

Modul Pelatihan Python

DAFTAR ISI

DAFTAR ISI	I
BAB I PENDAHULUAN	1
1.1 Apa itu Python ?	1
1.2 Kelebihan dan Kekurangan Python	1
Kelebihan	1
Kekurangan	2
1.3 Menginstall Python Dalam Sistem Operasi Windows	2
1.4 Jupyter Notebook	3
Instalasi Jupyter Notebook	4
Menjalankan Jupyter Notebook	5
Membuat Notebook	5
Cell	6
Toolbar	6
1.5 Anaconda	7
Instalasi Anaconda	7
1.6 Tipe data	8
Bilangan : Integer dan Float	9
String	10
1.7 Koleksi pada Python	11
List	12
Dictionary	14
1.8 Logika dan Control Flow	16
Operator Boolean dan Operator Logika	16
Kalimat Kondisional	17
Pengulangan While	18
Pengulangan For	19
1.9 Module	20
BAB II NumPy, Scipy, dan Pandas	22
2.1 Numerical Python – NumPy	22
Install NumPy	22
Matriks dan Vektor	23
N-Dimensional array	24
Matriks N-dimensional	25
Statistika Deskriptif	28
2.2 Scientific Python - SciPy	29

Aljabar Linear	30
Statistik	31
2.3 Panel Data = pandas	32
Series dan Dataframe	33
Eksplorasi Data dengan pandas	34
Tipe Data pada pandas	34
Manipulasi Data dengan pandas	35
Mengimpor Data	38
Melakukan Filtering pada Data	41
Mengelompokkan Data	41
BAB III VISUALISASI DATA	44
3.1 Visualisasi Apa yang Tepat ?	44
3.2 Visualisasi Data dengan Pandas dan Matplotlib	46
3.3 Visualisasi Data dengan Seaborn	47
3.4 Visualisasi data dengan Plotly	48
3.5 Scatter Plot	49
3.6 Line Chart	53
3.7 Bar Chart	54
3.8 Pie Chart	56
3.9 Histogram	58
3.10 Box Plot	61
3.11 Heatmap	64
3.12 Pairplot	68
BAB IV UJI HIPOTESIS DAN ANALISIS MEAN	70
4.1 Sampel dan Populasi	70
4.2 Uji Hipotesis	70
Hipotesis Nol dan Hipotesis Alternatif	70
Kesalahan	71
Kesimpulan	72
4.3 Uji Normalitas	72
Q-Q Plot	73
Uji Shapiro – Wilk	74
Uji Jarque Bera (JB)	75
4.4 Uji Hipotesis dengan Satu Sampel	76
Uji Hipotesis Mean Sampel Besar Populasi Berdistribusi Sembarang	76
Komputasi Uji Mean Sampel Besar Populasi Berdistribusi Sembarang dengan Python	77
Uji Hipotesis Mean Populasi Normal	78
Komputasi Uji Mean Populasi Normal dengan Python	79
Uji Proporsi Satu sampel	82

Komputasi Uji Proporsi Satu Sampel dengan Python	82
4.5 Uji Dua Sampel	84
Uji Proporsi Dua Sampel Independen.....	85
Komputasi Uji Proporsi Dua Sampel Independen Dengan Python	86
Uji Homogenitas (Kesamaan Varian).....	89
Uji Selisih Mean Dua Sampel Independen, Variansi Populasi Sama	90
Komputasi Uji Selisih Mean Dua Sampel Independen, Variansi Populasi Sama Dengan Python	91
Uji Selisih Mean Dua Sampel Independen, Variansi Populasi Berbeda (Uji Welch)	93
Komputasi Uji Selisih Mean Dua Sampel Independen, Variansi Populasi Berbeda (Uji Welch) Dengan Python	95
Uji Selisih Mean Dua Sampel Berpasangan dari Populasi Normal (Paired t-test)	97
Komputasi Uji Selisih Mean Dua Sampel Berpasangan dari Populasi Normal (Paired t-test) dengan Python.....	98
BAB V ANALISIS REGRESI.....	100
5.1 Korelasi Pearson.....	100
5.2 Komputasi Korelasi Pearson dengan Python	101
5.3 Regresi Linear	101
5.4 Komputasi Regresi Linear Sederhana Menggunakan Python	102
5.5 Komputasi Regresi Linear Berganda dengan Python.....	106
DAFTAR PUSTAKA.....	112

BAB I

PENDAHULUAN

1.1 Apa itu Python ?

Dewasa ini, tersedia cukup banyak software yang dapat digunakan untuk keperluan analisis data baik yang bersifat komersil maupun freeware, diantaranya software yang bersifat umum seperti : R, OpenStat, WINIDAMS, maupun sejumlah software yang didesain secara khusus untuk keperluan analisis statistika pada sejumlah bidang tertentu.

Python merupakan bahasa pemrograman yang diciptakan oleh Guido van Rossum pada tahun 1989. Beliau menamai ciptaanya berdasarkan nama comedian Inggris *Monty Python*. Pada buku ini akan dibahas penggunaan software open source dan freeware Python untuk keperluan analisis statistika dasar seperti visualisasi, analisis mean, dan analisis regresi.

Salah satu keunggulan Python adalah Bahasa yang dapat dibaca dan dimengerti oleh hampir semua orang. Python mengeksekusi kodenya tanpa menerjemahkan kedalam bahasa mesin, berbeda dengan bahasa C/C++, FORTRAN atau Java yang dieksekusi oleh CPU komputer. Namun meskipun begitu, bahasa seperti Python memang lebih lambat dibandingkan bahasa yang dieksekusi oleh CPU.

1.2 Kelebihan dan Kekurangan Python

Berikut adalah beberapa kelebihan dan kelemahan dari bahasa pemrograman Python sehingga diharapkan dengan mengetahuinya kita dapat memutuskan apakah kita bisa menggunakan Python untuk kebutuhan kita dan apakah software ini merupakan yang terbaik untuk menyelesaikannya.

Kelebihan

- Sintaksnya yang sederhana membuat penulisan program Python cepat dan umumnya mengurangi peluang terjadi kesalahan
- Gratis - Python dan library terkaitnya tidak dikenakan biaya dan sumbernya terbuka, tidak seperti software komersial seperti Mathematica dan MATLAB.
- Didukung lintas platform : Python tersedia untuk berbagai system computer yang kita kenal seperti Windows, Unix, Linux, dan macOS.
- Python mempunyai library yang memuat sangat banyak modul dan package yang dapat digunakan untuk menyelesaikan masalah di berbagai bidang. Beberapa diantaranya tersedia sebagai bagian dari “Standard Library” yang disediakan secara default dan ada juga yang dapat didownload secara terpisah seperti NumPy, SciPy, Matplotlib, dan Pandas.
- Python relatif mudah untuk dipelajari. Sintaks yang digunakan pada Python dapat dikatakan seperti “Bahasa manusia” begitu juga dengan pengorganisasian sintaksnya sehingga membuatnya lebih mudah dipelajari daripada pemrograman lain.

- Python bersifat fleksibel. Python sering kali dikatakan sebagai Bahasa “multi-paradigm” yang memuat berbagai fitur baik procedural, object-oriented dan pemrograman fungsional.

Kekurangan

- Kecepatan eksekusi Python kalah cepat. Jika dibandingkan dengan Bahasa pemrograman seperti C, C++, Fortran, dan Java, kecepatan eksekusi dan debugging pemrograman Python relative lebih lambat. Untuk penggunaan berat dengan menggunakan NumPy dan SciPy dapat mengurangi fleksibilitas karena library ini menggunakan Bahasa C didalamnya.
- Sulit untuk menyembunyikan source code Python untuk menghindari plagiarisme
- Perkembangan software Python yang sangat cepat membuat ketimpangan kompatibilitas antar versi. Sebagai contoh Python 2 dan Python 3 mempunyai banyak sekali perbedaan dalam hal penulisan code sehingga ada beberapa code yang dapat dieksekusi dengan Python 3 namun tidak bisa dieksekusi dengan versi yang ada di bawahnya. Untuk keperluan penyederhanaan, pada **buku ini akan digunakan Python 3.**

1.3 Menginstall Python Dalam Sistem Operasi Windows

Instalasi Python dapat didownload secara gratis pada website official <https://www.python.org/> pada bagian Downloads pilih versi Python yang ingin diinstall. Pada buku ini akan digunakan versi terakhir pada saat modul ini dibuat yaitu Python 3.12.2. Pastikan menginstall sesuai spesifikasi device yang kita punya (dapat dicek dengan langkah **Start > Settings > System > About** di bagian kiri paling bawah. Dan lihat pada bagian **System type**), pada buku ini digunakan installer untuk Windows 64-bit.

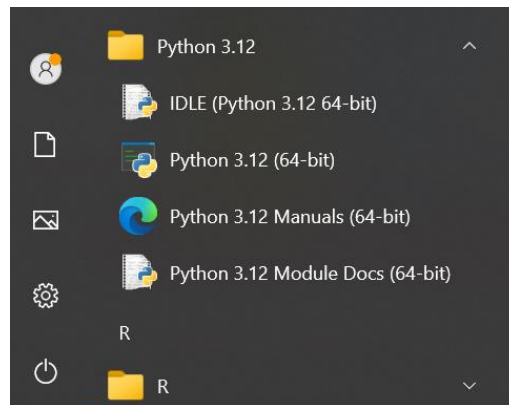
Files

Version	Operating System	Description	MD5 Sum	File Size	PGP	Sigstore	SBOM
Gzipped source tarball	Source release		4e64a004f8ad9af1a75607cfd0d5a8c8	25.9 MB	SIG	.sigstore	SPDX
XZ compressed source tarball	Source release		e7c178b97b8f7ccd677b94d614f7b3c	19.6 MB	SIG	.sigstore	SPDX
macOS 64-bit universal2 installer	macOS	for macOS 10.9 and later	f88981146d943b5517140fa96e96f153	43.5 MB	SIG	.sigstore	
Windows installer (64-bit)	Windows	Recommended	44abfae489d87cc005d50a9267b5d58d	25.4 MB	SIG	.sigstore	
Windows installer (ARM64)	Windows	Experimental	f769b05cd9d336d2d6e3f6399cb573be	24.7 MB	SIG	.sigstore	
Windows embeddable package (64-bit)	Windows		ded837d78a1efa7ea47b31c14c756faa	10.6 MB	SIG	.sigstore	
Windows embeddable package (32-bit)	Windows		787d286b66a3594e697134ca3b97d7fe	9.4 MB	SIG	.sigstore	
Windows embeddable package (ARM64)	Windows		1ffc0d4ea3f02a1b4dc2a6e74f75226d	9.8 MB	SIG	.sigstore	
Windows installer (32-bit)	Windows		bc4d721cf44a52fa9e19c1209d45e8c3	24.1 MB	SIG	.sigstore	

Setelah mendownload installer Python (klik pada lingkaran merah), selanjutnya lakukan langkah instalasi. Centang semua pilihan pada jendela instalasi dan disarankan memilih opsi **Install Now**. Setelah proses instalasi selesai klik **Close**.



Untuk mengecek apakah Python sudah terinstall, klik tombol windows lalu pada daftar aplikasi di bagian kiri akan ada Python 3.12 sebagai tanda bahwa Python telah terinstall pada laptop kita.



Terdapat dua aplikasi yang terinstall yaitu IDLE dan Python 3.12, keduanya memberikan fungsi yang sama yaitu dapat digunakan untuk membuat code untuk Python namun aplikasi Python 3.12 hanya dapat digunakan sebagai terminal untuk eksekusi perintah, berbeda dengan IDLE yang selain dapat digunakan sebagai terminal eksekusi juga dapat digunakan untuk membuat script kode Python. Untuk penjelasan lebih lengkap mengenai IDLE silahkan membaca dokumentasi pada link berikut <https://docs.python.org/3/library/idle.html>.

1.4 Jupyter Notebook

Integrated development environment (IDE) adalah aplikasi perangkat lunak yang membantu para pemrogram mengembangkan kode perangkat lunak secara efisien. Layaknya para penulis yang menggunakan editor teks dan para akuntan yang menggunakan spreadsheet, developer perangkat lunak menggunakan IDE untuk memudahkan pekerjaan mereka. Secara default setelah melakukan instalasi Python maka akan terinstall IDE default dari Python yang bernama IDLE. Namun terdapat kekurangan dalam penggunaan IDE ini sehingga dibutuhkan IDE lain.

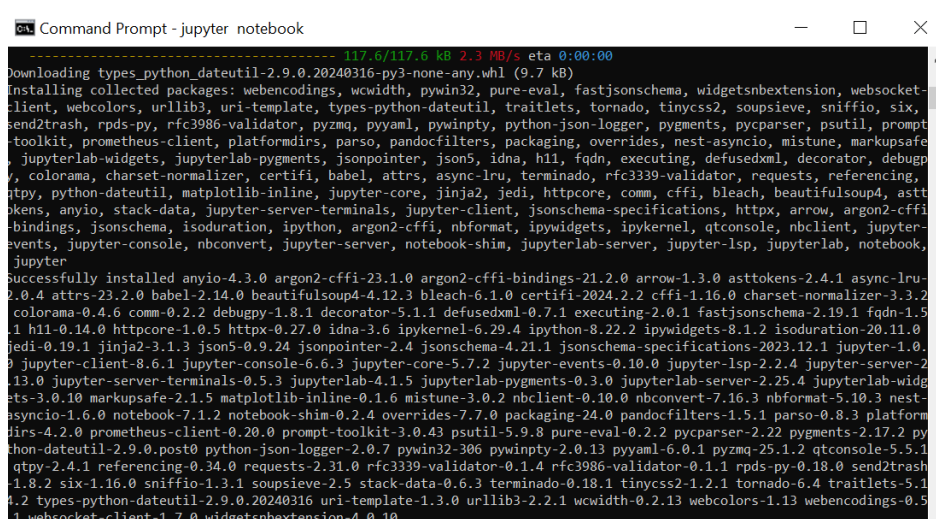
Secara garis besar terdapat dua jenis IDE yaitu IDE lokal dan IDE cloud. Contoh dari IDE lokal diantaranya PyCharm, Visual Studio Code, Spyder, Jupyter, dan masih banyak lagi. Berbeda dengan IDE yang harus diinstall pada komputer kita, IDE cloud dapat diakses secara fleksibel dari device manapun dan hanya membutuhkan koneksi internet, salah satu IDE cloud yang sering digunakan adalah *Google Colaboratory*. Pada buku ini hanya akan dibahas mengenai IDE Jupyter Notebook yang merupakan salah satu IDE Python yang gratis dan sangat mudah digunakan untuk penghitungan khususnya dalam bidang *Data Science*.

Jupyter Notebook memberikan kesan interaktif untuk pemrograman Python dengan memanfaatkan web browser (meskipun web browser based namun IDE ini tidak membutuhkan koneksi internet). Keuntungan notebook ini dibandingkan console-based tradisional seperti IPython shell adalah dengan menggunakan aplikasi ini selain untuk penulisan kode juga dapat mengkombinasikan dokumentasi (termasuk dalam bentuk LaTeX), gambar, dan media lain seperti video.

Instalasi Jupyter Notebook

Untuk melakukan instalasi Jupyter Notebook sangatlah mudah. Kita dapat melakukan langkah-langkah berikut

- Buka *Command Prompt* dengan **Klik logo Windows > Ketikkan cmd > Lalu jendela Command Prompt akan muncul.**
- Pertama kali yang harus dilakukan adalah menginstall *pip* dengan versi terbaru. Versi sebelumnya mungkin akan menyebabkan eror dalam beberapa kejadian. Ketikkan pada command prompt perintah berikut
`pip3 install --upgrade pip`
- Selanjutnya lakukan instalasi dengan mengetikkan perintah berikut
`pip3 install jupyter`
- Jika proses berhasil maka akan muncul status seperti berikut.



