

PDPL Source Code

Berikut adalah penjelasan dari source code untuk tugas pdpl ini. Kode sudah menyesuaikan dengan tugas pdpl ini sehingga 95% kode yang digunakan sudah menerapkan design pattern. Untuk peletakan design pattern pada kode yang digunakan yaitu dari awal baris kode sampai akhir baris kode.

Anggota Kelompok :

- Ihsanul Fikri Abiyyu
- Muhammad Azhar Rasyad
- Muhammad Adil Nasrulhaq
- Triyas Tono

Design Pattern yang digunakan pada tugas ini yaitu :

1. ***Creational*** : *Singleton, Abstract Factory Method, Builder*
2. ***Structural*** : *Proxy*
3. ***Behavioral*** : *Chain of Responsibility*
4. ***Architectural*** : *MVC*

Kodingan dibuat menggunakan IDE jupyter notebook. Karena itu file yang digunakan berformat .ipynb dan hanya bisa dibuka melalui jupyter notebook atau google code. File kodingan terdapat pada source code yang sudah dikirim bersamaan dengan file ini.

Untuk google code bisa diakses disini :

<https://colab.research.google.com/drive/1iPEX1qZehk6eVkcCc34fR6sOepS1Tsxo-?usp=sharing>

Kita menyediakan juga untuk file python biasa yang berformat .py dan bisa diakses di text editor apa saja, tetapi kita lebih merekomendasikan menggunakan file berformat .ipynb yang hanya bisa diakses melalui jupyter notebook atau google code

- Abstract Factory Method & Singleton

Abstract factory digunakan untuk menampilkan data dan menghapus data yang digunakan pada tugas link n match kita

Abstract Factory method dan inisiasi data yang digunakan

```
In [2]: class abstractData:  
    def show_data(self): pass  
  
    def delete_data(self) :pass  
  
In [3]: class earthquake_data(abstractData):  
    __shared_state = dict()  
  
    # constructor method  
    def __init__(self, tableData):  
        self.__dict__ = self.__shared_state  
        self.tableData = tableData  
  
    def show_data(self):  
        return self.tableData  
  
    def delete_data(self):  
        self.tableData = {}  
        print("Data is gone")
```

Implementasi pada abstract factory diatas adalah bagaimana kita mengimplementasikan abstract class yang bertugas untuk show_data dan delete_data pada spesifik class yang masih berhubungan dengan ke 2 factory method tersebut. Pada contoh diatas kita hanya mengimplementasikan pada class earthquake_data dikarenakan kita hanya menggunakan 1 data saja.

Class earthquake_data sendiri mengimplementasikan singleton, karena data yang digunakan hanya 1 data sehingga instansiasi class pada objek, hanya menjadi 1 objek saja yang nantinya bisa diakses secara global

```
In [3]: class earthquake_data(abstractData):  
    __shared_state = dict()  
  
    # constructor method  
    def __init__(self, tableData):  
        self.__dict__ = self.__shared_state  
        self.tableData = tableData  
  
    def show_data(self):  
        return self.tableData  
  
    def delete_data(self):  
        self.tableData = {}  
        print("Data is gone")  
  
In [4]: read_data = pd.read_csv('./all_month.csv')  
earthquake = earthquake_data(read_data)  
  
In [5]: # earthquake2.delete_data()  
earthquake_show = earthquake.show_data()  
earthquake_show
```

Out[5]:

	time	latitude	longitude	depth	mag	magType	nst	gap	dmin	rms	...	updated	place	type	horizontalErr
0	2020-05-27T09:53:17.940Z	38.236700	-117.8493	0.00	2.00	ml	19.0	66.67	0.07500	0.31	...	2020-05-27T10:10:42.578Z	57km WNW of Tonopah, Nevada	earthquake	Ni
1	2020-05-27T09:52:42.630Z	38.251800	-117.8007	0.00	1.30	ml	13.0	153.58	0.06500	0.23	...	2020-05-27T10:10:33.427Z	54km WNW of Tonopah, Nevada	earthquake	Ni
2	2020-05-27T09:43:43.860Z	17.951300	-66.8401	11.00	1.90	md	12.0	209.00	0.04580	0.13	...	2020-05-27T10:02:48.425Z	5km SSW of Indios, Puerto Rico	earthquake	O

