setiap baris data. Data yang telah diproses kemudian dikembalikan.

## 2.2 display

```
1 from tabulate import tabulate
2
3 #Menampilkan data ke file main
4 def display_data(data, headers):
5     formatted_data = [
6         [f"{col:.2f}" if isinstance(col, float) else col for col in row]
7         for row in data
8     ]
9     print(tabulate(formatted_data, headers=headers, tablefmt="grid"))
10 #Menampilkan hasil yang didapat dari perhitungan
11 def display_statistics(statistics, label):
12     """
13     Menampilkan hasil statistik.
14     """
15     print(f"\nStatistics for '{label}':")
16     print(f"Mean: {statistics['mean']}")
17     print(f"Median: {statistics['median']}")
18     print(f"Mode: {statistics['mode']}")
```

- **Method `display_data(data, headers)`:** Metode ini menampilkan data dalam bentuk tabel menggunakan library `tabulate`. Data yang diterima diolah untuk memastikan bahwa semua nilai numerik dalam bentuk float ditampilkan hingga dua angka di belakang koma, sementara nilai non-numerik dibiarkan apa adanya. Tabel ini dilengkapi dengan header untuk memberikan penjelasan pada setiap kolom, dan format tabel dirancang dengan gaya "grid" untuk kemudahan pembacaan.
- **Method `display_statistics(statistics, label)`:** Metode ini menampilkan hasil perhitungan statistik berupa rata-rata (`mean`), nilai tengah (`median`), dan nilai yang paling sering muncul (`mode`) untuk dataset tertentu. Label diberikan untuk menjelaskan konteks statistik yang ditampilkan, seperti kategori data atau pengelompokannya. Metode ini membantu pengguna memahami ringkasan data dengan cepat dan terstruktur.

## 2.3 Calculator\_py

```

1  from collections import Counter
2  class Statistics:
3      #Mencari rata-rata
4      @staticmethod
5      def mean(data):
6          return sum(data) / len(data)
7      #mencari median
8      @staticmethod
9      def median(data):
10         sorted_data = sorted(data)
11         n = len(sorted_data)
12         mid = n // 2
13         if n % 2 == 0:
14             return (sorted_data[mid - 1] + sorted_data[mid]) / 2
15         else:
16             return sorted_data[mid]
17     #Mencari modus
18     @staticmethod
19     def mode(data):
20         freq = Counter(data)
21         max_count = max(freq.values())
22         modes = [k for k, v in freq.items() if v == max_count]
23         return modes[0] if len(modes) == 1 else modes

```

**Kelas Statistics** adalah kelas utilitas yang menyediakan metode statis untuk melakukan berbagai perhitungan statistik dasar, seperti rata-rata, median, dan modus. Karena semua metode bersifat statis, pengguna dapat langsung memanggil metode tersebut tanpa perlu membuat instance dari kelas ini. Penjelasan masing-masing metode:

- **Method `mean(data)`:** Metode ini menghitung rata-rata dari sekumpulan data numerik dengan menjumlahkan semua elemen dalam data menggunakan `sum(data)` dan membaginya dengan jumlah elemen (`len(data)`). Hasilnya adalah nilai rata-rata yang dikembalikan sebagai `float`.
- **Method `median(data)`:** Metode ini menghitung median atau nilai tengah dari sekumpulan data numerik. Data terlebih dahulu diurutkan dengan `sorted(data)`. Jika jumlah elemen genap, median dihitung sebagai rata-rata dari dua elemen tengah. Jika jumlah elemen ganjil, median adalah elemen di posisi tengah. Hasilnya dikembalikan dalam bentuk `float`.
- **Method `mode(data)`:** Metode ini menghitung modus, yaitu nilai yang paling sering muncul dalam data. Metode menggunakan `Counter` dari modul `collections` untuk menghitung frekuensi kemunculan setiap elemen. Jika hanya ada satu nilai yang muncul paling banyak, metode mengembalikannya. Jika ada beberapa nilai dengan frekuensi yang sama, metode mengembalikan daftar nilai tersebut.