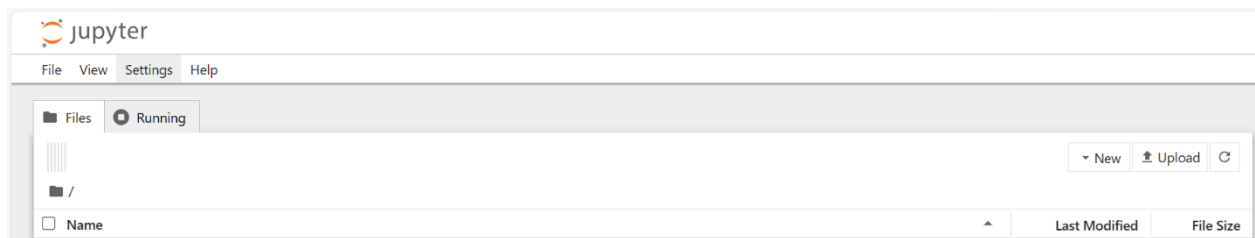
```
Command Prompt - jupyter notebook
117.6/117.6 kB 2.3 MB/s eta 0:00:00
Downloading types-python-dateutil-2.9.0.20240316-py3-none-any.whl (9.7 kB)
Installing collected packages: webencodings, wcwidth, pywin32, pure-eval, fastjsonschema, widgetsnbextension, websocket-client, webcolors, urllib3, uri-template, types-python-dateutil, traitlets, tornado, tinycss2, soupsieve, sniffio, six, send2trash, rpsd-py, rfc3986-validator, pyzmq, pyyaml, pywinpty, python-json-logger, pygments, pycparser, psutil, prompt-toolkit, prometheus-client, platformdirs, parso, pandocfilters, packaging, overrides, nest-asyncio, mistune, markupsafe, jupyterlab-widgets, jupyterlab-pygments, jsonpointer, json5, idna, h11, fqdn, executing, defusedxml, decorator, debugpy, colorama, charset-normalizer, certifi, babel, attrs, async-lru, terminado, rfc3339-validator, requests, referencing, qtpy, python-dateutil, matplotlib-inline, jupyter-core, Jinja2, jedi, httpcore, comm, cffi, bleach, BeautifulSoup4, asttokens, anyio, stack-data, jupyter-server-terminals, jupyter-client, jsonschema-specifications, httpx, arrow, argon2-cffi-bindings, jsonschema, isoduration, ipython, argon2-cffi, nbformat, ipywidgets, ipykernel, qtconsole, nbclient, jupyter-events, jupyter-console, nbconvert, jupyter-server, notebook-shim, jupyterlab-server, jupyter-lsp, jupyterlab, notebook
Successfully installed anyio-4.3.0 argon2-cffi-23.1.0 argon2-cffi-bindings-21.2.0 arrow-1.3.0 asttokens-2.4.1 async-lru-2.0.4 attrs-23.2.0 babel-2.14.0 BeautifulSoup4-4.12.3 bleach-6.1.0 certifi-2024.2.2 cffi-1.16.0 charset-normalizer-3.3.2 colorama-0.4.6 comm-0.2.2 debugpy-1.8.1 decorator-5.1.1 defusedxml-0.7.1 executing-2.0.1 fastjsonschema-2.19.1 fqdn-1.5.1 h11-0.14.0 httpcore-1.0.5 httpx-0.27.0 idna-3.6 ipykernel-6.29.4 ipython-8.22.2 ipywidgets-8.1.2 isoduration-20.11.0 jedi-0.19.1 Jinja2-3.1.3 json5-0.9.24 jsonpointer-2.4 jsonschema-4.21.1 jsonschema-specifications-2023.12.1 jupyter-1.0.0 jupyter-client-8.6.1 jupyter-console-6.6.3 jupyter-core-5.7.2 jupyter-events-0.10.0 jupyter-lsp-2.24.0 jupyter-server-2.13.0 jupyter-server-terminals-0.5.3 jupyterlab-4.1.5 jupyterlab-pygments-0.3.0 jupyterlab-server-2.25.4 jupyterlab-widgets-3.0.10 markupsafe-2.1.5 matplotlib-inline-0.1.6 mistune-3.0.2 nbclient-0.10.0 nbconvert-7.16.3 nbformat-5.10.3 nest-asyncio-1.6.0 notebook-7.1.2 notebook-shim-0.2.4 overrides-7.7.0 packaging-24.0 pandocfilters-1.5.1 parso-0.8.3 platformdirs-4.2.0 prometheus-client-0.20.0 prompt-toolkit-3.0.43 psutil-5.9.8 pure-eval-0.2.2 pycparser-2.22 pygments-2.17.2 python-dateutil-2.9.0.post0 python-json-logger-2.0.7 pywin32-306 pywinpty-2.0.13 pyyaml-6.0.1 pyzmq-25.1.2 qtconsole-5.5.1 qtpy-2.4.1 referencing-0.34.0 requests-2.31.0 rfc3339-validator-0.1.4 rfc3986-validator-0.1.1 rpsd-py-0.18.0 send2trash-1.8.2 six-1.16.0 sniffio-1.3.1 soupsieve-2.5 stack-data-0.6.3 terminado-0.18.1 tinycss2-1.2.1 tornado-6.4 traitlets-5.1.2 types-python-dateutil-2.9.0.20240316 uri-template-1.3.0 urllib3-2.2.1 wcwidth-0.2.13 webcolors-1.13 webencodings-0.5.1 websocket-client-1.7.0 widgetsnbextension-4.0.10
```


Menjalankan Jupyter Notebook

Untuk membuka Jupyter notebook ketikkan pada command prompt perintah berikut

```
jupyter notebook
```

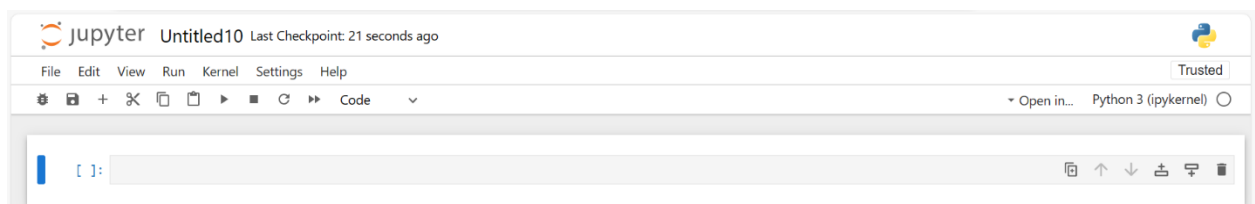
Secara otomatis jendela Jupyter Notebook pada web browser akan terbuka pada URL dari aplikasi Jupyter Notebook yang berada di system lokal. Secara default URL tersebut adalah localhost:8888/tree.



Halaman home dari Jupyter Notebook memuat daftar project kita yang tersedia di direktori dimana server notebook ini dimulai (Sebagai contoh pada saat membuka di command line muncul `C:\Users\Dell>jupyter notebook` maka folder home pada Jupyter Notebook dapat dibuka pada Folder `C:\Users\Dell`. Terdapat dua tab pada jendela Jupyter Notebook yaitu Files memuat semua file termasuk notebook-notebook kita dan subdirektori di dalam direktori kerja kita. Dan tab Running memuat semua notebook yang sedang aktif pada sesi sekarang.

Membuat Notebook

Dari jendela Jupyter Notebook kita dapat membuat notebook baru dengan cara “**Klik New > Notebook : Python 3**” atau membuka notebook yang tersedia dengan cara mengklik nama filenya. Untuk mengimpor proyek yang telah kita kerjakan di tempat lain, dapat melalui menu **Upload** atau drag file ke dalam jendela Jupyter Notebook. Tunggu beberapa saat dan jendela di tab baru akan terbuka.



Di dalam jendela notebook baru terdapat *title bar*, *menu bar*, dan *tool bar*. Pada title bar terdapat nama dari notebook yang kita buka, secara default dokumen kita akan diberi nama “**Untitled**”, untuk mengubahnya dapat dilakukan dengan cara mengklik bagian ini dan mengganti nama sesuai yang diinginkan. Di dalam menu baru terdapat beberapa pilihan untuk menyimpan, mengcopy, mencetak Jupyter Notebook. Tool bar memuat ikon-ikon sebagai shortcut untuk operasi-operasi yang dapat diakses melalui menu bar.

Cell

Cell merupakan area untuk menaruh kode di Jupyter Notebook. Untuk menjalankan cell tekan SHIFT+ENTER di keyboard atau tombol RUN di Toolbar pada cell aktif sehingga kode akan dieksekusi dan menghasilkan output. Terdapat tiga tipe sel untuk input, yaitu

- Code cells : Secara default jika kita menambahkan sel maka akan bertipe Code, tipe ini digunakan untuk eksekusi kode.
- Markdown cells : Tipe ini memperbolehkan kita untuk membuat dokumentasi baik dalam bentuk tulisan, LaTeX, maupun memasukkan gambar/video kedalam notebook.
- Raw cells : Jika kita memasukkan sesuatu kedalam sel ini maka isi dan formatnya sama persis dengan apa yang dituliskan.

```
•[1]: # Code Cell
      1+1

[1]: 2
```

Contoh *Markdown Cell* penulisannya mirip dengan html







Contoh *Raw Cell*, akan dieksekusi sesuai input dan tidak akan berubah formatnya





Toolbar

Toolbar mempunyai beberapa tombol shortcut penting



Fungsi dari masing-masing toolbar adalah sebagai berikut

Simbol	Keterangan	Deskripsi
	Save and checkpoint	Menyimpan file
	Insert cell below	Menambah cell dibawah cell aktif
	Cut selected cell	Memotong cell aktif
	Copy selected cell	Menyalin cell aktif
	Paste cell below	Menempel ke cell dibawah cell aktif
	Run	Mengeksekusi cell aktif

	Interrupt kernel	Menghentikan paksa kernel
	Restart the kernel	Memulai ulang kernel
	Restart the kernel and re-run the whole notebook	Memulai ulang kernel dan menjalankan seluruh kode notebook
Code 	Cell type	Tipe cell (code, markdown, raw)

1.5 Anaconda

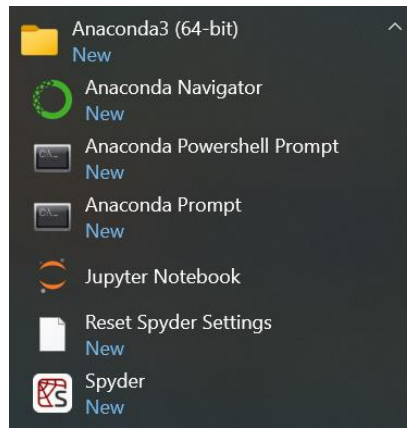
Anaconda adalah distribusi Python yang disertai dengan sejumlah besar paket dan alat bantu seperti Jupyter Notebook, Spyder, dan Visual Studio Code yang umumnya digunakan dalam ilmu data dan komputasi ilmiah. Distribusi ini mencakup interpreter Python itu sendiri bersama dengan ribuan pustaka yang umum digunakan seperti NumPy, Pandas, Matplotlib, scikit-learn, dll. Pengguna dapat dengan mudah menginstal dan mengelola paket-paket ini menggunakan Conda atau pip (manajer paket Python standar).

Conda adalah manajer paket dan lingkungan yang dikembangkan oleh Anaconda, Inc. Ini memungkinkan pengguna untuk menginstal, mengelola, dan memperbarui paket-paket Python dan non-Python dengan mudah. Conda juga memungkinkan Anda membuat dan mengelola lingkungan kerja yang terisolasi, yang memungkinkan Anda memiliki proyek-proyek yang menggunakan versi paket yang berbeda-beda tanpa konflik.

Instalasi Anaconda

Untuk dapat menginstall Anaconda cukup mendownload aplikasi ini melalui link <https://www.anaconda.com/download/success> . Meskipun ukurannya terbilang cukup besar namun bagi pengguna Python yang lebih memilih untuk menginstall semua tools secara menyeluruh aplikasi ini terbilang sangat simpel karena tidak harus menginstall Python terlebih dahulu dan nantinya tidak harus melakukan instalasi library yang bukan menjadi bawaan Python seperti NumPy, matplotlib, pandas, dan lain-lain. Library yang terinstall secara otomatis pada Anaconda dapat dilihat pada link <https://docs.continuum.io/free/anaconda/pkg-docs/>.

Setelah melakukan instalasi, kita bisa menemukan daftar aplikasi yang terinstall bersama Anaconda pada Start Menu seperti gambar di bawah. Kita bisa membuat shortcut dari Jupyter Notebook pada desktop sehingga nantinya tidak perlu lagi membukanya melalui Command Prompt.



1.6 Tipe data

Sebelum masuk kedalam pembahasan mengenai tipe data perlu diketahui bahwasanya Python berbeda dengan bahasa lain, dimana Python menggunakan indentasi di dalam penulisan kodenya, berbeda dengan C++ atau R yang menggunakan tanda kurung kurawal untuk memisahkan blok kode.

```
for i in [1,2,3,4]:
    print i
    for j in [1,2,3] :
        print j
        print i+j
    print i
print "looping selesai"
```

Hal ini membuat kode Python sangat nyaman dibaca, namun ini juga membuat kita harus teliti dalam indentasi. Kita juga dapat menggunakan backslash untuk menunjukkan sebuah kalimat dilanjutkan di baris selanjutnya.

```
dua_tambah_tiga = 2+\
3
dua_tambah_tiga
```

5

Sampai pembahasan ini kita telah membicarakan mengenai manfaat dari menggunakan Python dan penjelasan mengenai Jupyter Notebook sebagai IDE yang kita gunakan pada pembahasan di buku ini. Lebih lanjut akan dijelaskan contoh-contoh operasi sederhana yang seringkali digunakan untuk pengolahan data. Berikut ini akan diberikan operasi-operasi aritmatika seperti penjumlahan dan pengurangan.

Operasi	Operator
Penjumlahan	+
Pengurangan	-
Perkalian	*
Pembagian	/

Dalam pemrograman Python kita mengenal *variabel* yang secara bisa kita anggap sebagai tempat untuk menyimpan objek yang kita punya. Kita tidak perlu mendeklarasikan variabel sebelum digunakan dan tidak perlu mengenalkan apa tipe variabel kita tidak seperti C/C++. Namun, kita perlu mengetahui tipe dasar yang digunakan dalam pemrograman Python sehingga kita dapat membuat objek lain.

Bilangan : Integer dan Float

Seperti yang kita bayangkan, Python dapat digunakan sebagai kalkulator. Secara garis besar, Python menggunakan dua tipe dasar bilangan yaitu **Integer (bilangan bulat contoh -2,-1,0,75)** dan **Float (bilangan desimal contoh 0.25, 1.4,3.14)** . Kita dapat memasukkan nilai kedalam variabel seperti berikut

```
bil = 2 #Memasukkan nilai 2 kedalam variabel bernama bil
```

Jika cell diatas dirun maka tidak akan memunculkan output, namun hanya menyimpan nilai 2 kedalam variabel bernama `bil` . Kita dapat mengecek tipe dari objek yang kita punya seperti berikut

```
type(bil)
```

```
int
```

sesuai yang kita punya variabel `bil` memuat objek integer. Selanjutnya akan diberikan contoh tipe data float

```
bil_float = 1.5
```

```
type(bil_float)
```

```
float
```

Meskipun berbeda tipe akan tetapi kita bisa mengoperasikan dua variabel yang mempunyai jenis yang sama seperti variabel `bil` dan `bil_float` . Apabila kedua variabel tersebut dioperasikan maka hasilnya akan menjadi float sebagai tipe yang lebih umum diantara keduanya.

```
x = bil_float*bil
```

```
print(x)
```

```
type(x)
```

```
3.0
```

```
float
```

Untuk mengubah tipe data dari float ke dalam bentuk integer dapat menggunakan sintaks berikut `int(x)` .

Sebagai contoh, kita dapat mengubah objek `x` ke dalam bentuk integer seperti berikut (perhatikan bahwa kita harus memasukkan nilai hasil konversi kedalam variabel `x` lagi untuk mengupdate nilainya).

```
x = int(x)
type(x)

int
```

Terdapat tiga fungsi yang dapat digunakan untuk mengkonversi tipe suatu variabel yaitu

- `int()` untuk mengkonversi ke dalam bentuk integer baik dari bentuk desimal maupun string.
- `float()` untuk mengkonversi ke dalam bentuk float baik dari bentuk integer maupun string.
- `str()` untuk mengkonversi ke dalam bentuk string baik dari bentuk integer maupun float.

Perhatikan contoh berikut

```
x = float(2) #x akan menyimpan nilai 2.0
y = int(2.0) #y akan menyimpan nilai 2
z = str(3.14) #z akan menyimpan nilai '3.14' sebagai string
```

String

Selain data berbentuk angka, kita seringkali menemukan data non numerik seperti nama, nama jalan, nama negara, dan lain-lain. Tipe data seperti ini disebut dengan **string** yaitu barisan dari karakter-karakter/huruf. String di dalam Python didefinisikan dengan cara memberikan tanda (`' '`) atau (`" "`). Berikut ini diberikan contohnya

```
contoh1 = 'ini contoh string'
contoh2 = "ini juga contoh string"
```

String dapat dimunculkan pada layar dengan menggunakan perintah `print()`

```
print(contoh1)
type(contoh1)

ini contoh string
str
```

Perhatikan bahwa tipe data dari `contoh1` adalah `str` yang menunjukkan bahwa `contoh1` merupakan sebuah string. Selanjutnya Python dapat melakukan operasi penjumlahan pada bilangan, di sisi lain, Python juga dapat melakukan operasi pejumlahan pada string yang menghasilkan gabungan dari dua string yang dioperasikan.

```
#Contoh pendefinisian dua variabel #secara langsung
string1, string2 = "pejuang", "data"
print(string1 + " " + string2)

pejuang data
```

Jika menggunakan bahasa pemrograman lain, sintaks diatas terlihat aneh. Inilah keistimewaan dari Python yaitu dapat membuat banyak variabel dalam satu baris tanpa harus mendefinisikan satu-satu. Jadi sintaks diatas memasangkan string 'pejuang' ke dalam variabel string1 dan string 'data' kedalam variabel string2. Perlu diingat bahwa kita tidak dapat menjumlahkan dua variabel yang berbeda tipe datanya seperti string dengan integer, namun kita bisa menempelkan bilangan ke dalam string dengan cara mengkonversi integer tersebut ke dalam bentuk string dengan menggunakan perintah `str()`.

```
string1+2
-----
TypeError                                Traceback (most recent call last)
Cell In[10], line 1
----> 1 string1+2

TypeError: can only concatenate str (not "int") to str

string1+str(2)
'pejuang2'
```

Selain penjumlahan pada string, kita juga bisa melakukan perkalian string dengan bilangan bulat. Tentu saja ini menghasilkan pengulangan string sebanyak yang kita mau. Berikut diberikan contohnya

```
string1*3
'pejuangpejuangpejuang'
```

Selain yang telah dijelaskan diatas, masih banyak lagi operasi yang bisa dilakukan terhadap string, namun kita akan membahasnya di lain kesempatan. Perlu diingat bahwa string adalah objek yang *immutable*, yang berarti kita tidak mengedit isi dari string tanpa menghapusnya terlebih dahulu.