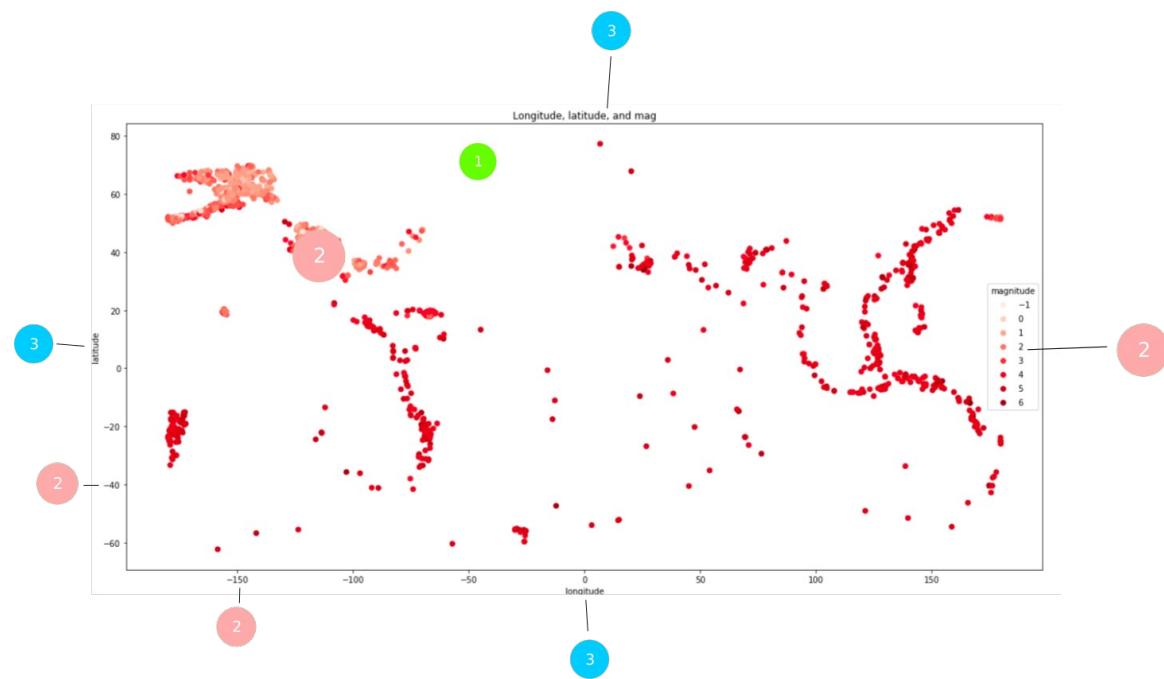
Pada Contoh diatas, class earthquake_data di instansiasi pada variable earhquake, instansiasi sendiri juga melibatkan data gempa dalam 1 bulan. Lalu setelah di instansiasi, variable earthquake memanggil method show_data yang digunakan untuk menampilkan data, dimana hasil dari menampilkan data disimpan pada variable earthquake_show. Nantinya variable earthquake_show akan digunakan pada pattern yang akan dijelaskan setelah ini.

- Builder Pattern

Builder pattern untuk membuat plotting data

```
In [6]: class plotting:  
    def make_plot(self): pass  
  
    def add_plot_data(self): pass  
  
    def set_plot(self): pass  
  
    def show_plot(self): pass
```

Builder kita gunakan untuk membuat ploting grafik dari data yang telah dimasukan. Tahapan pada pembuatan plotting nya adalah :



1. make_plot = digunakan untuk membuat kanvas tempat grafik akan diterapkan
2. add_plot_data = digunakan untuk menaruh data apa saja yang akan ditaruh pada kanvas, dan jenis grafik apa yang ingin dibuat
3. set_plot = digunakan untuk membuat labeling pada grafik
4. show_plot = digunakan untuk menampilkan grafik

Untuk class yang menggunakan builder sebagai bahan rancangan pembuatan grafik adalah scatter, plot, histogram. Masing masing class akan membuat grafik sesuai plotting data yang diinginkan. Tetapi untuk perancangan pembuatan nya mengikuti tahapan yang ada pada abstract class builder