## 2.4 Testing

```
1 import unittest
2 from Data_raw import read_csv, process_data, write_csv
3 from statistics import Statistics
4
5 class TestDataProcessing(unittest.TestCase):
6     #Menyiapkan unit test
7     def setUp(self):
8         self.test_file = "data.csv" # File CSV yang diunggah
9         self.processed_file = "processed_data.csv"
10        self.headers = ["Charity", "Categories", "Fee", "Duration", "Percentage Discount", "Total"]
11        self.data = [
12            ["Spark", "DS", 20000, 30, 0.05],
13            ["Hadoop", "DS", 25000, 40, 0.1],
14            ["Pandas", "PI", 30000, 35, 0.05]
15        ]
16        #Melakukan testing
17        def test_read_csv(self):
18            try:
19                data = read_csv(self.test_file)
20                self.assertIsInstance(data, list)
21                self.assertGreater(len(data), 0) # Data tidak boleh kosong
22            except Exception as e:
23                self.fail(f"read_csv raised an exception: {e}")
24
25        #Menguji fungsi process data
26        def test_process_data(self):
27            try:
28                processed_data = process_data(self.data)
29                for row in processed_data:
30                    self.assertEqual(len(row), len(self.headers))
31            except Exception as e:
32                self.fail(f"process_data raised an exception: {e}")
33
34        #Menguji pembacaan data
35        def test_write_csv(self):
36            try:
37                processed_data = process_data(self.data)
38                write_csv(processed_data, self.processed_file, self.headers)
39            except Exception as e:
40                self.fail(f"write_csv raised an exception: {e}")
41
42        class TestStatistics(unittest.TestCase):
43
44            #Memberikan sampel
45            def setUp(self):
46                self.data = [35, 20, 20, 30, 40]
47
48            #Menguji nilai mean
49            def test_mean(self):
50                stat = Statistics.mean(self.data)
51                self.assertEqual(stat, 24)
52
53            #Menguji nilai median atau nilai tengah
54            def test_median(self):
55                stat = Statistics.median(self.data)
56                self.assertEqual(stat, 20)
57
58            #Menguji nilai modus
59            def test_mode(self):
60                stat = Statistics.mode(self.data)
61                self.assertEqual(stat, 20)
62
63        if __name__ == "__main__":
64            unittest.main()
65
```

Kelas **TestDataProcessing** bertanggung jawab untuk menguji fungsi-fungsi yang terkait dengan pengolahan data CSV, yaitu **read\_csv**, **process\_data**, dan **write\_csv**.

- **Metode `setUp()`** digunakan untuk menyiapkan data uji, termasuk nama file CSV uji, file keluaran hasil proses, header kolom, dan beberapa data sampel.
- **Metode `test_read_csv()`** menguji fungsi `read_csv` untuk memastikan bahwa file CSV dapat dibaca dengan benar, menghasilkan tipe data list, dan tidak kosong. Jika terjadi kesalahan, pengujian akan gagal dengan pesan kesalahan.
- **Metode `test_process_data()`** menguji fungsi `process_data` untuk memastikan bahwa setiap baris data yang diproses memiliki jumlah kolom yang sesuai dengan header yang diharapkan.
- **Metode `test_write_csv()`** menguji fungsi `write_csv` untuk memastikan bahwa data yang diproses dapat ditulis kembali ke file CSV tanpa error.

**Kelas `TestStatistics`** bertanggung jawab untuk menguji metode statistik dalam kelas `Statistics`, yaitu mean, median, dan mode.

- **Metode `setUp()`** digunakan untuk menyiapkan data sampel berupa daftar angka yang akan digunakan dalam pengujian.
- **Metode `test_mean()`** menguji fungsi `mean` untuk memastikan bahwa hasil rata-rata dari data yang diberikan sesuai dengan nilai yang diharapkan.
- **Metode `test_median()`** menguji fungsi `median` untuk memastikan bahwa nilai tengah dari data yang diberikan dihitung dengan benar.
- **Metode `test_mode()`** menguji fungsi `mode` untuk memastikan bahwa nilai yang paling sering muncul dalam data dikembalikan dengan benar.

### 3. Error Handling

Pengujian dalam file `Testing` menggunakan error handling untuk memastikan bahwa setiap fungsi diuji dengan aman, dan jika terjadi kesalahan, pengujian tidak langsung gagal tanpa informasi yang jelas. Dengan memanfaatkan blok `try-except`, fungsi-fungsi seperti `read_csv`, `process_data`, dan `write_csv` diuji untuk memastikan mereka berjalan sesuai spesifikasi. Apabila terjadi kesalahan, seperti format data yang tidak sesuai, file tidak ditemukan, atau exception lainnya, pesan kesalahan ditangkap di blok `except`, dan pengujian dihentikan dengan memanggil `self.fail()`. Metode ini memberikan pesan kesalahan yang spesifik, seperti *"read\_csv raised an exception"* atau pesan terkait lainnya, sehingga memudahkan pengembang untuk mengidentifikasi penyebab masalah.