1.7 Koleksi pada Python

Sebuah wadah yang berisi objek-objek pada Python disebut sebagai koleksi. Pada bahasa pemrograman R kita mengenal objek vector yang merupakan kumpulan dari objek-objek dengan tipe yang sama. Begitu juga pada bahasa pemrograman Python, kita mengenal koleksi objek-objek baik yang mempunyai tipe yang sama maupun tidak.

Sebuah koleksi pada Python merupakan wadah untuk objek-objek berbeda dan pada akhirnya dianggap sebagai satu unit. Python mempunyai empat tipe koleksi yaitu : List, set, tuple, dan dictionary. Setiap koleksi merupakan *iterable object* yang berarti kita dapat menganggap isinya sebagai barisan sehingga dapat melakukan loop terhadap isi dari koleksi dan mengakses setiap item satu persatu. Pada kesempatan ini hanya akan dijelaskan mengenai tipe list dan dictionary yang sangat sering digunakan dalam pengolahan data.

List

List merupakan tipe koleksi yang sangat sering digunakan di dalam bahasa pemrograman Python, layaknya tipe objek vector pada R. Perbedaannya adalah kita dapat memasukkan berbagai tipe data ke dalam list, baik integer, string, bahkan memasukkan list sekalipun. Item di dalam list dapat diakses satu persatu, dapat ditambahkan atau dihapus, dan dapat diurutkan.

Untuk membuat list kita menggunakan tanda kurung siku “[]”. Berbeda dengan string, list merupakan objek yang *mutable* sehingga kita bisa mengubah isi dari list secara langsung. Lebih lanjut, item-item di dalam list merupakan item yang **terurut**. Berikut diberikan contoh dari list

```
angka = [1,2,3,4,5]
kota = ["Sleman", "Gunung kidul", "Surabaya", "Tulungagung"]
nilai = [90,85,"No1",90.5]
```

Hal yang perlu diingat adalah, pada bahasa pemrograman Python indeks suatu list dimulai dari 0 sehingga apabila kita bermain petak umpet dengan Python maka Python akan menghitung dengan “0,1,2,3,...”. Perhatikan ilustrasi berikut, pada list kota diatas beserta indeksnya.

Sleman	Gunung kidul	Surabaya	Tulungagung
0	1	2	3
-4	-3	-2	-1

sehingga apabila kita memanggil kota[2] yang akan muncul adalah Surabaya. Perlu diketahui bahwa untuk memanggil elemen pada list tidak harus menggunakan bilangan positif namun juga dapat menggunakan angka negatif.

```
Tulungagung = kota[-1]
Surabaya = kota[-2] #eight bernilai 8 yaitu item kedua dari akhir pada x
```

Ada banyak cara untuk membuat sebuah list selain menggunakan “[]” kita juga bisa membuat list dari list yang sudah ada, mengingat list merupakan objek iterable. Perhatikan contoh berikut

```
angka
[1, 2, 3, 4, 5]
angka_kuadrat = [x**2 for x in angka]
angka_kuadrat
[1, 4, 9, 16, 25]
```

Secara umum untuk membuat list dari objek iterabel lain dapat menggunakan pola berikut

```
newlist = [expression for item in iterable if condition==True]
```


Berikut ini diberikan daftar operasi-operasi yang dapat digunakan terhadap tipe data list

Fungsi atau Metode	Contoh	Output	Deskripsi
<code>len()</code>	<code>len(angka)</code>	5	Banyaknya item di dalam list
<code>max()</code>	<code>max(angka)</code>	5	Nilai terbesar (harus memiliki tipe yang sama)
<code>min()</code>	<code>min(angka)</code>	1	Nilai terkecil (harus memiliki tipe yang sama)
<code>sum()</code>	<code>sum(angka)</code>	15	Jumlahan item di dalam list (harus memiliki tipe angka)
<code>count(k)</code>	<code>angka.count(2)</code>	1	Menghitung jumlah kemunculan objek "k" di dalam list
<code>index()</code>	<code>nilai.index("Nol")</code>	2	Mencari indeks kemunculan pertama suatu objek
<code>reverse()</code>	<code>angka.reverse()</code>	<code>[5,4,3,2,1]</code>	Membalik urutan item di dalam list
<code>clear()</code>	<code>angka.clear()</code>	<code>[]</code>	Menghapus semua isi di dalam list (tidak dengan variabelnya)
<code>append()</code>	<code>angka.append(100)</code>	<code>[1,2,3,4,5,100]</code>	Menyisipkan objek di akhir list
<code>extend()</code>	<code>angka.extend([9,7])</code>	<code>[1,2,3,4,5,9,7]</code>	Menyisipkan list baru di akhir list pertama
<code>del</code>	<code>del kota[2]</code>	<code>["Sleman", "Gunung kidul", "Tulungagung"]</code>	Menghapus item sesuai indeks yang dimasukkan
<code>remove()</code>	<code>angka.remove(5)</code>	<code>[1,2,3,4]</code>	Menghapus kejadian pertama dari suatu objek
<code>insert()</code>	<code>angka.insert(1,10)</code>	<code>[1,10,2,3,4,5]</code>	Angka pertama menjadi calon indeks dari item

			(angka kedua) setelah disisipkan
+	<code>['a','b']+[1,2]</code>	<code>['a','b',1,2]</code>	Menggabungkan dua list
*	<code>[0]*3</code>	<code>[0,0,0]</code>	Pengulangan

Telah disebutkan bahwasanya list mempunyai indeks untuk mewakili elemennya, hal ini dapat digunakan untuk *slicing* seperti kita melakukan blok pada cell Excel. Berikut diberikan contoh pemanggilan elemen di dalam list dan juga slicing

```
x = range(10) #Mendefinisikan list x = 0,1,2,...,9
zero = x[0] #zero bernilai 0 yaitu item dengan indeks 0 dari x
first_three = x[:3] #Mengambil item ke 0 sampai 2 (3-1)
three_to_end = x[3:] #Mengambil item ke 3 sampai akhir
one_to_four = x[1:5] #Mengambil item ke 1 sampai 4 (5-1)
x2 = x[:] #Mengcopy x
```

Python juga menggunakan perintah `in` untuk mengecek apakah suatu objek berada pada list

```
1 in [1,2,3] #bernilai True
"satu" in [1,2,3] #bernilai False
```

Dictionary

Salah satu objek yang paling penting dalam bahasa pemrograman Python adalah dictionary. Berbeda dari list, indeks dari list adalah bilangan terurut dari 0 sedangkan index dari dictionary dapat berupa objek apapun yang nantinya index ini disebut sebagai *key* dan nilainya adalah *value*. Jadi dictionary di dalam Python adalah koleksi objek-objek yang tersusun dari *key* dan *value*.

Kita dapat mendefinisikan dictionary dengan menggunakan kurung kurawal “{}”. Setiap pasangan key-value dikonstruksi dengan menggunakan kolon “:” diantara key dan valuenya, dan setiap pasangan key-value berbeda dipisahkan dengan koma. Perhatikan contoh berikut

```
kosong = {}
kosong2 = {}
nilai = {"Alzim" : 100, "Badril" : 99, "Karjo" : 50}
jabatan = {"Alzim" : "Kapten", "Badril" : ["Wakil", "Ketua divisi"]}
```

Perhatikan bahwa key pada dictionary merupakan objek yang immutable seperti string, sedangkan value dari key tersebut bisa berupa objek apapun bahkan list atau dictionary juga. Kita dapat menampilkan value dari suatu key dengan memanggil nama key yang berpasangan.

```
jabatan["Badril"]  
['Wakil', 'Ketua divisi']
```

Apabila kita mencari value dari suatu key yang tidak terdaftar maka akan muncul error seperti berikut

```
jabatan["Umam"]  
  
-----  
KeyError                                Traceback (most recent call last)  
Cell In[27], line 1  
----> 1 jabatan["Umam"]  
  
KeyError: 'Umam'
```

Kita dapat mengecek apakah suatu objek merupakan key dari dictionary yang kita punya dengan menggunakan `in`

```
"Alzim" in nilai #bernilai True  
"Umam" in nilai #bernilai False
```

Layaknya list, dictionary merupakan objek yang *mutable* sehingga kita bisa menambahkan key-value baru ke dalam dictionary. Selain itu kita juga bisa mengganti value dari key yang sudah tersedia di dalam dictionary dengan memasang key ini dengan value baru.

```
nilai["Umam"] = 100  
print(nilai)  
nilai["Badril"] = 95  
print(nilai)  
  
{'Alzim': 100, 'Badril': 95, 'Karjo': 50, 'Umam': 100}  
{'Alzim': 100, 'Badril': 95, 'Karjo': 50, 'Umam': 100}
```

Berikut ini diberikan daftar operasi-operasi yang dapat digunakan terhadap tipe data dictionary

Fungsi atau Metode	Contoh	Output	Deskripsi
<code>keys()</code>	<code>names = nilai.keys()</code>	<code>["Alzim", "Badril", "Karjo"]</code>	Menampilkan semua key yang ada di dalam dictionary

<code>values()</code>	<code>angka = nilai.values()</code>	<code>[100, 99, 50]</code>	Menampilkan semua value yang ada di dalam dictionary
<code>pairs()</code>	<code>names = nilai.keys()</code>	<code>[('Alzim', 100), ('Badril', 99), ('Karjo', 50)]</code>	Menampilkan pasangan key-value yang ada di dalam dictionary ke dalam bentuk tuple
<code>clear()</code>	<code>nilai.clear()</code>	<code>[]</code>	Menghapus semua isi di dalam dictionary (tidak dengan variabelnya)
<code>pop()</code>	<code>nilai.pop("Badri 1")</code>	<code>{"Alzim" : 100, "Karjo" : 50}</code>	Menghapus key-value sesuai yang dimasukkan

1.8 Logika dan Control Flow

Setelah kita memahami benda-benda yang ada di dalam Python selanjutnya akan dikenalkan logika untuk dapat membuat algoritma sesuai *flow* yang kita inginkan. Hal yang perlu diingat adalah di dalam Python, pergantian baris dan indentasi mendefinisikan blok dari kode sehingga *whitespace* merupakan hal terpenting dalam pemrograman Python.

Operator Boolean dan Operator Logika

Kita telah mendiskusikan berbagai tipe objek di dalam Python seperti integer, float, dan string. Terdapat satu tipe lagi yang belum dijelaskan yaitu tipe Boolean. Tipe Boolean hanya terdiri dari dua nilai saja yaitu `True` dan `False`, kedua nilai ini akan kita dapatkan apabila membandingkan dua objek seperti berikut

<code>print(3<10)</code>
<code>True</code>
<code>print(1024>=1025)</code>
<code>False</code>
<code>type(3<10)</code>
<code>bool</code>

Seperti pada bahasa pemrograman lain, kita juga bisa mengkombinasikan banyak perbandingan untuk mendapatkan operasi logika seperti `and`, `or`, dan `not`. Akan diberikan operator-operator logika yang sering digunakan dalam pemrograman.

Operator	Operasi	Contoh
==	Equal to	1==2 #Nilainya False
!=	Not Equal to	1!=2 #Nilainya True
>=	Greater than or equal to	1>=2 #Nilainya False
<=	Less than or equal to	1<=2 #Nilainya True
and	Logical And	(1<2) and (2<=3) #Nilainya True
or	Vectorized Or	(1<2) or (2<=3) #Nilainya True
no	Not	not(1==2) #Nilainya True

Mengingat pentingnya operator Boolean untuk menulis program pada setiap jenis bahasa pemrograman maka kita wajib untuk mengetahui konsep operator Boolean sehingga kita bisa membuat keputusan tentang bagaimana alur program yang ingin kita buat.

Kalimat Kondisional

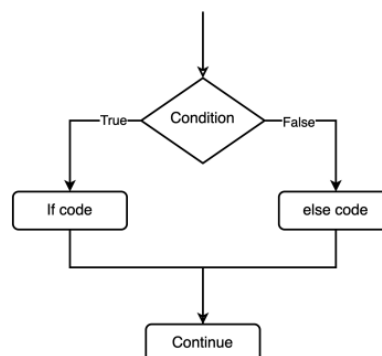
Ketika membuat keputusan kita mempunyai suatu kondisi yang harus dipenuhi sehingga keputusan kita sesuai dengan yang kita inginkan. Dalam pemrograman kita bisa melakukan ini dengan menggunakan operasi Boolean. Kalimat kondisional memperbolehkan kita untuk melakukan sesuatu jika kondisi dari `if` terpenuhi, dan apabila kondisi ini tidak dipenuhi kita masih bisa melakukan aksi lain atau bahkan tidak sama sekali. Dalam Python kita bisa menggunakan alur berikut (perhatikan tanda titik dua : dan indentasi)

```

if kondisi1 :
    kode yang akan dieksekusi jika kondisi1 terpenuhi
elif kondisi2 :
    kode yang akan dieksekusi jika kondisi2 terpenuhi
...
elif kondisiN :
    kode yang akan dieksekusi jika kondisiN terpenuhi
else :
    kode yang akan dieksekusi jika tidak ada kondisi di atasnya
    yang terpenuhi

```

Berikut ini diberikan representasi control flow dari kalimat kondisional dalam bentuk diagram.



Kondisi dalam kalimat kondisional dapat berupa satu perbandingan atau kombinasi operator logika **and**, **or**, dan **not**. Kita dapat menguji banyak kondisi menggunakan operand **elif** (“else if”). Kondisi-kondisi yang kita punya akan diuji sesuai urutan penulisan kode dan ketika satu kondisi terpenuhi maka *flow* akan berhenti dan sisanya tidak akan diuji.

```
siswa = ["Alzimna","Badril","Umam"]
guru = ["Badril","Zimna","Al"]
ketua = "Karjo"

if ketua in siswa :
    print("Ketuanya adalah siswa")
elif ketua in guru :
    print("Ketuanya adalah guru")
else :
    print("Ketuanya salah alamat")
```

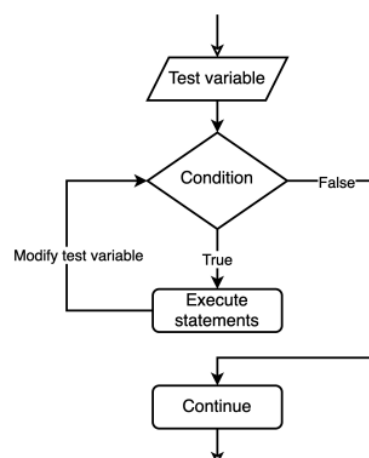
Ketuanya salah alamat

Pengulangan While

Seringkali kita mempunyai list yang berisi angka dan kita mempunyai sebuah aksi yang akan kita lakukan selama kondisi yang kita inginkan dari angka-angka tersebut terpenuhi. Kadang-kadang kita tidak tahu kapan aksi ini dihentikan asalkan kondisinya tidak terpenuhi maka aksinya akan berhenti. Dalam kasus ini kita gunakan pengulangan **while**.

```
while kondisi :
    kode yang akan dieksekusi
    jangan lupa untuk mengupdate variabel yang akan diuji lagi
```

Berikut ini diberikan representasi control flow dari pengulangan **while** dalam bentuk diagram.



Yang perlu diperhatikan adalah eksekusi akan dilakukan apabila variabel pertama yang diuji memenuhi kondisi sehingga apabila kondisi pertama menghasilkan nilai False maka kode tidak akan

dieksekusi. Selain itu pastikan untuk mengupdate variabel setiap aksi dieksekusi supaya tidak terjadi *infinite loop* dan akan mengeksekusi aksinya selamanya.

Kita akan mencoba mempraktekkan pengulangan ini dengan menggunakan nilai dari Karjo pada contoh sebelumnya. Nilai pertama yang dimiliki oleh Karjo adalah 50 dan gurunya memintanya untuk remidi hingga nilainya melebihi 75. Jika setiap remidi nilai Karjo naik 10% maka ada berapa kali gurunya meminta Karjo untuk remidi

```
nilai_karjo = 50
while nilai_karjo < 75 :
    #Jika nilai_karjo masih kurang dari 75 maka remidi
    print("Remidi ya !")

    #setelah remidi nilai karjo naik 10%
    nilai_karjo = 1.1*nilai_karjo
```

```
Remidi ya !
Remidi ya !
Remidi ya !
Remidi ya !
Remidi ya !
```

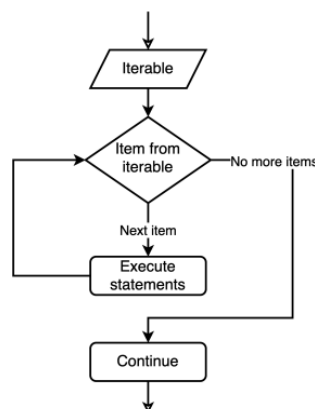
Setelah nilai Karjo melebihi 75 yaitu setelah 5 kali remidi maka Karjo tidak lagi diminta untuk remidi.