1. Scatter plot

```
In [8]: class scatter_plot(plotting):
    def __init__(self, data):
        self.data = data
        self.make_plot()
        self.add_plot_data()
        self.set_data()

    def make_plot(self):
        self.fig, self.ax = plt.subplots(figsize=(20,10))

    def add_plot_data(self):
        self.scatter = self.ax.scatter(x=self.data["longitude"],
                                       y=self.data["latitude"],
                                       c=self.data["mag"],
                                       cmap="Reds")

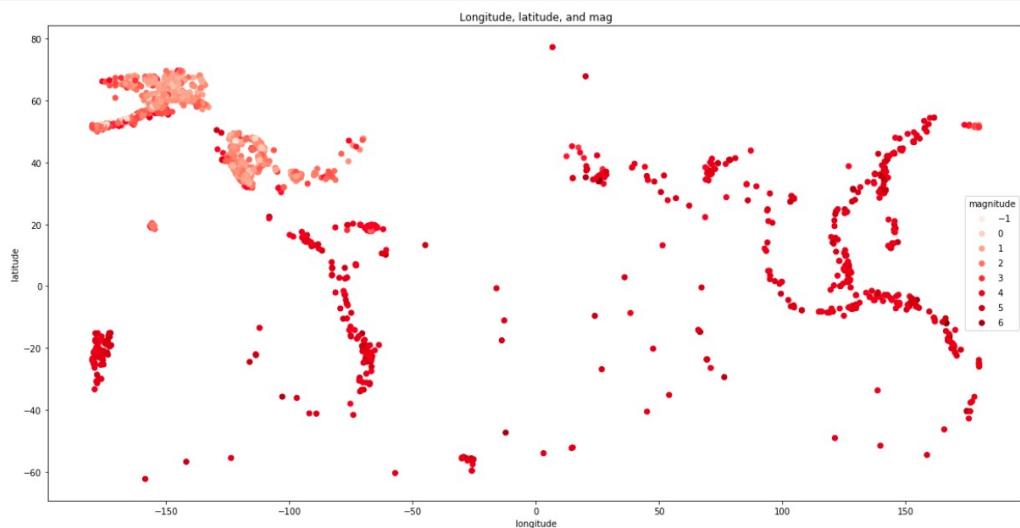
    def set_data(self):
        self.ax.set(title="Longitude, latitude, and mag",
                   xlabel="longitude",
                   ylabel="latitude");

        self.ax.legend(*self.scatter.legend_elements(), title="magnitude");

    def show_plot(self):
        return self.fig
```

Digunakan untuk membuat grafik dengan plotting berbentuk scatter. Data yang digunakan untuk pembuatan grafik berasal dari variable earthquake_show yang berasal dari pattern sebelumnya. Grafik yang dihasilkan dari class scatter_plot adalah sebagai berikut

```
In [9]: scatter = scatter_plot(earthquake_show)
```



2. Histogram Plot

```
In [11]: class histogram_plot(plotting):
    def __init__(self, data):
        self.data = data
        self.make_plot()
        self.add_plot_data()
        self.set_data()
        self.show_plot()

    def make_plot(self):
        self.fig, self.ax = plt.subplots(figsize=(15,8))

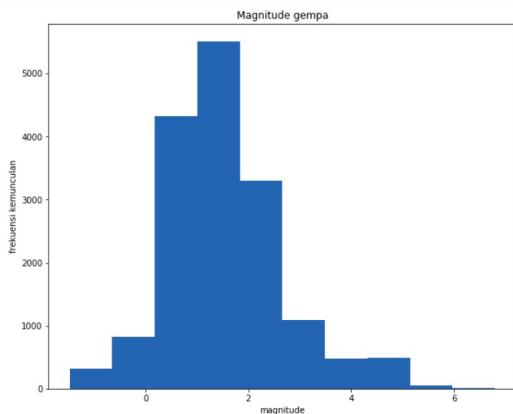
    def add_plot_data(self):
        self.hist = self.ax.hist(self.data["mag"])

    def set_data(self):
        self.ax.set(title="Magnitude gempa",
                   xlabel="magnitude",
                   ylabel="frekuensi kemunculan");

    def show_plot(self):
        return self.fig
```

Sama seperti scatter plot, histogram plot yang dibuat disini berdasarkan abstract class builder bernama plotting. Tahapan yang digunakan untuk membuat histogram plot sesuai dengan class builder nya, yaitu membuat kanvas dengan menggunakan method make_plot, lalu menambahkan data dan memilih grafik yang diinginkan dengan menggunakan method add_plot_data, Setelah itu menambahkan labeling pada grafik dengan method set_data. Hasil dari keseluruhan proses dapat ditampilkan menjadi sebuah grafik sesuai plotting yang sudah dibuat dan ditampilkan dengan method show_plot.

```
In [11]: histogram = histogram_plot(earthquake_show)
/home/ihsanul/Documents/ml-ds-course/env/lib/python3.7/site-packages/numpy/lib/histograms.py:839: RuntimeWarning:
invalid value encountered in greater_equal
    keep = (tmp_a >= first_edge)
/home/ihsanul/Documents/ml-ds-course/env/lib/python3.7/site-packages/numpy/lib/histograms.py:840: RuntimeWarning:
invalid value encountered in less_equal
    keep &= (tmp_a <= last_edge)
```



Untuk error pada grafik diatas, hanya peringatan untuk masalah data. Tetapi secara keseluruhan plot bekerja membuat grafik sesuai yang diinginkan. Data yang digunakan pada plot ini adalah data yang tersimpan pada variable earthquake_show yang berasal dari pattern sebelumnya.