Selain itu, dalam pengujian statistik (`TestStatistics`), blok `try-except` tidak digunakan karena pengujian ini langsung memanfaatkan metode `assertEqual`

untuk memverifikasi hasil yang diharapkan. Pendekatan ini cocok karena perhitungan statistik biasanya tidak melibatkan operasi eksternal yang rentan terhadap error, seperti membaca file. Dengan kombinasi ini, pengujian memberikan cakupan yang baik, mendeteksi error dengan cepat, dan menyediakan informasi yang jelas untuk debugging jika ada kegagalan.

## 4. UnitTesting

Pengujian dalam kode di atas menggunakan unit testing untuk memverifikasi fungsi-fungsi secara independen, memastikan setiap bagian kode berjalan sesuai spesifikasi. Kelas `TestDataProcessing` menguji fungsionalitas pengolahan data terkait file CSV, seperti `read_csv`, `process_data`, dan `write_csv`. Unit test ini memastikan bahwa file CSV dapat dibaca, data diproses dengan benar sesuai jumlah kolom yang diharapkan, dan hasil akhirnya dapat ditulis kembali ke file tanpa error. Sementara itu, kelas `TestStatistics` menguji metode statistik seperti `mean`, `median`, dan `mode` untuk memastikan bahwa perhitungan rata-rata, nilai tengah, dan nilai yang paling sering muncul dari data menghasilkan hasil yang benar. Setiap metode pengujian menggunakan berbagai asersi, seperti `assertEqual`, `assertIsInstance`, dan `assertGreater`, untuk membandingkan output fungsi dengan hasil yang diharapkan. Dengan pendekatan ini, kode dapat diuji secara modular, memudahkan identifikasi kesalahan, serta memberikan jaminan bahwa setiap fungsi berjalan sesuai yang dirancang.

## 5. Module, Package, Library

### Modules

Modules adalah file Python yang berisi kumpulan fungsi, variabel, atau kelas yang dibuat untuk menyelesaikan tugas tertentu. Dengan menggunakan module, kode dapat dibagi menjadi bagian-bagian kecil yang lebih terorganisir. Misalnya, module `Data_raw` yang saya gunakan untuk memproses data, sedangkan module `Calculator_py` berisi fungsi statistik seperti menghitung rata-rata atau median yang dapat mengimpor dan menggunakan module tersebut di program lain untuk menghindari penulisan ulang kode yang sama.

### Packages

Packages adalah kumpulan beberapa module yang dikelompokkan ke dalam satu folder untuk memudahkan pengelolaan kode yang saling terkait. Package biasanya berisi file `__init__.py` sebagai penanda bahwa folder tersebut adalah package Python. Contoh nya folder yang dinamai `uas_pi` berisi module `Data_raw.py`, `Calculator_py.py`, dan `display.py` juga `Testing.py`, maka kita bisa



mengaksesnya dengan struktur impor seperti `from uas_pi.Data_raw import process_data`. Packages membantu menjaga struktur kode tetap rapi saat proyek menjadi lebih besar.

### **Libraries**

Libraries adalah kumpulan module atau package yang menyediakan alat siap pakai untuk berbagai kebutuhan, baik bawaan Python maupun dari pihak ketiga. Contohnya, library bawaan seperti csv digunakan untuk mengolah file CSV, dan `collections.Counter` membantu saya menghitung elemen dalam list. Ada juga library pihak ketiga seperti `tabulate` untuk menampilkan data dalam format tabel. Dengan libraries, pengembang tidak perlu membuat semua fungsi dari awal, hingga pengembangan nya menjadi lebih cepat dan efisien.

## **6. Motivasi**

Saya termotivasi untuk bekerja dalam bidang data sains ataupun data analisis karena saya suka pekerjaan yang membutuhkan struktur sehingga jelas dalam tahapan pengerjaan nya dibutuhkan proses yang runtut dan flexibel. Tidak hanya itu, bekerja dalam bidang database membuat saya memiliki tahapan yang jelas untuk memecahkan suatu masalah atau menganalisa suatu masalah, kedepannya saya ingin membuat project mengenai sistem pakar yang dapat membantu dalam pengambilan keputusan dari sampel suatu data.