Pengulangan For

Ketika kita mengetahui berapa kali eksekusi akan kita ulang maka lebih baik kita menggunakan pengulangan **for**. Kita bisa melakukan iterasi berdasarkan list angka, maupun objek di Python yang *iterabel* seperti list, tuple, atau string. Sintaks untuk pengulangan **for** diberikan sebagai berikut

```
for item in iterable object :
    kode yang akan dieksekusi
```

Berikut ini diberikan representasi control flow dari pengulangan **for** dalam bentuk diagram



Sebagai contoh mari kita berikan selamat kepada nama-nama siswa yang diberikan pada contoh kalimat kondisional.

```
siswa = ["Alzimna", "Badril", "Umam"]
for nama in siswa :
    print("Selamat "+nama+" !")
```

Selamat Alzimna !
Selamat Badril !
Selamat Umam !

Objek *iterable* yang sering digunakan dalam pengulangan `for` adalah `range()` yang menghasilkan list angka-angka yang berada di dalam rentang yang kita berikan seperti contoh `range(1,11)` membentuk list `[1,2,3,...,10]`. Perhatikan bahwa `range(a,b)` membentuk list dari `a` sampai `b-1`.

```
for i in range(1,11) :
    print(i**2,end = " ")
```

1 4 9 16 25 36 49 64 81 100

1.9 Module

Module merupakan koleksi script dari objek-objek yang dituliskan dengan bahasa Python yang membantu kita untuk menghemat waktu dan tidak harus menuliskan kode yang sangat panjang berulang kali. Module mempunyai karakteristik yang sama dengan library pada bahasa pemrograman R yang harus diinstall terlebih dahulu untuk dapat digunakan. Tidak semua module harus diinstall, kita bisa mengetikkan `help('modules')` untuk mengetahui module-module yang terinstall secara default pada Jupyter Notebook.

Setelah module yang kita inginkan dipastikan terinstall, maka untuk mengakses module ini kedalam lembar kerja kita adalah dengan mengetikkan `import` sebelum module ini digunakan. Sebagai contoh, akan digunakan nilai π untuk menghitung luas lingkaran berjari-jari 10. Karena kita tidak mempunyai nilai π eksak maka tidak mungkin kita membuat kode terlebih dahulu untuk menghitung nilai π ini. Akan tetapi nilai eksak dari π telah disimpan di dalam module `math` sehingga kita cukup memanggilnya supaya dapat digunakan.

```
import math
print(math.pi*10*10)
```

314.1592653589793

Perhatikan bahwa untuk memberitahu Python bahwa kita perlu nilai π dari module `math` kita menggunakan sintaks `math.pi`. Pada contoh diatas kita telah memanggil semua fungsi yang ada pada module `math` yang mana hal ini tidaklah efisien sehingga kita cukup memanggil `pi` pada modul ini dengan sintaks berikut.


```
from math import pi
print(pi * 10 * 10)
```

Dengan menggunakan sintaks ini kita dapat langsung menggunakan nilai π dengan nama pi tanpa notasi titik seperti sebelumnya. Dalam beberapa kasus notasi titik pada contoh pertama lebih nyaman digunakan karena module yang kita gunakan untuk memanggil objek tersebut terlihat jelas.

Telah dijelaskan sebelumnya bahwa terdapat banyak sekali module yang tersedia pada bahasa pemrograman Python. Kita dapat menemukan informasi terkait module-module yang dapat digunakan pada link <https://docs.python.org/3/library/> .

BAB II

NumPy, Scipy, dan Pandas

Pada bab sebelumnya di bagian akhir kita belajar mengenai module dimana hal ini sangat membantu kita untuk mengefisienkan waktu tanpa harus membuat objek, fungsi, metode dari awal. Ketika kita membuat banyak fungsi untuk menyelesaikan suatu pekerjaan akan lebih mudah jika kita menyimpannya di dalam sebuah module atau bisa juga di dalam package/library. Pada bab ini kita akan fokus untuk membahas package-package yang sangat powerful untuk melakukan analisis statistika yaitu NumPy, Scipy, dan pandas. Untuk module lain pembaca bisa langsung melihat pada website <https://pypi.org/>.

2.1 Numerical Python – NumPy

Salah satu package yang sangat powerful dan sering digunakan oleh data analis adalah NumPy. Nama ini adalah singkatan dari “Numerical Python” dan digunakan sangat luas sebagai komponen penting untuk module lain, secara khusus untuk analisis data dan komputasi. Package ini memberikan fungsionalitas untuk array multidimensi yang tidak dapat dikerjakan hanya dengan menggunakan list biasa.

Untuk melakukan analisis data yang mempunyai dimensi yang sangat besar tentu saja kita tidak bisa melakukannya dengan hanya menggunakan list. Salah satu cara untuk melakukan penghitungan pada data yang mempunyai dimensi yang sangat besar adalah menggunakan vektor dan matriks. Pada bahasa pemrograman Python kita dapat mendefinisikan array. Array adalah tipe data yang digunakan untuk menyimpan nilai-nilai di dalam satu variabel dan setiap itemnya mempunyai indeks. Hampir sama dengan list, namun array hanya bisa menyimpan item dengan tipe data yang sama seperti integer dengan sesama integer.

Array disini dapat menyimpan data dengan bentuk vektor (array $1 \times n$) atau matriks (array $m \times n$) dan masing-masing dapat dioperasikan dengan menggunakan penjumlahan, pengurangan, atau perkalian.

Install NumPy

Mengingat NumPy bukan merupakan package yang otomatis terinstall ketika kita menginstall Python maka kita harus menginstall NumPy pada komputer kita terlebih dahulu. Caranya cukup mudah yaitu

1. Buka Command Prompt dengan cara **Klik logo Windows > Ketikkan cmd > Pilih Command Prompt**
2. Ketikkan perintah berikut `pip3 install numpy`, pastikan kita mempunyai koneksi internet agar komputer kita dapat mendownload file instalasi NumPy. Setelah proses selesai maka akan muncul status seperti berikut

```
C:\Windows\system32\cmd.exe
Microsoft Windows [Version 10.0.19045.4170]
(c) Microsoft Corporation. All rights reserved.

C:\Users\Dell>pip3 install numpy
Collecting numpy
  Downloading numpy-1.26.4-cp312-cp312-win_amd64.whl.metadata (61 kB)
----- 61.0/61.0 kB 22.7 kB/s eta 0:00:00
  Downloading numpy-1.26.4-cp312-cp312-win_amd64.whl (15.5 MB)
----- 15.5/15.5 MB 1.3 MB/s eta 0:00:00
Installing collected packages: numpy
Successfully installed numpy-1.26.4
```

Matriks dan Vektor

Sebuah matriks adalah array yang berbentuk persegi panjang dengan m baris dan n kolom. Kita sebut matriks ini berukuran $m \times n$. Dalam hal $m = 1$ kita mempunyai vektor kolom dan ketika $n = 1$ kita mempunyai vektor baris. Kita dapat menamakan entri entri pada matriks A sesuai letaknya dengan menggunakan symbol a_{ij} yang berarti elemen pada baris ke i dan kolom ke j .

$$A = \begin{bmatrix} a_{11} & \cdots & a_{1n} \\ \vdots & \ddots & \vdots \\ a_{m1} & \cdots & a_{mn} \end{bmatrix}$$

Selanjutnya akan kita lihat perbedaan antara list dan array. Kita tahu bahwa list dapat digunakan untuk menyimpan item dan juga dapat dibentuk sebagai vektor-vektor baris atau kolom. Perhatikan contoh dua list berikut yang menyatakan banyaknya apel tipe 0 sampai 4 yang dibeli oleh Karjo masing masing dari toko A dan toko B

```
apel_a = [0,1,1,2,3]
apel_b = [5,8,13,21,34]
```

Jika kita ingin mengetahui jumlah dari apel dari masing-masing tipe maka kita harus menjumlahkan nilai yang ada pada kedua list tersebut sesuai tipenya. Kita lihat apakah operasi penjumlahan dapat dilakukan pada kedua list ini

```
apel_a = [0,1,1,2,3]
apel_b = [5,8,13,21,34]
apel_a+apel_b

[0, 1, 1, 2, 3, 5, 8, 13, 21, 34]
```

Terlihat bahwa hasil ini bukanlah yang kita harapkan. Meskipun begitu list merupakan objek di dalam Python yang sangat powerful juga untuk beberapa pekerjaan. Namun untuk melakukan penghitungan numerik yang membutuhkan operasi yang seragam untuk setiap item kita memerlukan array sebagai *container*.

N-Dimensional array

Numpy memungkinkan kita untuk melakukan operasi layaknya penjumlahan matriks. Dengan menggunakan module ini kita dapat mendefinisikan array n dimensi. Sebuah array n dimensi merupakan array multidimensional dimana elemen-elemennya berasal dari tipe yang sama. Pada Python sebuah array n dimensi mempunyai tipe `ndarray`. Bilangan pada `ndarray` menandakan ukurannya, yang berarti container ini terdiri dari n bilangan.

Kita dapat menganggap array merupakan penguatan dari list mengingat kita bisa membentuk array dari list yang kita punya. Kembali menggunakan contoh sebelumnya kita definisikan masing-masing list menjadi array dengan sintaks berikut

```
import numpy as np #Module numpy biasa diberikan nama np
array_a = np.array(apel_a)
array_b = np.array(apel_b)
```

ada sintaks diatas kita mengimpor package NumPy dan memberikan namanya dengan `np` untuk keperluan penyederhanaan. Kita dapat mengecek apakah list kita sudah berupa array dengan melihat tipe datanya, apabila sudah bertipe `numpy.ndarray` maka list tersebut sudah berbentuk array.

```
type(array_a)
numpy.ndarray
```

Selanjutnya kita akan lakukan penjumlahan pada dua array ini.

```
hasil = array_a+array_b
hasil
array([ 5,  9, 14, 23, 37])
```

Kali ini Python telah menjumlahkan kedua array sesuai yang kita inginkan. Ada beberapa operasi yang dapat kita lakukan terhadap array-array ini seperti yang kita tahu pada aljabar linear, apabila pembaca lupa akan operasi-operasi ini disarankan untuk mereview kembali materi mengenai matriks pada sekolah menengah.

Operator	Operasi	Contoh
+	Penjumlahan Vektor	<code>array_a+array_b</code> #hasilnya <code>[5,9,14,23,37]</code>
-	Pengurangan Vektor	<code>array_a-array_b</code> #hasilnya <code>[-5,-7,-12,-19,-31]</code>
*	Perkalian vektor (antar elemen)	<code>array_a*array_b</code> #hasilnya <code>[0,8,13,42,102]</code>
<code>np.dot</code>	Perkalian dot product	<code>np.dot(array_a,array_b)</code> #hasilnya <code>165</code>

Matriks N-dimensional

Numpy juga dapat mendefinisikan objek berupa matriks dengan menggunakan sintaks `np.matrix`. Mari kita lihat contoh pendefinisianya

```
M1 = np.matrix([[1,0],[0,1]])
M2 = np.matrix([[2,3],[4,5]])
type(M1)

numpy.matrix
```

Perhatikan bahwa tipe data dari matriks pada NumPy adalah `numpy.matrix`. Perhatikan bahwa untuk mendefinisikan matriks, kita menyimpan nilai pada baris-baris matriks menggunakan list lalu dikumpulkan menjadi satu list untuk diubah menjadi matriks. Seperti yang kita tahu pada teori aljabar linear untuk dapat melakukan operasi matriks kita harus memperhatikan ukurannya terlebih dahulu dan hal ini tidak akan dibahas pada buku ini. Sintaks diatas menghasilkan matriks *M1* dan *M2* sebagai berikut

$$M1 = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \text{ dan } M2 = \begin{bmatrix} 2 & 3 \\ 4 & 5 \end{bmatrix}$$

Berikut ini diberikan operasi-operasi yang dapat digunakan dengan menggunakan matriks.

Operator/Fungsi/Method	Operasi	Contoh
+	Penjumlahan Matriks	<code>M1+M2</code> [16]: <code>matrix([[3, 3], [4, 6]])</code>
-	Pengurangan Matriks	<code>M1-M2</code> [17]: <code>matrix([[-1, -3], [-4, -4]])</code>
*	Perkalian Matriks biasa	<code>M1*M2</code> [18]: <code>matrix([[2, 3], [4, 5]])</code>
<code>transpose()</code>	Transpose Matriks	<code>M2.transpose()</code> [19]: <code>matrix([[2, 4], [3, 5]])</code>
<code>shape()</code>	Melihat ukuran matriks	<code>M1.shape</code> [21]: (2, 2)

dtype	Melihat tipe data yang ada di dalam matriks	<pre>M1.dtype</pre> <pre>[23]:</pre> <pre>dtype('int32')</pre>
<code>np.zeros((a,b))</code>	Membuat matriks dengan entri nol semua berukuran $a \times b$	<pre>np.zeros((2,3))</pre> <pre>[24]:</pre> <pre>array([[0., 0., 0.],</pre> <pre> [0., 0., 0.]])</pre>
<code>np.ones((a,b))</code>	Membuat matriks dengan entri 1 semua berukuran $a \times b$	<pre>np.ones((2,3))</pre> <pre>[25]:</pre> <pre>array([[1., 1., 1.],</pre> <pre> [1., 1., 1.]])</pre>
<code>np.arange(n)</code>	Membuat array yang berisikan $0, 1, \dots, n$	<pre>np.arange(4)</pre> <pre>[27]:</pre> <pre>array([0, 1, 2, 3])</pre>

Array yang diproduksi oleh NumPy juga mempunyai indeks seperti list sehingga kita bisa mengakses elemen sesuai posisi yang kita minta dan memotong arraynya. Berikut ini diberikan contoh cara mengakses elemen array pada rentang tertentu.

```
a = np.arange(12)
```

```
print(a[1:4])
```

```
print(a[0:10:2])
```

```
[1 2 3]
```

```
[0 2 4 6 8]
```

Perhatikan bahwa pada contoh kedua nilai 2 menunjukkan *step* yaitu kita mengambil elemen pada array a setiap 2 langkah. Kita juga bisa melakukan slicing serupa pada matriks seperti berikut.

```
#Membuat matriks berukuran 2 x 4
```

```
b = np.array([[2,4,5,6],[1,3,5,7]])
```

```
print(b[1,:])
```

```
[1 3 5 7]
```

Sintaks diatas mengambil baris dengan indeks 1 dan kolom yang diambil adalah semuanya. Kita juga dapat mengambil kolomnya saja dengan menukar letak angkanya. Berikut diberikan contoh slicing pada matriks menggunakan daftar nilai siswa sesuai mata pelajaran.

Nama	Fisika	Matematika	Kimia
Alzimna	8	9	7
Badril	4.5	6	3.5
Abu	8.5	10	9
Umam	8	6.5	9.5
Karjo	9	10	7.5

Pertama-tama akan didefinisikan array yang memuat semua nilai dengan baris menyatakan nilai siswa di tiga mata pelajaran.

```
nilai = np.array([[8, 9, 7],
                  [4.5, 6, 3.5],
                  [8.5, 10, 9],
                  [8, 6.5, 9.5],
                  [9, 10, 7.5]])
```

Berikut ini slicing yang dapat dilakukan pada array, bayangkan seperti kita memblok cell pada Excel.

Sintaks	Operasi	Contoh
<code>nilai[i,j]</code>	Mengakses elemen baris ke- <i>i</i> kolom ke- <i>j</i> array nilai (ingat indeksnya mulai dari nol)	<code>nilai[1,2]</code> [35]: 3.5
<code>nilai[a,:], nilai[:,a]</code>	Mengakses elemen baris (kolom) ke a	<code>nilai[1,:]</code> [37]: array([4.5, 6. , 3.5])
<code>nilai[a:b,c:d]</code>	Mengakses elemen baris ke a sampai b-1 dan kolom ke c sampai d-1	<code>nilai[1:4,1:3]</code> [40]: array([[6. , 3.5], [10. , 9.], [6.5, 9.5]])

Statistika Deskriptif

NumPy juga menyediakan fungsi fungsi untuk melakukan operasi pada array terkait statistika deskriptif seperti maximum, minimum, sum, mean, dan standard deviasi. Berikut ini diberikan contoh dari penggunaan fungsi pada NumPy untuk keperluan statistika deskriptif

```
#Untuk mencari nilai maksimum pada kolom ke-0  
nilai[:, 0].max()
```

9

```
#Untuk mencari nilai minimum pada kolom ke-0  
nilai[:, 0].min()
```

4.5

```
#Menjumlahkan nilai-nilai pada kolom ke-0  
nilai[:, 0].sum()
```

38

```
#Menghitung rata-rata kolom ke 0  
nilai[:, 0].mean()
```

7.6

```
#Menghitung standard deviasi kolom ke 0  
nilai[:, 0].std()
```

1.594

Perhatikan bahwa untuk mencari nilai statistika deskriptif pada kolom atau baris tertentu maka kita harus melakukan slicing terlebih dahulu lalu dicari nilainya pada hasil slicing. Lebih lanjut kita dapat memperoleh nilai statistika deskriptif dari keseluruhan matriks yaitu memanggil fungsinya tanpa melakukan slicing terlebih dahulu.