3. Plot

```
In [13]: class plotting:
    def __init__(self, data):
        self.data = data
        self.make_plot()
        self.add_plot_data()
        self.set_data()

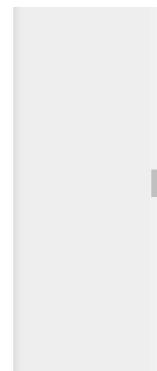
    def make_plot(self):
        self.fig, self.ax = plt.subplots(figsize=(30,6))

    def add_plot_data(self):
        self.plot = self.ax.plot(self.data["time"], self.data["mag"],)

    def set_data(self):
        self.ax.set_title("Earthquake Time and Magnitude",
                          xlabel="Time",
                          ylabel="Magnitude");

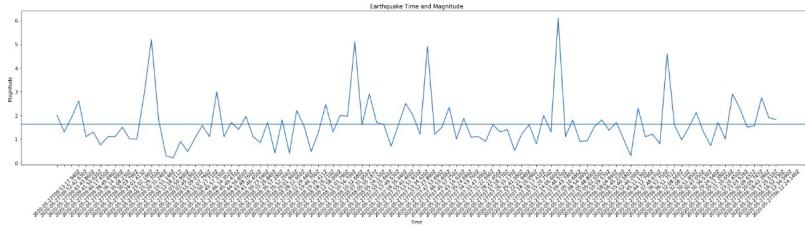
    plt.draw()
    self.ax.set_xticklabels(self.ax.get_xticklabels(), rotation=45);
    self.ax.axhline(earthquake_100["mag"].mean());

    def show_plot(self):
        return self.fig
```



Untuk plotting pada grafik kali ini kita menggunakan plot dasar, yaitu berupa garis. Kita masih mengimplementasikan class builder yaitu class plotting sebagai pembuatan plot pada grafik.

```
In [14]: plot100 = plot(earthquake_100)
```



4. Scatter plot 2

```
In [15]: class scatter_plot2(plotting):
    def __init__(self, data):
        self.data = data
        self.make_plot()
        self.add_plot_data()
        self.set_data()

    def make_plot(self):
        self.fig, self.ax = plt.subplots(figsize=(30,6))

    def add_plot_data(self):
        self.scatter = self.ax.scatter(x=self.data["place"],
                                      y=self.data["depth"],
                                      c=self.data["mag"],
                                      cmap="winter")

    def set_data(self):
        self.ax.set_title("Earthquake and Depth",
                          xlabel="Place",
                          ylabel="Depth");

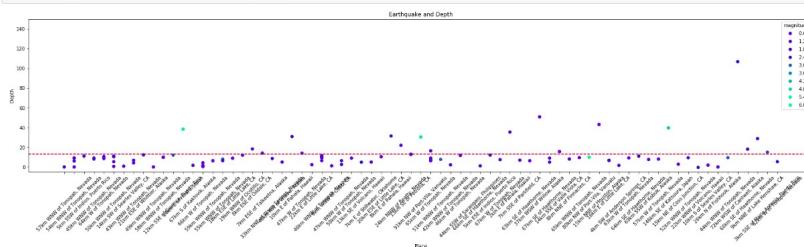
    plt.draw()
    self.ax.set_xticklabels(self.ax.get_xticklabels(), rotation=45)
    self.ax.set_ylim([-5,150])
    self.ax.legend(self.scatter.legend_elements(), title="magnitude");
    self.ax.axhline(earthquake_100["depth"].mean(),
                   linestyle="--",
                   color="r");

    def show_plot(self):
        return self.fig
```



Untuk scatter plot 2 sama saja dengan scatter plot pertama, class ini dibuat untuk perbedaan data yang dipakai, tetapi dasarnya masih sama dengan scatter plot pertama.

```
In [16]: scatter2 = scatter_plot2(earthquake_100)
```



- Chain of responsibility

Kita menggunakan design pattern ini untuk membuat pemilihan grafik yang digunakan. Jadi semisal user menginginkan grafik berbentuk scatter, maka apa yang diinginkan user akan di check ke masing masing handler dan jika handler nya sesuai, maka handler tersebut akan memproses permintaan user.