Untuk mencari rata-rata nilai setiap kolom/baris kita tidak harus melakukannya satu persatu, kita bisa mengatur axis pada fungsi mean. Perhatikan bahwa jika axis = 0 maka akan dicari rata-rata per kolom sedangkan untuk axis = 1 akan dicari rata-rata per baris.

```
print(nilai.mean(axis = 0))  
print(nilai.mean(axis = 1))
```

[7.6 8.3 7.3]

[8. 4.66666667 9.16666667 8. 8.83333333]

Dari hasil diatas kita bisa melihat bahwa mata pelajaran dengan rata-rata tertinggi adalah matematika yaitu sebesar 8.3 dan siswa dengan rata-rata nilai terbesar adalah Abu.

Selain penghitungan statistika deskriptif diatas kita juga bisa mengambil nilai dari suatu array yang muncul hanya satu kali atau dengan kata lain kita mencari nilai apa saja yang ada pada suatu array. Selain itu kita juga bisa mendapatkan berapa kali masing-masing nilai tersebut muncul pada array. Perhatikan contoh berikut


```

m = [3,3,4,5,4,3,10,9,12,9]
np.unique(m)
array([ 3,  4,  5,  9, 10, 12])

np.unique(m, return_counts = True)

(array([ 3,  4,  5,  9, 10, 12]), array([3, 2, 1, 2, 1, 1], dtype=int64))

```

Pada sintaks diatas apabila return_counts bernilai True maka Python akan menampilkan setiap nilai unik pada array m dan juga banyaknya kemunculan dari masing-masing nilai diurutkan sesuai letaknya sebagai contoh 3 muncul sebanyak 3 kali dan 9 muncul sebanyak 1 kali. Perlu diketahui bahwa kita tidak hanya bisa mencari nilai yang unik pada suatu array n dimensi namun nilai unik pada array $m \times n$ juga bisa dicari dengan sintaks serupa.

Selanjutnya setelah membahas mengenai array, kita akan mencoba untuk mengembangkan berbagai macam tipe operasi yang dapat kita lakukan dengan array. Package SciPy memuat banyak sekali operasi dan algoritma yang berkaitan dengan array.

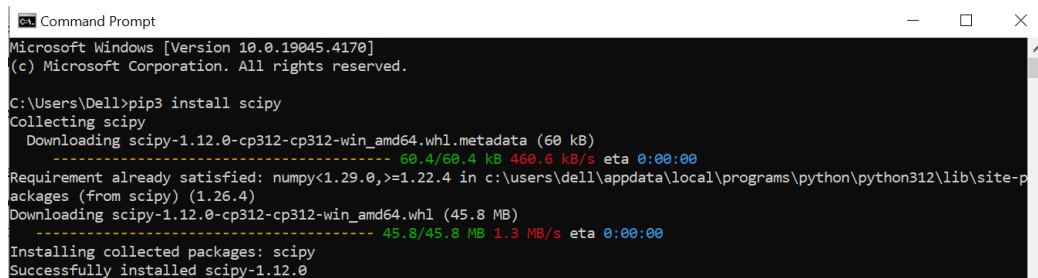
2.2 Scientific Python - SciPy

Penggunaan Python untuk melakukan komputasi yang semakin banyak diminati membuat kita membutuhkan bantuan dari library yang tersedia pada Python. Package SciPy memfasilitasi kita untuk mengimplementasikan algoritma dan sub module yang membuat kita bisa melakukan penghitungan untuk statistic, menggunakan fungsi special, dan optimisasi. Pada buku ini hanya akan dibahas setetes dari banyaknya kegunaan SciPy, berikut ini diberikan sub module dari SciPy dan bidangnya.

Nama Submodule	Kegunaan
cluster	Untuk algoritma clustering
constants	Memuat konstanta Matematika dan Fisika
fftpack	Untuk Fast Fourier Transform
integrate	Menyelesaikan integrase dan persamaan diferensial biasa
interpolate	Untuk interpolasi dan smoothing splines
io	Untuk input dan output
linalg	Untuk bidang aljabar linear
ndimage	Untuk pemrosesan gambar N -dimensional
odr	Untuk regresi orthogonal
optimize	Untuk optimisasi dan pencarian akar
signal	Untuk pemrosesan sinyal
sparse	Untuk mengatur matriks sparse dan asosiasi
spatial	Untuk data spasial dan algoritmanya
special	Menyediakan implementasi dari fungsi-fungsi special
stats	Untuk analisis statistika termasuk distribusi dan fungsi pdf

Package SciPy sangat bergantung dengan NumPy untuk banyak kegunaan. Biasanya ketika kita membutuhkan SciPy kita harus mengimpor package NumPy terlebih dahulu. Layaknya NumPy SciPy tidak langsung terinstall seketika kita menginstall Jupyter Notebook. Oleh karena itu kita harus melakukan instalasi package SciPy pada command prompt. Langkahnya sama dengan penginstallan NumPy namun hanya berbeda sintaks.

1. Buka Command Prompt dengan cara **Klik logo Windows > Ketikkan cmd > Pilih Command Prompt**
2. Ketikkan perintah berikut `pip3 install scipy`, pastikan kita mempunyai koneksi internet agar komputer kita dapat mendownload file instalasi SciPy Setelah proses selesai maka akan muncul status seperti berikut



```
Command Prompt
Microsoft Windows [Version 10.0.19045.4170]
(c) Microsoft Corporation. All rights reserved.

C:\Users\Dell>pip3 install scipy
Collecting scipy
  Downloading scipy-1.12.0-cp312-cp312-win_amd64.whl.metadata (60 kB)
----- 60.4/60.4 kB 460.6 kB/s eta 0:00:00
Requirement already satisfied: numpy<1.29.0,>=1.22.4 in c:\users\dell\appdata\local\programs\python\python312\lib\site-packages (from scipy) (1.26.4)
  Downloading scipy-1.12.0-cp312-cp312-win_amd64.whl (45.8 MB)
----- 45.8/45.8 MB 1.3 MB/s eta 0:00:00
Installing collected packages: scipy
Successfully installed scipy-1.12.0
```

Beberapa bidang yang menggunakan SciPy untuk menyelesaikannya diantaranya

Aljabar Linear

Telah dijelaskan pada bagian NumPy kita bisa melakukan pendefinisian matriks dan juga melakukan operasi sederhana. Selanjutnya dengan menggunakan SciPy kita bisa melakukan operasi yang lebih rumit daripada yang disediakan oleh NumPy. Sebagai contoh mencari invers matriks, seperti yang kita tahu ini akan sangat berguna untuk menyelesaikan system persamaan linear. Pada contoh berikut akan dicari invers dari matriks

$$X = \begin{pmatrix} 1 & 0 & 0 \\ 10 & 4 & 0 \\ 1 & 5 & 2 \end{pmatrix}$$

Dengan menggunakan submodule linalg diperoleh

```
#Kita import numpy dahulu
import numpy as np

#Kita cukup mengimport submodule linalg
#Gunakan from scipy import * jika ingin mengimpor semua submodule
from scipy import linalg

x = np.array([[1,0,0],[10,4,0],[1,5,2]])
print(linalg.inv(x))

[[ 1.    0.    0. ]
 [-2.5   0.25  0. ]
 [ 5.75 -0.625 0.5 ]]
```

Diperoleh invers dari matriks X adalah

$$X^{-1} = \begin{pmatrix} 1 & 0 & 0 \\ -2.5 & 0.25 & 0 \\ 5.75 & -0.625 & 0.5 \end{pmatrix}$$

Selain mencari invers dari matriks kita juga melakukan berbagai operasi seperti determinan dan mencari nilai eigen. Ketika kita menggunakan fungsi `linalg.eig` maka akan menghasilkan dua output, output pertama adalah nilai eigen dan output kedua adalah vektor eigen yang bersesuaian dan karena kita mengenal nilai eigen dan vektor eigen dapat berupa bilangan kompleks maka jika dipastikan matriks kita hanya mempunyai nilai eigen dan vektor eigen real kita bisa mengambil komponen realnya saja dengan method `real`. kita bisa mengambil kedua nilai tersebut secara simultan dengan sintaks berikut

```
linalg.det(x)

8.0

l,v = linalg.eig(x)
print(l) #Menghasilkan nilai eigen dengan komponen kompleks
print(l.real) #Hanya bagian real setiap nilai eigen
print(v)

[2.+0.j 4.+0.j 1.+0.j]
[2. 4. 1.]
[[ 0.          0.          0.06231097]
 [ 0.          0.37139068 -0.20770325]
 [ 1.          0.92847669  0.97620526]]
```

Statistik

Salah satu module di dalam SciPy yang sering digunakan adalah `stats`, karena module ini memuat alat-alat untuk menyelesaikan permasalahan statistika dan probabilitas. Dengan menggunakan module ini bersama dengan generator bilangan random dari NumPy membuat kita dapat melakukan berbagai macam hal berkaitan dengan distribusi. Sebagai contoh kita bisa melakukan simulasi dengan menggenerate data berukuran n dari distribusi normal dan selanjutnya membuat histogramnya.

```
#Membuat sampel random dari distribusi normal sebanyak 1500
s = np.random.normal(size = 1500)

#Membuat rentang untuk histogramnya dari [-4,-3,...,3,4]
b = np.arange(-4,5)

#Menghitung proporsi banyak bilangan s yang masuk pada setiap rentang
#sebagai tinggi batang masing-masing rentang
h = np.histogram(s,
                 bins = b,
                 density = True)[0]
h

array([0.00133333, 0.03        , 0.124        , 0.364        , 0.32333333,
       0.12933333, 0.02733333, 0.00066667])
```

Kita akan mengeksplor berbagai konsep lebih detail dari kedua bidang diatas kedepannya. Dari berbagai contoh diatas, kita melihat array, matriks, dan vektor hanya bisa menyimpan data dengan tipe yang sama. Namun di kehidupan di luar sana, data mempunyai bentuk yang beragam. Misalkan saja di suatu kelas perkuliahan kita bisa mengambil data nama mahasiswa, jenis kelamin, tinggi badan, berat badan, dan asal daerahnya. Terlihat bahwa data pada kehidupan nyata tidak hanya memuat data numerik saja namun juga terdapat data yang memuat objek bertipe string. NumPy dan SciPy telah melakukan pekerjaan mulia untuk melakukan penghitungan pada data numerik. Akan tetapi kita memerlukan module lain untuk dapat berurusan dengan data yang berisikan objek-objek yang berbeda tipe. Disinilah saat yang tepat untuk package Pandas bersinar.

2.3 Panel Data = pandas

Pandas merupakan project yang dikerjakan oleh Wes McKinney pada 2008 dengan tujuan untuk membuat Python menjadi alat untuk komputasi statistika. Dewasa ini, Pandas digunakan oleh berbagai bidang termasuk machine learning dan data science. Package ini merupakan salah satu Package yang powerful karena memudahkan kita untuk bekerja dengan data panel. Data panel seringkali ditemukan pada bidang ekonomi, ilmu sosial atau epidemiologi karena menghasilkan data hasil observasi dengan cross section berbeda terhadap waktu.

Dengan menggunakan bantuan dari Pandas, kita dapat membaca berbagai bentuk data dari berbagai sumber sebagai dataframe yang sangat mudah untuk dimanipulasi. Pandas seringkali digunakan untuk pengindeksan data dengan banyak kolom dan dapat berurusan dengan missing values, encoding, dan lain-lain.

Layaknya kedua kakaknya, pandas tidak langsung terinstall seketika kita menginstall Jupyter Notebook. Oleh karena itu kita harus melakukan instalasi package pandas pada command prompt. Langkahnya sama dengan penginstallan NumPy namun hanya berbeda sintaks.

1. Buka Command Prompt dengan cara **Klik logo Windows > Ketikkan cmd > Pilih Command Prompt**
2. Ketikkan perintah berikut `pip3 install pandas`, pastikan kita mempunyai koneksi internet agar komputer kita dapat mendownload file instalasi pandas Setelah proses selesai maka akan muncul status seperti berikut

```
C:\Users\Dell>pip3 install pandas
Collecting pandas
  Downloading pandas-2.2.1-cp312-cp312-win_amd64.whl.metadata (19 kB)
Requirement already satisfied: numpy<2, >=1.26.0 in c:\users\dell\appdata\local\programs\python\python312\lib\site-packages (from pandas) (1.26.4)
Requirement already satisfied: python-dateutil>=2.8.2 in c:\users\dell\appdata\local\programs\python\python312\lib\site-packages (from pandas) (2.9.0.post0)
Collecting pytz>=2020.1 (from pandas)
  Downloading pytz-2024.1-py2.py3-none-any.whl.metadata (22 kB)
Collecting tzdata>=2022.7 (from pandas)
  Downloading tzdata-2024.1-py2.py3-none-any.whl.metadata (1.4 kB)
Requirement already satisfied: six>=1.5 in c:\users\dell\appdata\local\programs\python\python312\lib\site-packages (from python-dateutil>=2.8.2->pandas) (1.16.0)
Downloading pandas-2.2.1-cp312-cp312-win_amd64.whl (11.5 MB)
----- 11.5/11.5 MB 1.1 MB/s eta 0:00:00
Downloading pytz-2024.1-py2.py3-none-any.whl (585 kB)
----- 585.5/585.5 kB 1.9 MB/s eta 0:00:00
Downloading tzdata-2024.1-py2.py3-none-any.whl (345 kB)
----- 345.4/345.4 kB 1.2 MB/s eta 0:00:00
Installing collected packages: pytz, tzdata, pandas
Successfully installed pandas-2.2.1 pytz-2024.1 tzdata-2024.1
```

Series dan Dataframe

Tipe data paling dasar pada pandas adalah *series* yaitu array 1-dimensi. Koleksi dari banyak series disebut sebagai *dataframe*, secara sederhana dataframe sama dengan sebuah tabel namun dapat dioperasikan dan series merupakan pembentuk kolomnya. Setiap series pada dataframe mempunyai masing-masing tipe data dan tentu saja pandas sangat membutuhkan NumPy dan SciPy. Series dapat dibentuk dari array NumPy seperti berikut

```
import numpy as np
import pandas as pd #Biasanya pandas diberi nama pd

a = np.array([14.75, 18.5, 72.9, 35.7])
s = pd.Series(a)
type(s)

pandas.core.series.Series
```

Terlihat bahwa s merupakan objek bertipe series. Selanjutnya mari kita lihat data sederhana mengenai kepadatan penduduk di DIY.

Kota	Populasi	Luas Wilayah (km persegi)
Yogyakarta	415509	32.50
Bantul	956513	506.85
Kulon progo	442874	586.27
Gunungkidul	774441	1485.36
Sleman	1088109	574.82

Kita dapat memasukkan data ini kedalam Python dengan cara membuat list-list berdasarkan data tiap kolom.

```
nama = ["Yogyakarta", "Bantul", "Kulon progo", "Gunungkidul", "Sleman"]
populasi = [415509, 956513, 442874, 774441, 1088109]
luas = [32.50, 506.85, 586.27, 1485.36, 574.82]
```

Selanjutnya untuk mendefinisikan dataframe dari list-list diatas kita gunakan method `DataFrame` yang ada di dalam pandas dengan menggunakan dictionary sebagai input. Key dari dictionary ini akan menjadi nama kolom dan valuenya adalah nilai yang ada di dalam list. Perhatikan contoh berikut

```
df = pd.DataFrame({
    "Kota" : nama,
    "Populasi" : populasi,
    "Luas" : luas
})
df
```

	Kota	Populasi	Luas
0	Yogyakarta	415509	32.50
1	Bantul	956513	506.85
2	Kulon progo	442874	586.27
3	Gunungkidul	774441	1485.36
4	Sleman	1088109	574.82

Eksplorasi Data dengan pandas

Setelah berhasil membuat dataframe dari data kita maka kita dapat memulai untuk mengeksplorasi apa saja fitur dari data kita. Berikut ini diberikan beberapa method/fungsi dari pandas yang seringkali digunakan untuk keperluan eksplorasi data

- `.head(n)` untuk menampilkan n baris pertama, apabila n tidak diisi maka otomatis akan menampilkan 5 baris pertama.

```
df.head(2)
```

	Kota	Populasi	Luas
0	Yogyakarta	415509	32.50
1	Bantul	956513	506.85

- `.tail(n)` untuk menampilkan n baris terakhir, apabila n tidak diisi maka otomatis akan menampilkan 5 baris terakhir.

```
df.tail(2)
```

	Kota	Populasi	Luas
3	Gunungkidul	774441	1485.36
4	Sleman	1088109	574.82

- `.shape` untuk menampilkan ukuran dari dataframe. Pada contoh di bawah diperoleh dataframe `df` terdiri dari 5 baris dan 3 kolom.

```
df.shape
```

```
(5, 3)
```

Tipe Data pada pandas

Telah disebutkan bahwa pandas bersifat seperti tabel yang dapat kita modifikasi dan dikenakan operasi. Perbedaan paling mendasar dengan matriks $m \times n$ dari NumPy adalah dataframe dapat menyimpan data sesuai dengan tipe data awal. Perhatikan contoh berikut

```
x = np.matrix([[1,2,3],['a','b','c']])
x
matrix([[1, 2, 3],
        ['a', 'b', 'c']], dtype='<U11')
```

```
x = pd.DataFrame({
    'angka' : [1,2,3],
    'huruf' : ['a','b','c']
})

#Kita bisa melihat tipe data dari masing-masing kolom
x.dtypes

angka      int64
huruf      object
dtype: object
```

Perhatikan bahwa pada matriks, semua item pada list angka diubah menjadi string karena kelas dari integer lebih rendah dari string dalam artian integer bisa dianggap sebagai string namun tidak sebaliknya. Sedangkan pada dataframe tipe data pada masing-masing list dipertahankan. Oleh karena itu dataframe merupakan container yang sering digunakan oleh seorang analis data untuk melakukan eksplorasi terlebih ketika datanya memuat berbagai macam tipe data seperti bilangan dan string.

Informasi mengenai dataframe secara lengkap seperti index, tipe data dari masing-masing kolom, dan nilai non-null dapat diketahui dengan menggunakan method `.info()` sebagai berikut

```
df.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5 entries, 0 to 4
Data columns (total 3 columns):
#   Column      Non-Null Count  Dtype
---  ---
0   Kota         5 non-null      object
1   Populasi     5 non-null      int64
2   Luas         5 non-null      float64
dtypes: float64(1), int64(1), object(1)
memory usage: 252.0+ bytes
```

Terkadang kita perlu mengubah tipe data dari suatu kolom seperti kolom populasi yang mempunyai tipe integer dapat diubah menjadi float. Pada kasus ini, kita gunakan fungsi `astype()` seperti berikut

```
df['Populasi'] = df['Populasi'].astype('float')
```

Manipulasi Data dengan pandas

Telah kita ketahui bahwa untuk mendefinisikan dataframe dapat menggunakan dictionary, hal ini menandakan bahwa kita juga dapat melakukan indexing, dan mengakses elemen pada dataframe layaknya indexing dan akses elemen pada dictionary. Untuk mengakses elemen dari dictionary kita memerlukan nama dari keynya, begitu juga untuk dataframe kita dapat mengakses elemen melalui nama kolomnya. Untuk mengakses nama kolom dari dataframe dapat menggunakan method `.columns` seperti berikut

```
df.columns
```

```
Index(['Kota', 'Populasi', 'Luas'], dtype='object')
```

Selanjutnya untuk melihat data pada baris tertentu dari suatu kolom akan lebih sederhana jika kita mengingat kolom dari suatu dataframe berasal dari list sehingga cara mengakses itemnya sama persis dengan list. Misalkan kita ingin mengambil data populasi yang ada di baris ke 2 dan 3 maka dapat menggunakan sintaks berikut

```
df["Populasi"][2:4]
```

```
2    442874
```

```
3    774441
```

```
Name: Populasi, dtype: int64
```

Perhatikan bahwa nama kolom harus dalam bentuk string (diapit tanda petik). Ketika kita mempunyai data pada Excel kita biasanya hanya memerlukan data yang ada di letak tertentu dan kita melakukan pemilihan dengan cara memblok cell yang kita inginkan. Dengan cara yang sama kita juga bisa melakukannya pada dataframe. Yang kita butuhkan adalah nama kolom dan data yang ingin kita ambil terletak pada baris berapa. Sebagai contoh kita ingin mengambil data Kota dan Luas dari baris ke-1 dan ke-2 maka kita bisa melakukannya dengan sintaks berikut

```
df[["Kota", "Luas"]][1:3]
```

	Kota	Luas
1	Bantul	506.85
2	Kulon progo	586.27

Secara default suatu dataframe mempunyai index 0,1,2,... yang ditandai dengan angka disebelah kiri kolom ke-0. Kita dapat membuat sebuah kolom sebagai index, yang nantinya akan membantu mempermudah untuk mengakses elemen sesuai indeksny. Sebagai contoh, kita tertarik untuk melihat dataframe df berdasarkan nama kota maka kita dapat mendefinisikan kolom “Kota” sebagai index dengan menggunakan method `set_index()`.

```
df.set_index("Kota", inplace = True)
```

```
df
```

	Populasi	Luas
Kota		
Yogyakarta	415509	32.50
Bantul	956513	506.85
Kulon progo	442874	586.27
Gunungkidul	774441	1485.36
Sleman	1088109	574.82

Perhatikan bahwa angka yang berada di kiri kolom Kota tergantikan dengan kolom Kota. Argument inplace bernilai True menandakan bahwa kita ingin mengubah “df” secara langsung.

Setelah mengubah indeks dari dataframe menggunakan kolom yang kita inginkan, kita bisa mengakses baris berdasarkan indeksnya dengan menggunakan method `loc()`.

```
df.loc["Bantul"]
```

```
Populasi    956513.00  
Luas         506.85  
Name: Bantul, dtype: float64
```

Kita juga dapat mengambil kolom tertentu dari indeks yang kita inginkan. Misalkan kita hanya ingin mengambil luas wilayah dari kota Bantul maka kita bisa menggunakan sintaks berikut

```
df.loc["Bantul", "Luas"]
```

```
506.85
```

Jadi jika kita ingin mengambil nilai dari suatu baris dan kita hanya tahu kata kuncinya saja maka kita perlu membuat kolom yang memuat kata kunci ini menjadi index lalu dengan menggunakan loc kita bisa mengambil nilai dari baris yang memuat kata kunci yang kita inginkan.

Sejauh ini kita telah melihat berbagai macam perintah yang sangat berguna khususnya untuk melihat isi dari dataframe. Salah satu method yang juga sangat berguna untuk melihat intisari dari dataframe adalah method `describe()`. Method ini akan memberikan nilai-nilai statistika deskriptif **dari kolom-kolom numerik** pada dataframe kita. Ini akan sangat membantu kita karena tidak harus mencari nilai-nilai ini satu persatu.

```
df.describe()
```

	Populasi	Luas
count	5.000000e+00	5.000000
mean	7.354892e+05	637.160000
std	3.011314e+05	526.429891
min	4.155090e+05	32.500000
25%	4.428740e+05	506.850000
50%	7.744410e+05	574.820000
75%	9.565130e+05	586.270000
max	1.088109e+06	1485.360000

Perhatikan bahwa data berbentuk string (kolom ‘Kota’) tidak muncul dan apabila terdapat data bertipe kategori seperti jenis kelamin, status ekonomi negara maka yang akan ditampilkan hanyalah count, apa saja kategori yang ada, dan kategori dengan frekuensi terbanyak.

Selanjutnya kita juga bisa menambahkan kolom baru pada dataframe dengan sangat mudah. Yaitu dengan sintaks

```
df['nama kolom baru'] = [listuntukkolombaru]
```

kita juga bisa membuat kolom baru dengan mengoperasikan kolom-kolom yang sudah ada. Misalkan kita ingin melihat tingkat kepadatan penduduk tiap kota maka kita bisa menggunakan rumus

$$kepadatan = \frac{populasi}{luas}$$

Sehingga sintaksnya adalah sebagai berikut

```
df["kepadatan"] = df["Populasi"]/df["Luas"]
df
```

	Populasi	Luas	kepadatan
Kota			
Yogyakarta	415509	32.50	12784.892308
Bantul	956513	506.85	1887.171747
Kulon progo	442874	586.27	755.409624
Gunungkidul	774441	1485.36	521.382695
Sleman	1088109	574.82	1892.956056

Ingat bahwa kolom setiap dataframe berbentuk series yang berasal dari array sehingga pembagian antara dua kolom menghasilkan pembagian elementwise (antar elemen) kolom tersebut.

Mengimpor Data

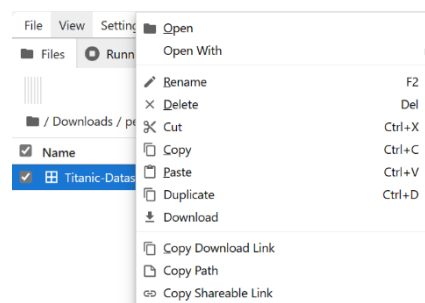
Pada penjelasan diatas kita tahu bahwa kita bisa membentuk dataframe dengan data yang kita masukkan secara manual ke dalam Python. Namun, pada kasus di dunia nyata kita telah mempunyai data dalam berbagai format seperti file excel (xlsx) atau file CSV lalu ingin mengolahnya dengan menggunakan Python. Hal ini sudah difasilitasi oleh pandas sebagai package yang membantu analisis data. Berikut ini diberikan format file dan fungsi untuk mengimpor file tersebut kedalam Python

Sumber	Perintah
Flat file	read_table() read_csv() read_fwf()
Excel	read_excel()

JSON	<code>read_json()</code>
SQL	<code>read_sql_table()</code> <code>read_sql_query()</code> <code>read_sql()</code>
HTML	<code>read_html()</code>

Sebagai contoh akan digunakan data Titanic yang merupakan data paling populer untuk belajar data science dari link <https://www.kaggle.com/datasets/yasserh/titanic-dataset?resource=download> . Data ini memuat informasi pribadi dari para penumpang dan status apakah mereka selamat atau tidak. Setelah mendownload file pada link tersebut, letakkan file pada direktori yang kita inginkan. Misalkan pada folder download. Untuk dapat mengimpor data pada notebook kita harus mendapatkan PATH dari file tersebut yang dapat dilakukan dengan langkah

- Pada halaman Home Jupyter notebook cari folder yang memuat file yang ingin kita impor
- Setelah menemukan file yang kita impor, klik kanan pada nama file tersebut kemudian pilih Copy Path



Setelah mendapatkan Path dari file yang kita inginkan lalu gunakan perintah yang sesuai dengan format file kita. Dalam contoh ini data Titanic berbentuk CSV sehingga akan digunakan `read_csv()` , pastekan Path yang kita copy kedalam fungsi tersebut dan jangan lupa apit dengan tanda petik.

```
import numpy as np
import pandas as pd
titanic = pd.read_csv("Downloads/pelatihan/Titanic-Dataset.csv")
```

Selanjutnya mari kita lihat beberapa baris dari dataset kita. Untuk melakukannya kita gunakan method `head()` .

<code>titanic.head()</code>												
	Id	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500	NaN	S
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.2833	C85	C
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.9250	NaN	S
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803	53.1000	C123	S
4	5	0	3	Allen, Mr. William Henry	male	35.0	0	0	373450	8.0500	NaN	S

Terlihat bahwa data Titanic mempunyai kolom-kolom nomor identitas, status, nama, jenis kelamin dari penumpang. Untuk melihat ukuran dari dataframe kita gunakan method `shape`.

```
titanic.shape  
  
(891, 12)
```

Diperoleh data Titanic terdiri dari 891 baris dan 12 kolom. Kita juga bisa mengubah nama dari kolom yang kita inginkan dengan menggunakan dictionary.

```
namakedua = {'PassengerId' : 'Id'}  
titanic.rename(columns = namakedua,inplace = True)
```

Selanjutnya mari kita lihat nama-nama kolom setelah kita mengubah nama kolom PassengerId dengan menggunakan method `columns()`.

```
titanic.columns  
  
Index(['Id', 'Survived', 'Pclass', 'Name', 'Sex', 'Age', 'SibSp', 'Parch',  
       'Ticket', 'Fare', 'Cabin', 'Embarked'],  
      dtype='object')
```

Kita juga bisa melihat ringkasan dari data yang kita punya menggunakan method `info()`.

```
titanic.info()  
  
<class 'pandas.core.frame.DataFrame'>  
RangeIndex: 891 entries, 0 to 890  
Data columns (total 12 columns):  
#   Column      Non-Null Count  Dtype  
---  ---  
0   Id           891 non-null    int64  
1   Survived     891 non-null    int64  
2   Pclass       891 non-null    int64  
3   Name         891 non-null    object  
4   Sex          891 non-null    object  
5   Age          714 non-null    float64  
6   SibSp        891 non-null    int64  
7   Parch        891 non-null    int64  
8   Ticket       891 non-null    object  
9   Fare         891 non-null    float64  
10  Cabin        204 non-null    object  
11  Embarked     889 non-null    object  
dtypes: float64(2), int64(5), object(5)  
memory usage: 83.7+ KB
```

Terlihat bahwa kolom Id mempunyai tipe integer sedangkan kita tahu bahwa nomor identitas tidak mempunyai informasi penting sebagai data numerik sehingga kita bisa mengubahnya menjadi tipe string dengan sintaks berikut

```
titanic['Id'] = titanic['Id'].astype(str)
```

Melakukan Filtering pada Data

Misalkan kita ingin melihat penumpang mana saja yang selamat dari insiden Titanic, jika menggunakan Excel maka kita langsung terfikir untuk melakukan filter pada kolom Survived lalu memilih status 1 untuk ditampilkan pada layar. Begitu juga dengan Python, kita bisa melakukan filtering pada semua penumpang yang selamat yaitu semua penumpang dengan kolom Survived bernilai 1 dengan sintaks berikut

```
titanic[titanic["Survived"]==1].head()
```

	Id	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.2833	C85	C
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.9250	NaN	S
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803	53.1000	C123	S
8	9	1	3	Johnson, Mrs. Oscar W (Elisabeth Vilhelmina Berg)	female	27.0	0	2	347742	11.1333	NaN	S
9	10	1	2	Nasser, Mrs. Nicholas (Adele Achem)	female	14.0	1	0	237736	30.0708	NaN	C

Terlihat bahwa nilai pada kolom Survived bernilai 1 semua. Perlu diingat bahwa sintaks ini tidak akan menghapus dataframe Titanic awal karena kita hanya melakukan filtering. Contoh lain adalah jika kita ingin menampilkan tabel yang sama dengan gambar diatas namun hanya membutuhkan nama, jenis kelamin, dan umur maka kita bisa melakukan kombinasi dengan cara melakukan slicing terhadap kolom nama, jenis kelamin, dan umur terlebih dahulu lalu dilakukan filtering seperti langkah diatas.

```
titanic[["Name", "Sex", "Age"]][titanic["Survived"]==1].head()
```

	Name	Sex	Age
1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0
2	Heikkinen, Miss. Laina	female	26.0
3	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0
8	Johnson, Mrs. Oscar W (Elisabeth Vilhelmina Berg)	female	27.0
9	Nasser, Mrs. Nicholas (Adele Achem)	female	14.0

Mengelompokkan Data

Ketika kita mempunyai sebuah data yang memiliki kolom bertipe kategori (bayangkan seperti kelompok pria dan wanita, selamat dan tidak, besar dan kecil) biasanya kita ingin membandingkan antara kategori tersebut. Misalkan dalam suatu kelas terdapat siswa pria dan wanita dan kita ingin melihat rata-rata tinggi badan dari masing-masing kelompok, atau melihat banyaknya anggota masing-masing kelompok, pada contoh data Titanic kita melihat rata-rata umur penumpang yang Survived dan tidak.

Hal tersebut diatas dapat dilakukan dengan cara mengelompokkan data menggunakan pandas lalu melakukan agregasi pada dataframe titanic. Fungsi yang dapat digunakan untuk mengelompokkan data berdasarkan kriteria tertentu adalah `groupby()` dan untuk melakukan agregasi menggunakan method `.agg()`.

Kita akan melihat bagaimana cara mengetahui banyaknya penumpang yang selamat dan tidak, ini dapat dilakukan dengan melakukan pengelompokan berdasarkan kolom `Survived`.

```
titanic_grouped = titanic.groupby("Survived")
titanic_grouped.size()
```

```
Survived
0      549
1      342
dtype: int64
```

Diperoleh sebanyak 549 penumpang tidak selamat dan 342 penumpang selamat (betapa tragisnya insiden ini hingga dibuatkan film dokumenter). Selanjutnya kita ingin melihat rata-rata umur dari masing-masing kategori. Kita dapat melakukannya dengan cara melakukan slicing terlebih dahulu pada dataframe `titanic_grouped` dengan mengambil kolom “Age” saja lalu menghitung meannya.

```
titanic_grouped["Age"].mean()
```

```
Survived
0      30.626179
1      28.343690
Name: Age, dtype: float64
```

Dari hasil diatas didapat rata-rata usia penumpang yang tidak selamat adalah 30 dan rata-rata usia penumpang selamat adalah 28 tahun.

Selain menggunakan method `mean` kita juga bisa mendapatkan nilai statistika deskriptif lain untuk masing-masing kelompok dengan menggunakan `aggregate`. Misalkan kita ingin melihat rata-rata usia, usia minimum dan maksimum masing-masing kelompok maka dapat menggunakan sintaks

```
titanic_grouped["Age"].agg(["mean", "min", "max"])
```

	mean	min	max
Survived			
0	30.626179	1.00	74.0
1	28.343690	0.42	80.0

Setelah melihat informasi pada data, melakukan eksplorasi, seorang data saintis dituntut untuk dapat mempresentasikan hasilnya kepada orang lain. Untuk membantu menyampaikan pesan mengenai informasi pada data, kita bisa menggunakan diagram atau plot dari data kita. Diagram atau plot ini akan membantu menjelaskan secara sekilas bagaimana behaviour dari data yang kita punya, lalu setelah melakukan eksplorasi dan grouping kita juga bisa menyajikan hasilnya dengan menggunakan diagram.

Untuk keperluan menggambar plot, pandas tidak memainkan cukup peran sebagai garda terdepan. Pandas akan sangat berperan untuk mengolah data dibalik layar yang selanjutnya disajikan dalam bentuk plot/diagram dengan menggunakan module lain seperti matplotlib dan seaborn. Pada pembahasan berikutnya kita akan membahas module yang dapat membantu kita membuat plot dan tentu saja tidak akan terlepas dengan tiga module yang telah di bahas di depan.