Contoh nya, semisal user meminta scatter, maka akan dicheck ke handler pertama, apakah scatter akan ditangani handler tersebut, jika iya akan ditangan handler tersebut. Jika tidak akan diteruskan ke handler berikutnya. Dan begitu seterusnya sampai handler terakhir.

Chain of responsibility, digunakan untuk mencari grafik yang sesuai dengan permintaan user

```
In [17]: class AbstractHandler(object):
    """Parent class of all concrete handlers"""
    def __init__(self, nxt):
        """change or increase the local variable using nxt"""
        self._nxt = nxt
    def handle(self, request):
        """It calls the processRequest through given request"""
        handled = self.processRequest(request)
        """case when it is not handled"""
        if not handled:
            self._nxt.handle(request)
    def processRequest(self, request):
        """throws a NotImplementedError"""
        raise NotImplementedError('First implement it !')
```

Pada class abstract diatas, method handle digunakan untuk mengecek apakah sudah sesuai handler yang akan dipakai dengan permintaan user. Untuk mengecek nya, method handle memanggil method berikutnya yaitu processRequest, dimana jika handler nya tepat sesuai dengan yang diinginkan user, maka akan ditangani dengan handler tersebut, tetapi jika tidak, maka method handle akan mengarahkan ke handler berikutnya, dan dilakukan langkah yang sama.

Terdapat 4 handler yang digunakan pada chain of responsibility, yaitu :

```
class FirstConcreteHandler(AbstractHandler):
    """Concrete Handler # 1: Child class of AbstractHandler"""
    def processRequest(self, request):
        if request == 'scatter':
            return scatter_plot(earthquake_show)

class SecondConcreteHandler(AbstractHandler):
    """Concrete Handler # 2: Child class of AbstractHandler"""
    def processRequest(self, request):
        if request == 'histogram':
            return histogram_plot(earthquake_show)

class ThirdConcreteHandler(AbstractHandler):
    """Concrete Handler # 3: Child class of AbstractHandler"""
    def processRequest(self, request):
        if request == 'plot':
            return plot(earthquake_100)

class FourthConcreteHandler(AbstractHandler):
    """Concrete Handler # 3: Child class of AbstractHandler"""
    def processRequest(self, request):
        if request == 'scatter2':
            return scatter_plot2(earthquake_100)
```

Handler pertama adalah class FirstConcreteHandler, ini akan menghandle jika permintaan adalah scatter. Handler kedua adalah class SecondConcreteHandler, ini digunakan jika permintaan adalah histogram. Handler ketiga adalah class ThirdConcreteHandler, digunakan jika permintaan hanya plot dasar. Handler keempat adalah FourthConcreteHandler, sama seperti handler pertama, tetapi untuk keperluan link n match ada pembeda data dari yang digunakan dihandler pertama. Digunakan jika permintaan scatter2.

Untuk cara kerja handler nya, pertama permintaan akan masuk ke handler pertama, lalu jika permintaan tidak cocok akan diarahkan kehandler berikutnya, begitu seterus nya sampai handler ke 4

Lalu, jika dari handler 1 sampai 4 tidak bisa menangani nya, maka akan dipanggil default handler

Contoh penerapan dari class class diatas adalah sebagai berikut

```
class User:

    """User Class"""

    def __init__(self):
        """Provides the sequence of handles for the users"""

        initial = None

        self.handler = FirstConcreteHandler(SecondConcreteHandler(ThirdConcreteHandler(FourthConcreteHandler(DefaultConcreteHandler()))))

    def agent(self, user_request):
        """Iterates over each request and sends them to specific handles"""

    #     for request in user_request:
    #         self.handler.handle(user_request)

    """main method"""

if __name__ == "__main__":
    """Create a client object"""
    user = User()

    """Create requests to be processed"""

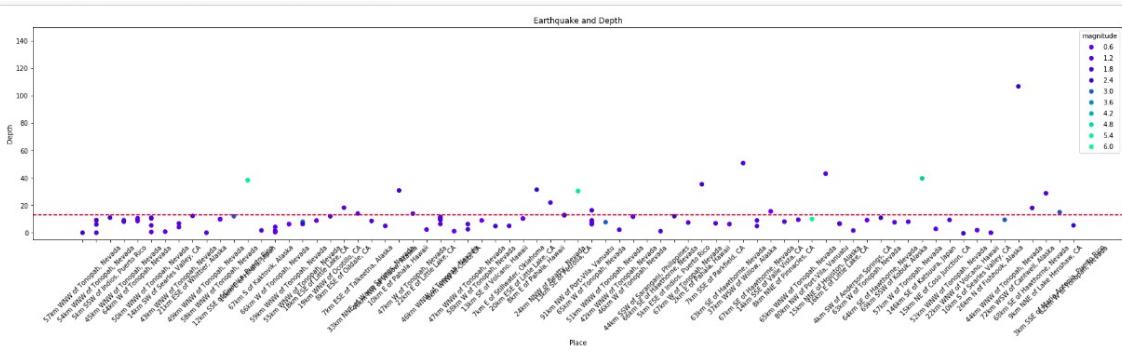
    string = "plot"
    #     requests = list(string)

    """Send the requests one by one, to handlers as per the sequence of handlers defined in the Client class"""
    user.agent(string)
```

Pertama dibuat class User, dimana constructornya menginisiasi attribute handler yang nantinya digunakan untuk menyimpan handler handler yang akan digunakan.

Pada method agent, attribute handler itu dipanggil, dengan tambahan parameter yang berisi permintaan user. Handler yang disimpan pada attribute handler, akan menanganani parameter tersebut sesuai dengan penjelasan sebelumnya.

Pada contoh diatas, class User diinstansiasi pada variable user, variable user memanggil method agent, dengan parameter plot, lalu parameter plot itu akan diberikan pada handler pertama, jika tidak cocok akan dialihkan ke handler berikutnya, begitupun seterunya, sampai ditemukan handler yang cocok. Handler yang cocok adalah plot, maka akan ditampilkan plotting grafik berbentuk plot dasar sebagai berikut :



- Proxy

Digunakan agar user tidak langsung mengakses class pada pattern pattern yang dibuat sebelum nya, melainkan melalui proxy. Jadi semisal user ingin menambahkan data, menampilkan data, atau membuat grafik tidak mengakses langsung class yang berhubungan dengan class tersebut. Tetapi melalui proxy, yang nantinya akan memanggil class yang dibutuhkan sesuai keinginan user

Proxy agar user tidak mengakses langsung class yang akan digunakan

```
In [18]: class earthquake_proxy:  
    '''Relatively less resource-intensive proxy acting as middleman.  
    Instantiates a College object only if there is no fee due.'''  
  
    def __init__(self):  
        self.data = None  
  
    def add_data(self, data):  
        self.earthquake = earthquake_data(data)  
  
    def show_data(self):  
        return self.earthquake.show_data()  
  
    def delete_data(self):  
        self.earthquake.delete_data()  
  
    def show_plot(self, plot):  
        self.user = User()  
        self.user.agent(plot)
```

Pada class earthquake_proxy, terdapat 3 method yang mempunyai fungsi masing masing. Jika user menginginkan untuk menambahkan data, maka akan dipanggil method add_data. Method add_data sendiri akan memanggil class earthquake_data yang bertanggung jawab untuk pengelolaan data menggunakan pattern yang digunakan sebelumnya.

Untuk method show_data digunakan untuk menampilkan data yang nantinya akan memanggil class earthquake_data dan menggunakan method show_data yang terdapat pada class earthquake_data untuk menampilkan data.

Pada method delete_data fungsinya untuk menghapus data, dan menggunakan method delete_data pada class earthquake_data.

Sedangkan untuk method show_plot, digunakan untuk membuat grafik. Method show_plot sendiri memanggil class User yang di instansiasi pada attribute user, dan proses selanjutnya seperti pada penjelasan pattern chain of responsibility.