BAB III

VISUALISASI DATA

Pada bab-bab sebelumnya kita telah membahas bagaimana mengimpor data yang sudah kita punya ke dalam Python lalu melakukan eksplorasi dan modifikasi terhadap data tersebut. Jika kita diminta menjelaskan informasi dari data yang kita punya kepada orang lain apakah kita hanya akan menampilkan analisis numerik yang kita lakukan selama eksplorasi data ? Akan lebih mudah memberikan gambaran umum dari data kita kepada orang lain untuk menjamin big picture ditangkap oleh audiens tanpa harus memberikan ringkasan berupa angka dari data kita. Oleh karena itu kita membutuhkan visual dari data kita supaya kita dapat menceritakan data ini kepada mereka, selain itu visualisasi data juga dapat membantu kita melakukan eksplorasi data yaitu dalam hal mengetahui insight dari data kita misalkan data pergerakan penjualan barang atau harga dari suatu saham.

3.1 Visualisasi Apa yang Tepat ?

Tidak ada grafik yang salah karena grafik terbentuk dari data. Namun bayangkan jika kita mempunyai data jumlah barang yang terjual dalam satu hari selama satu tahun. Tentu tidak bijak jika kita memberikan gambaran apakah barang yang kita jual mengalami peningkatan atau penurunan dengan menggunakan bar chart (diagram batang) karena nantinya akan ada 365 batang pada grafik kita. Meskipun dapat menggambarkan kenaikan dan penurunan jumlah barang akan tetapi hal tersebut kurang efisien untuk rentang waktu yang sangat panjang. Bandingkan apabila kita menggunakan diagram garis dimana nilai pada sumbu-y adalah banyaknya barang yang terjual selama satu hari, grafik yang terbentuk akan memberikan gambaran yang tepat daripada diagram batang.

Akan lebih mudah untuk kita memilih jenis visualisasi data dengan mempertimbangkan tujuan umum dari suatu jenis grafik. Berikut ini diberikan pertanyaan dan jenis diagram yang dapat membantu kita sesuai tujuan yang kita inginkan.

Pertanyaan	Tujuan	Tipe Data	Jenis Diagram
Bagaimana bentuk distribusi dari data kita ?	Distribusi	1 Kontinu	Histogram, Box plot, Violin plot
Seberapa banyak kemunculan tiap nilai ?	Perbandingan	1 Kategori	Bar chart, Pie chart
Bagaimana hubungan dari suatu bagian data terhadap keseluruhan data ?	Komposisi	1 Categorical	Bar chart, Stacked bar chart, Pie chart

Apakah data kita mempunyai trend ?	Trend	1 Continuous dan 1 Ordinal	Line chart, Area chart
Apakah terdapat korelasi antar variabel	Korelasi	2 Kontinu	Scatterplot, Bubble chart
Seberapa sering suatu nilai tertentu muncul ?	Perbandingan	2 Kategori	Heatmap

Tabel diatas bersifat sebagai pemberi saran apa diagram yang sesuai dengan tujuan yang kita maksud dan bukan merupakan suatu tolok ukur benar dan salah. Berikut ini dijelaskan beberapa kasus yang sering kita temui mengenai tujuan dari interpretasi data.

1. **Distribusi** : Kita tertarik untuk menggunakan diagram yang menunjukkan bagaimana data kita terdistribusi. Diagram yang tepat untuk keperluan ini adalah histogram dan box plot
2. **Hubungan antar variabel** : Salah satu kegunaan statistika adalah menunjukkan hubungan antar variable. Scatter plot sangat cocok dalam hal ini.
3. **Trend** : Terkadang kita ingin mengetahui arah pergerakan dari data kita terhadap waktu. Hal ini dapat dibantu dengan menggunakan line chart atau bar chart.
4. **Perbandingan** : Apabila kita mempunyai dua atau lebih variabel dan ingin membandingkannya maka kita dapat menggunakan bar chart, heatmap, dan pie chart.
5. **Komposisi** : Misalkan kita mempunyai data yang mempunyai beberapa bagian di dalamnya dan kita ingin melihat bagaimana komposisi data ini dilihat dari seberapa besar bagian-bagian di dalamnya. Kita dapat menggunakan pie chart, area chart, atau stacked bar chart.

Kasus-kasus diatas tidak perlu dihafal karena sifatnya hanya memberikan bantuan untuk kita dapat membuat visualisasi data yang tepat sasaran. Sebagai contoh apabila kita mempunyai lebih dari satu variabel dan ingin melihat pergerakan masing-masing variabel terhadap waktu kita dapat menggunakan line chart dengan banyak garis sesuai jumlah variabel. Setelah kita mengetahui inti dari visualisasi data dan memutuskan bagaimana data kita “diceritakan” maka tugas kita selanjutnya adalah mengeksekusinya dengan menggunakan Python. Ada banyak sekali alternatif untuk memvisualisasi data dengan menggunakan Python, kita hanya akan membahas beberapa alat yang sering digunakan seperti Matplotlib, Seaborn, dan Plotly.

3.2 Visualisasi Data dengan Pandas dan Matplotlib

Telah dibahas di bab sebelumnya bahwa package pandas membantu kita untuk mengatur struktur dari data dan analisis data. Salah satu kegunaan lain dari pandas adalah membuat plot dari dataframe yang sudah ada. Untuk melakukannya kita membutuhkan package lain yaitu **Matplotlib**. Seperti package-package yang terdahulu, Matplotlib tidak secara default terinstall dan kita harus melakukan instalasi dengan menggunakan Command Prompt. Dengan cara yang sama kita ketikkan perintah

```
pip3 install matplotlib
```

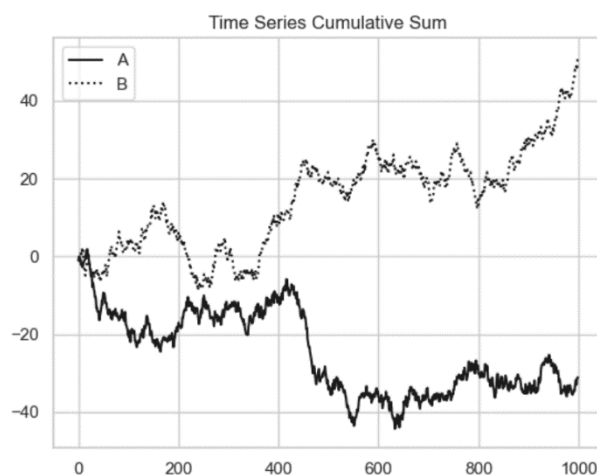
pada Command Prompt dan menunggu proses instalasi package Matplotlib selesai.

Pada bagian ini kita akan memberikan contoh plotting dengan menggunakan pandas. Kita akan membuat data secara random dengan bantuan NumPy, misalkan kita akan membuat data banyaknya barang yang terjual selama satu tahun dari toko A dan toko B dengan rentang 0-1000. Selanjutnya kita gambarkan nilai kumulatif (nilai kumulatif pada waktu ke- t menandakan jumlah barang yang terjual dari waktu ke-0 sampai t) dari masing-masing toko berdasarkan waktunya, kita dapat melakukannya dengan method `cumsum`. Perhatikan contoh berikut

```
import numpy as np
import pandas as pd
timeSeries = np.random.randn(1000, 2)
df = pd.DataFrame(timeSeries, columns=["A", "B"])
cm = df.cumsum()
```

Perhatikan bahwa kita menggenerate nilai pada kolom A dan B secara random sehingga setiap kali program tersebut dieksekusi akan memberikan data yang berbeda. Selanjutnya kita akan membuat visualisasi dari data tersebut. Karena data bersifat runtun waktu maka diagram yang paling tepat untuk menggambarannya adalah line chart.

```
import matplotlib.pyplot as plt
cm.plot(style={'A': 'k-', 'B': 'k:'}, title='Time Series Cumulative Sum')
plt.show()
```



Bisa dilihat bahwa untuk membuat line chart kita tinggal menggunakan method `plot` selanjutnya kita masukkan kriteria bagaimana plot kita digambarkan sebagai *parameter*. Selain line chart kita juga bisa membuat berbagai macam diagram dengan memasukkan parameter `kind = jenisdiagram` sebagai berikut

- Bar plot : 'bar' atau 'barh'
- Histogram : 'hist'
- Boxplot : 'box'
- Area plot : 'area'
- Scatter plot : 'scatter'
- Pie chart : 'pie'

Kita akan menyimpan pembahasan mengenai jenis-jenis diagram diatas dan cara memproduksinya setelah membahas package-package yang sering digunakan untuk visualisasi data dengan Python yaitu Seaborn dan Plotly

3.3 Visualisasi Data dengan Seaborn

Beberapa dari kita mungkin akan berfikir mengetahui visualisasi data dengan matplotlib saja sudah cukup. Namun apabila kita mencoba membuat visualisasi data yang mempunyai lebih dari satu pasang sumbu dan banyak garis atau bar maka hal ini akan terasa kompleks dengan menggunakan matplotlib saja. Michael Waskom memperkenalkan package yang bernama **seaborn** sebagai alternatif untuk menyelesaikan kompleksitas visualisasi data yang tidak dapat diselesaikan dengan matplotlib.

Untuk dapat menggunakan library seaborn pada Jupyter Notebook kita harus melakukan instalasi terlebih dahulu pada komputer kita. Dengan cara yang sama dengan package-package sebelumnya, ketikkan perintah

```
pip3 install seaborn
```

pada jendela Command Prompt dan tunggu instalasi selesai.

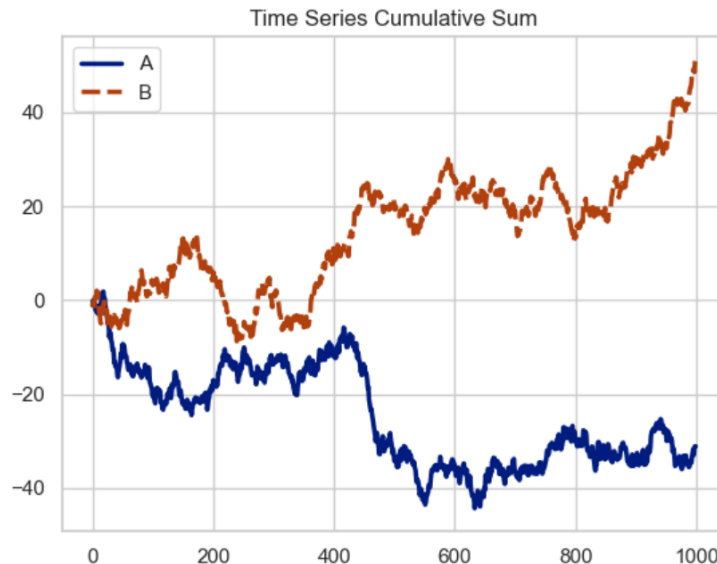
Selanjutnya untuk mengimpor seaborn kedalam lembar kerja kita, kita dapat menggunakan sintaks berikut

```
import seaborn as sns
```

Perhatikan bahwa alias yang sering digunakan untuk seaborn adalah `sns`. Kita dapat membuat plot serupa pada contoh matplotlib dengan menggunakan seaborn. Jika sintaks berikut dieksekusi

```
import seaborn as sns
sns.set_theme(style='whitegrid')
sns.lineplot(data=cm, palette='dark',
             linewidth=2.5).set(title='Time Series Cumulative Sum')

plt.show()
```



Perhatikan bahwa berbeda dengan matplotlib, kita bisa meminta seaborn menggunakan tema yang berbeda-beda dan secara otomatis seaborn akan memberikan warna yang berbeda untuk setiap garis.

Selanjutnya setelah membahas pembuatan plot yang “pasif” dengan matplotlib dan seaborn, kita akan membahas salah satu package yang dapat membuat plot dan memberikan kesempatan kepada user-nya untuk berinteraksi dengan hasil plotnya.

3.4 Visualisasi data dengan Plotly

Salah satu library untuk membuat plot interaktif pada Python adalah *Plotly*, library ini disediakan oleh perusahaan dengan nama yang sama yang berpusat di Quebec, Canada. Selain dapat digunakan pada Python library ini juga tersedia untuk bahasa pemrograman lain diantaranya R, MATLAB, Julia, dan lain-lain.

Kita akan menggunakan modul bernama *Plotly Express* yang menyediakan fungsi-fungsi untuk membuat plot dengan mudah. Plotly Express merupakan bagian dari library Plotly dan merupakan *starting point* yang baik untuk memulai menggunakan Plotly.

Untuk dapat menggunakan library ini kita harus melakukan instalasi pada komputer kita. Pastikan mempunyai koneksi internet lalu buka Command Prompt pada Windows dan ketikkan perintah

```
pip3 install plotly
```

Setelah melakukan instalasi kita akan mencoba membuat plot dengan menggunakan dataframe cumulative sum yang digunakan pada contoh sebelumnya. Perhatikan bahwa pada umumnya Plotly Express diberi nama sebagai “px” untuk menyingkat penulisan sintaks.

Untuk dapat membuat plot seperti pada contoh sebelumnya gunakan sintaks berikut

```
import plotly.express as px
fig = px.line(cm)
fig.show()
```



Perhatikan bahwa ketika cursor diarahkan pada garis pada plot maka akan muncul keterangan variabel, index (sumbu-x), value (sumbu-y) dari titik cursor. Selain itu Plotly secara otomatis menambahkan legend yang terletak di luar plot, dan menambahkan label pada masing-masing sumbu.

Selanjutnya akan diberikan penggunaan ketiga library yang telah disebutkan diatas berdasarkan jenis plot yang kita inginkan.

3.5 Scatter Plot

Sebuah scatter plot bekerja seperti koordinat cartesius yang menampilkan nilai dari dua variabel berbeda sebagai satu titik. Kegunaan utama dari jenis plot ini adalah untuk menunjukkan hubungan dari dua variabel bertipe numerik yang dapat dilihat dari pola titik yang terbentuk pada scatter plot.

Pada bab ini kita akan menggunakan data populasi dan luas area dari beberapa kota yang ada di dunia. Data ini bisa didownload pada link berikut <https://doi.org/10.6084/m9.figshare.14657391.v1>. Kita akan mencoba untuk membuat scatter plot variabel jumlah penduduk dan luas wilayah. Sebelum membuat plot kita akan mencoba melakukan eksplorasi pada data yang akan digunakan mulai dari menghapus nilai NA pada dataframe, selanjutnya mengubah kolom “Population” menjadi numerik. Kita juga menambahkan kolom baru yaitu ‘Population (M)’ dan ‘City Size’ yang berturut-turut menyatakan jumlah populasi dalam jutaan dan kategori kota berdasarkan ukuran populasinya (Small, Medium, Large).

```

#Memasukkan data pada notebook
df = pd.read_csv("Downloads/pelatihan/GLA_World_Cities_2016.csv")

#Menghapus nilai Na pada dataframe
df.dropna(inplace = True)

#Mengubah kolom populasi menjadi numerik
df['Population']=df['Population'].str.replace(',','').astype(float)

def city_size(x):
    if x < 1.5:
        s = 'Small'
    elif 1.5 <= x < 3:
        s = 'Medium'
    elif 3 <= x < 5:
        s = 'Large'
    else:
        s = 'Mega'
    return s

#Membuat kolom populasi dalam jutaan
df['Population (M)'] = df['Population']/1000000

#Membuat kategori ukuran kota berdasarkan Populasi (M)
df['City Size'] = df['Population (M)'].apply(city_size)

#Menampilkan lima data teratas
df.head()

```

	City	Country	Population	Inland area in km2	Density in people per hectare	Dwellings	Density in dwellings per hectare	People per dwelling	Approx city radius km	Main topographical constraint	Constraint	Population (M)	City Size
1	Greater London	England	8663300.0	1,572	55.0	3,454,490	22.0	2.5	23.0	Rivers	4%	8.663300	Mega
2	Inner London	England	3439700.0	319	108.0	1,460,840	46.0	2.4	10.0	Rivers	5%	3.439700	Large
4	Paris	France	2229621.0	105	212.0	1,336,209	127.0	1.7	8.0	Rivers	2%	2.229621	Medium
5	Lyon	France	500715.0	48	105.0	265,599	55.0	1.9	7.0	Rivers	5%	0.500715	Small
6	Berlin	Germany	3469849.0	892	39.0	1,892,000	21.0	1.8	16.0	Rivers	6%	3.469849	Large