Contoh penerapan proxy adalah sebagai berikut :

```
In [19]: # Instantiate the Proxy
proxy = earthquake_proxy()

In [29]: #Add Data
data = pd.read_csv('../all_month.csv')
proxy.add_data(data)

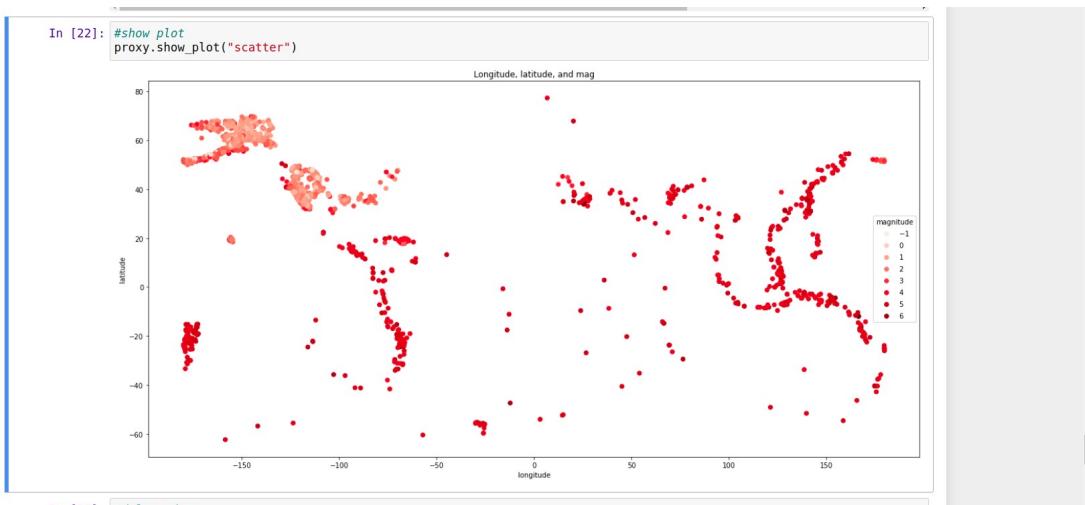
In [30]: #Show Data
proxy.show_data().head()

Out[30]:
   time      latitude  longitude  depth  mag  magType  nst  gap  dmin  rms ...  updated  place  type  horizontalError  depth
0 2020-05-27T09:53:17.940Z  38.2367 -117.8493    0.0  2.0       ml  19.0  66.67  0.0750  0.31 ... 2020-05-27T10:10:42.576Z  57km WNW of Tonopah, Nevada  earthquake  NaN
1 2020-05-27T09:52:42.630Z  38.2518 -117.8007    0.0  1.3       ml  13.0  153.58  0.0650  0.23 ... 2020-05-27T10:10:33.427Z  54km WNW of Tonopah, Nevada  earthquake  NaN
2 2020-05-27T09:43:43.860Z  17.9513 -66.8401   11.0  1.9       md  12.0  209.00  0.0458  0.13 ... 2020-05-27T10:02:48.425Z  5km SSW of Indios, Puerto Rico  earthquake  0.65
3 2020-05-27T09:41:50.620Z  38.1997 -117.7217    9.2  2.6       ml  34.0  69.76  0.0420  0.09 ... 2020-05-27T09:45:18.701Z  45km WNW of Tonopah, Nevada  earthquake  NaN
4 2020-05-27T09:40:17.650Z  38.1357 -117.9625    8.6  1.1       ml  14.0  100.75  0.0070  0.05 ... 2020-05-27T09:43:36.376Z  64km W of Tonopah, Nevada  earthquake  NaN

5 rows × 22 columns
```

1. Pertama user menginstasiasi class earthquake_proxy pada variable proxy
2. User dapat menambahkan data dengan memanggil method add_data
3. User juga dapat menampilkan data dengan method show_data
4. User dapat menampilkan plot dengan method show_plot dengan parameter sesuai grafik yang diinginkan nya.

Contoh pada gambar dibawah :



5. User dapat menghapus data dengan method delete_data. Lalu user juga dapat menampilkan data kembali dengan method show_data, karena data sudah dihapus, maka tidak ada data yang ditampilkan. Lalu user menambahkan data kembali dengan method add_data

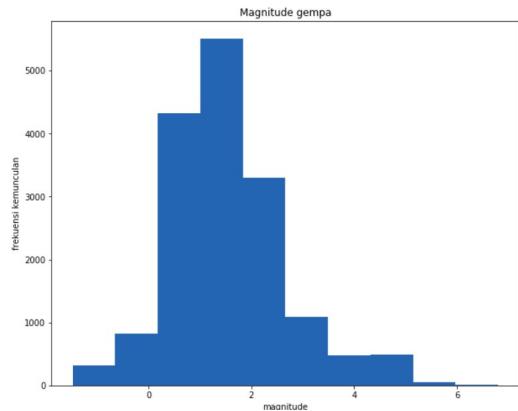
```
#delete data
proxy.delete_data()
proxy.show_data()

Data is gone
{}

proxy.add_data = pd.read_csv("./all_month.csv")
```

6. User kembali memanggil method show_plot tetapi kali ini dengan parameter histogram.

```
In [32]: proxy.show_plot("histogram")
/home/ihsanul/Documents/ml-ds-course/env/lib/python3.7/site-packages/numpy/lib/histograms.py:839: RuntimeWarning:
invalid value encountered in greater_equal
    keep = (tmp_a >= first_edge)
/home/ihsanul/Documents/ml-ds-course/env/lib/python3.7/site-packages/numpy/lib/histograms.py:840: RuntimeWarning:
invalid value encountered in less_equal
    keep &= (tmp_a <= last_edge)
```



- Model View Controller (MVC)

Pembuatan pattern pattern diatas sudah mengadaptasi arsitektur model view controller. Model digunakan pada pattern abstract factory method dan singleton, lalu view pada pattern builder dan controller pada pattern proxy.

Untuk mengakses model, maka akan dipanggil class earthquake_data, untuk mengakses view menggunakan builder dan untuk pemilihan grafik apa yang digunakan menggunakan chain of responsibility

Lalu untuk pengaksesan model dan view, user tidak langsung mengakses class yang telah disebutkan, tetapi menggunakan controller yang terdapat proxy

- Model

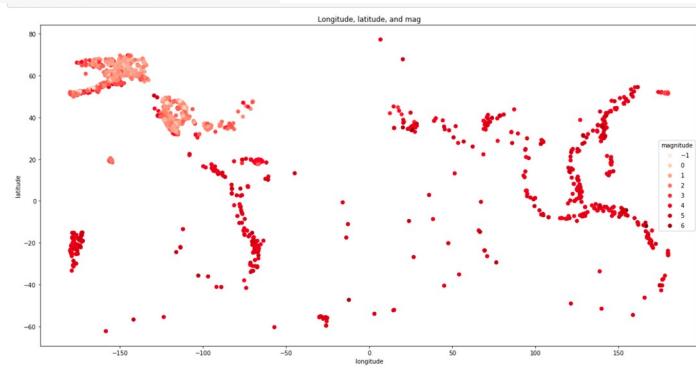
Abstract Factory method dan inisiasi data yang digunakan

```
In [2]: class abstractData:  
    def show_data(self): pass  
  
    def delete_data(self) :pass  
  
In [3]: class earthquake_data(abstractData):  
    _shared_state = dict()  
  
    # constructor method  
    def __init__(self, tableData):  
  
        self.__dict__ = self._shared_state  
        self.tableData = tableData  
  
    def show_data(self):  
        return self.tableData  
  
    def delete_data(self):  
        self.tableData = {}  
        print("Data is gone")
```

- View

```
In [11]: class histogram_plot(plotting):  
    def __init__(self, data):  
        self.data = data  
        self.make_plot()  
        self.add_plot_data()  
        self.set_data()  
        self.show_plot()  
  
    def make_plot(self):  
        self.fig, self.ax = plt.subplots(figsize=(15,8))  
  
    def add_plot_data(self):  
        self.hist = self.ax.hist(self.data["mag"])[0]  
  
    def set_data(self):  
        self.ax.set(title="Magnitude gempa",  
                   xlabel="magnitude",  
                   ylabel="frekuensi kemunculan");  
  
    def show_plot(self):  
        return self.fig
```

```
In [8]: class scatter_plot(plotting):  
    def __init__(self, data):  
        self.data = data  
        self.make_plot()  
        self.add_plot_data()  
        self.set_data()  
  
    def make_plot(self):  
        self.fig, self.ax = plt.subplots(figsize=(20,10))  
  
    def add_plot_data(self):  
        self.scatter = self.ax.scatter(x=self.data["longitude"],  
                                      y=self.data["latitude"],  
                                      c=self.data["mag"],  
                                      cmap="Reds")  
  
    def set_data(self):  
        self.ax.set(title="Longitude, latitude, and mag",  
                   xlabel="longitude",  
                   ylabel="latitude");  
  
    self.ax.legend(*self.scatter.legend_elements(), title="magnitude");  
  
    def show_plot(self):  
        return self.fig
```



- Controller

```
In [18]: class earthquake_proxy:  
    '''Relatively less resource-intensive proxy acting as middleman.  
    Instantiates a College object only if there is no fee due.'''  
  
    def __init__(self):  
        self.data = None  
  
    def add_data(self, data):  
        self.earthquake = earthquake_data(data)  
  
    def show_data(self):  
        return self.earthquake.show_data()  
  
    def delete_data(self):  
        self.earthquake.delete_data()  
  
    def show_plot(self, plot):  
        self.user = User()  
        self.user.agent(plot)
```

Referensi Kodingan

<https://refactoring.guru/design-patterns/abstract-factory>

<https://www.geeksforgeeks.org/abstract-factory-method-python-design-patterns/>

<https://refactoring.guru/design-patterns/builder>

<https://www.geeksforgeeks.org/builder-method-python-design-patterns/>

<https://www.geeksforgeeks.org/chain-of-responsibility-python-design-patterns/>

<https://refactoring.guru/design-patterns/chain-of-responsibility>

<https://refactoring.guru/design-patterns/proxy>

<https://www.geeksforgeeks.org/proxy-method-python-design-patterns/>