- Menggunakan matplotlib

Sintaks berikut menghasilkan plot sebagai berikut, perhatikan bahwa setiap titik yang terbentuk merupakan representasi dari masing-masing kota. Selain itu dengan memberikan nilai pada variabel s pada fungsi plot akan mengubah ukuran dari titik pada scatter plot.

```

#Membuat objek plot dengan ukuran 10x8
fig, ax = plt.subplots(figsize=(10,8))

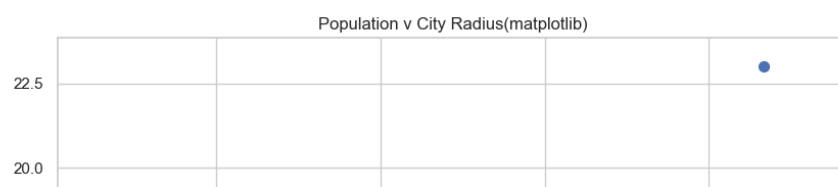
#Membuat scatter plot pada plot ax
ax.scatter(x=df['Populasi'],
          y = df['Luas'],
          s = 100) #semakin besar s semakin besar ukuran titik

#Memberikan label pada sumbu
ax.set_xlabel('Populasi')
ax.set_ylabel('Luas')

#Memberikan judul plot
ax.set_title('Populasi vs Luas (Matplotlib)')

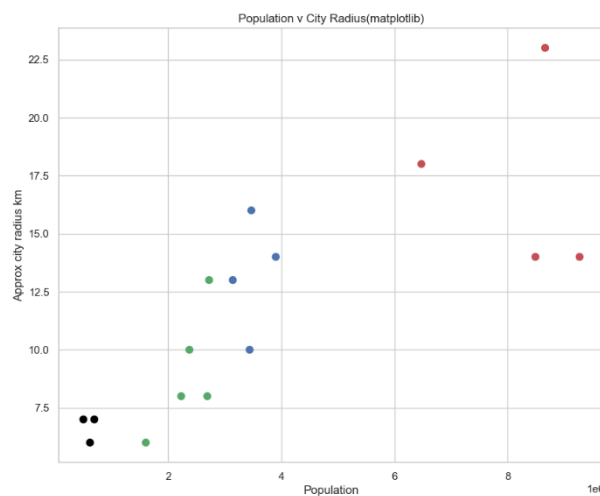
plt.show()

```



Kita juga bisa memberikan warna pada setiap titik berdasarkan variabel “City Size”, misalkan merah pada kategori “Mega”, biru pada “Large”, hijau pada “Medium”, dan hitam untuk “Small”. Dengan mengganti sintaks pada bagian `ax.scatter` dengan sintaks berikut maka akan menghasilkan plot yang kita inginkan.

```
c_map = {'Mega': 'r', 'Large': 'b', 'Medium': 'g', 'Small': 'black'}  
  
colours = df['City Size'].map(c_map)  
  
#Membuat scatter plot pada plot ax  
ax.scatter(x=df['Population'],  
           y=df['Approx city radius km'],  
           s=50, #semakin besar s semakin besar ukuran titik  
           c = colours)  
plt.show()
```



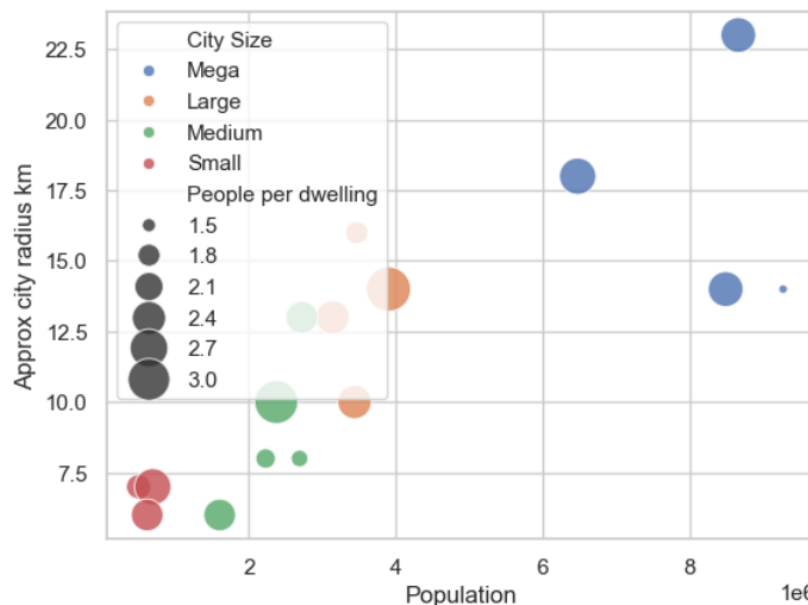
Perhatikan bahwa plot kita belum mempunyai legend sehingga kurang dapat memberikan informasi yang lengkap. Inilah salah satu kekurangan matplotlib, untuk dapat menambahkan legend pada plot kita membutuhkan fungsi di luar library yang menyebabkan penggunaannya tidak efisien. Oleh karena itu akan diberikan contoh pembuatan plot serupa dengan menggunakan library seaborn.

- **Menggunakan Seaborn**

Pada penggunaan seaborn kita bisa membuat ukuran titik pada scatter plot bergantung pada kolom numerik. Sehingga informasi yang bisa ditampilkan akan lebih banyak. Sebagai contoh kita akan membuat ukuran pada plot kita nanti sesuai nilai variabel “People per dwelling” dan warnanya berdasarkan “City Size”.

```
import seaborn as sns

sns.scatterplot(data = df,
                x = "Population",
                y = "Approx city radius km",
                hue = "City Size",
                alpha = 0.8, #Transparansi titik, semakin tinggi semakin
tebal
                size = "People per dwelling",
                sizes = (20, 500))
```



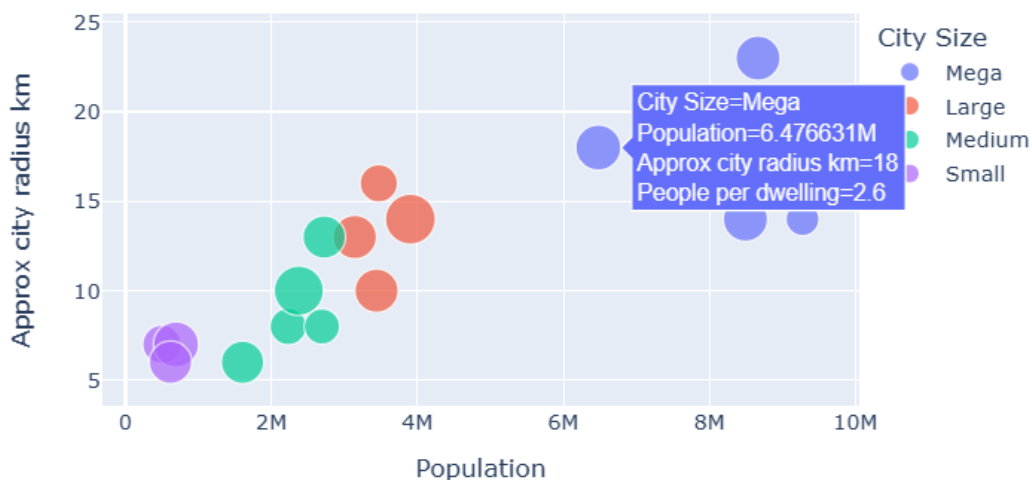
Perhatikan perbedaan sintaks pada penggunaan matplotlib dan seaborn. Pada library seaborn “hue” menyatakan warna dari setiap titik, dan menggunakan kolom “City Size” sebagai ukurannya. Selanjutnya untuk mengatasi titik yang overlap, kita gunakan nilai transparansi 0.8 pada parameter “alpha” dan pada akhirnya kita gunakan kolom “People per dwelling” sebagai ukuran titik dan membuat ukurannya berkisar antara 20 sampai 500 pada parameter “sizes”. Dengan fungsi ini legend akan dibuat secara otomatis oleh seaborn.

- **Menggunakan Plotly**

Selanjutnya akan diberikan contoh penggunaan Plotly untuk membuat scatterplot serupa. Untuk membuat scatterplot pada Plotly kita gunakan method `scatter()` dengan input yang sama dengan contoh-contoh sebelumnya

```
import plotly.express as px
```

```
fig = px.scatter(df, x = "Population",  
                 y = "Approx city radius km",  
                 color = "City Size",  
                 size = "People per dwelling")  
fig.show()
```



Perhatikan bahwa dengan menggunakan Plotly kita bisa mengetahui nilai pada setiap titik dengan mengarahkan kursor pada titik yang kita inginkan.

3.6 Line Chart

Line Chart merupakan diagram yang menghubungkan setiap titik dengan menggunakan garis dari kiri ke kanan untuk menunjukkan perubahan nilai. Variabel yang digunakan pada diagram ini pada sumbu horizontal haruslah kontinu dan seringkali variabel waktu digunakan pada sumbu ini.

Contoh yang digunakan pada [subbab 3.2](#) – [subbab 3.4](#) merupakan ilustrasi dari line chart sehingga tidak akan ditulis ulang pada bagian ini. Sebagai ringkasan berikut adalah fungsi-fungsi pada masing-masing library untuk membuat line chart.

- Matplotlib : Menggunakan method `plot` dari `pyplot`
- Seaborn : Menggunakan method `lineplot` dari `seaborn`
- Plotly : Menggunakan method `line` dari `plotly.express`

3.7 Bar Chart

Pada beberapa situasi kita perlu untuk membandingkan nilai-nilai pada kategori yang berbeda seperti banyaknya jumlah pria dan wanita, rata-rata PDB negara-negara di dunia, dan lain-lain. Kita akan lebih mudah membandingkan kuantitas-kuantitas tersebut dengan meng gambarkannya sebagai panjang suatu garis. Pada bar chart kita menggunakan batang baik horizontal maupun vertikal untuk menggambarkan nilai-nilai dari masing-masing kategori.

Pada bagian ini kita masih akan menggunakan data yang sama yaitu “GLA_World_Cities_2016.csv”. Kita akan mencoba membandingkan total populasi pada masing-masing negara sehingga kita perlu menjumlahkan populasi dari kota-kota dengan negara yang sama.

- **Menggunakan Matplotlib**

Pada [subbab 2.3](#) kita bisa mengelompokkan dataframe berdasarkan nilai pada suatu kolom dengan menggunakan method `groupby`, mula-mula kita akan membuat dataframe dengan mengelompokkan data sesuai kolom “Country” lalu menjumlahkan variabel “Population”. Ini dapat dilakukan dengan menggunakan sintaks berikut

```
pop = df.groupby("Country")["Population"].sum()
pop
```

Population	
Country	
Brazil	8851782.0
England	12103000.0
France	2730336.0
Germany	3469849.0
Japan	11964307.0
Singapore	3902710.0
Spain	5440424.0
United States	11831062.0

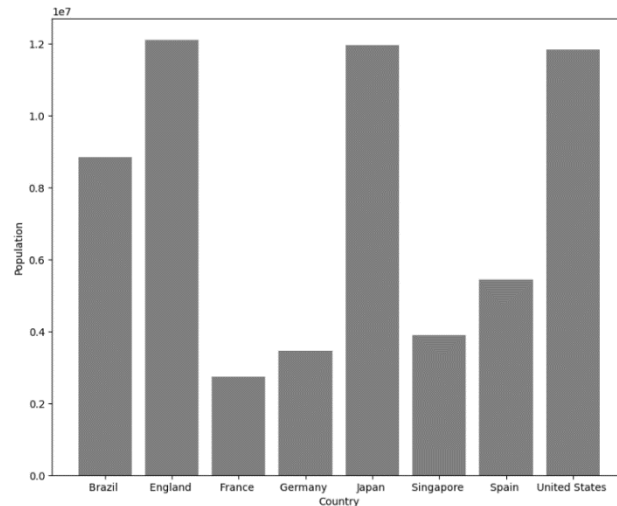
Terlihat bahwa sintaks diatas menghasilkan dataframe baru dengan variabel “Population” pada 8 negara berbeda. Selanjutnya kita akan menggunakan bar chart untuk menggambarkan nilai pada populasi pada masing-masing negara.

```
#Membuat objek plot dengan ukuran 10x8
fig, ax = plt.subplots(figsize=(10,8))

#Membuat bar chart
ax.bar(x = pop.index,
      height = pop["Population"],
      color = "gray")

ax.set_xlabel("Country")
ax.set_ylabel("Population")

plt.show()
```

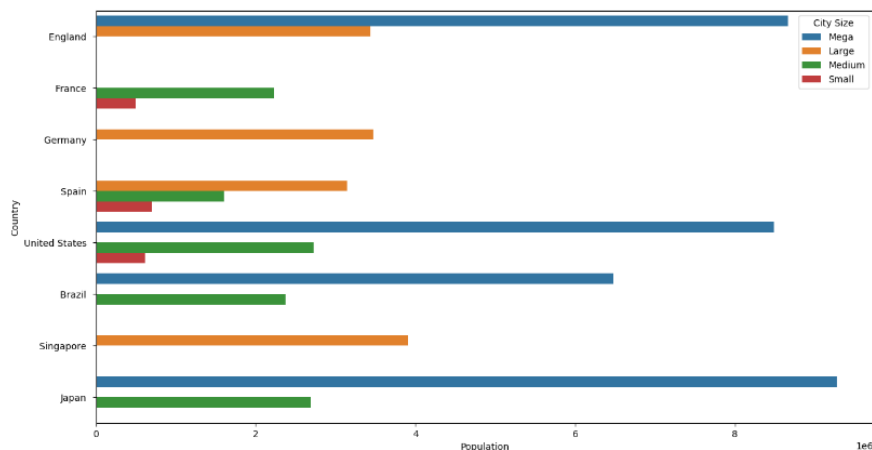


Perhatikan bahwa kita menggunakan nama-nama pada data frame “pop” sebagai sumbu horizontal dengan memanfaatkan index dari “pop” selanjutnya tinggi masing-masing bar diperoleh dari kolom “Population” dari “pop”.

- **Menggunakan Seaborn**

Selanjutnya kita akan menggunakan library Seaborn untuk membuat bar chart dengan data yang sama. Berbeda dengan matplotlib, dengan menggunakan seaborn kita tidak perlu membuat dataframe baru untuk mengetahui total populasi pada masing-masing negara. Pada contoh kali ini kita akan membuat bar plot dengan membedakan bar setiap negara berdasarkan “City Size”.

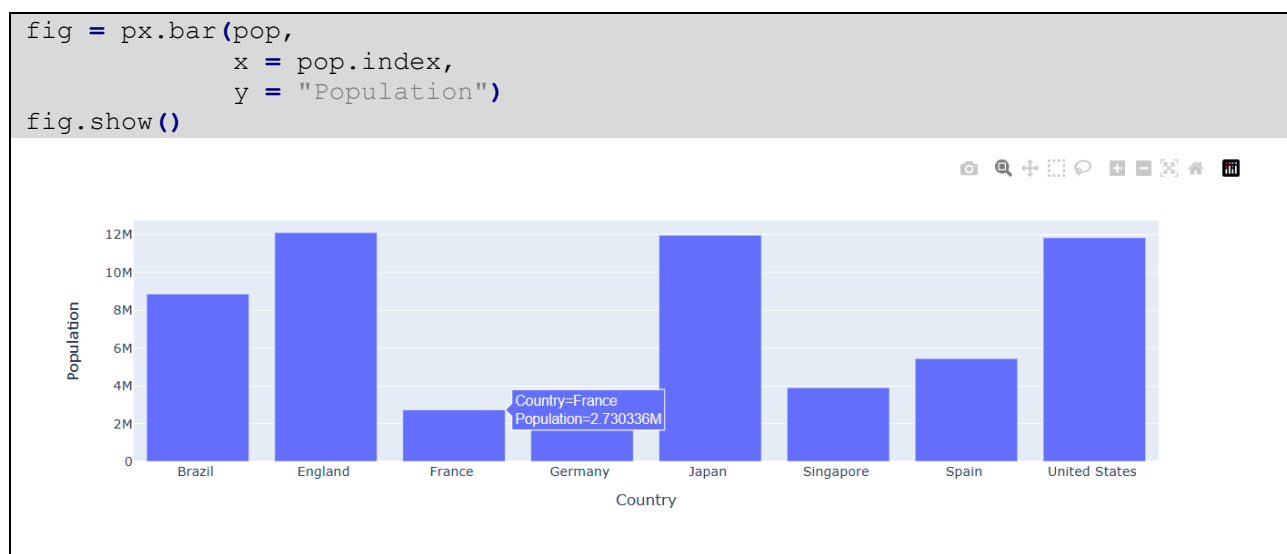
```
fig, ax = plt.subplots(figsize=(15,8))
sns.barplot(data = df,
            x = "Population",
            y = "Country",
            hue = "City Size",
            errorbar = None,
            estimator = sum)
plt.show()
```



Karena Seaborn mendukung penghitungan statistic secara otomatis kita tidak perlu mencari total populasi pada tiap kategori berbeda, kita cukup menggunakan parameter `estimator = sum` dan menggunakan parameter `errorbar = None` untuk menghilangkan interval konfidensi. Perhatikan bahwa kita juga bisa membuat plot serupa yang berorientasi vertikal dengan menukar nilai pada parameter `x` dan `y` pada sintaks diatas.

- **Menggunakan Plotly**

Selanjutnya kita akan memberikan ilustrasi pembuatan bar chart dengan menggunakan Plotly dengan tujuan membuat plot yang interaktif sehingga selain memberikan visualisasi kita juga dapat menunjukkan kuantitas secara bersamaan.



Perhatikan bahwa kita menggunakan dataframe “pop” seperti pada matplotlib selanjutnya membuat sumbu-`x` dari index dataframe dan sumbu-`y` dari kolom “Population”.

3.8 Pie Chart

Salah satu alternatif selain bar chart untuk membandingkan nilai pada masing-masing kategori suatu data adalah *Pie Chart* atau yang kita kenal sebagai diagram lingkaran. Sesuai namanya, pie chart merupakan diagram berbentuk lingkaran yang menggambarkan nilai-nilai pada masing-masing kategori dengan memanfaatkan sudut pada lingkaran sehingga terbentuk daerah-daerah sesuai proporsi/persentase masing-masing kategori.

Perlu diketahui bahwa pie char efektif digunakan untuk jumlah kategori yang tidak terlalu banyak. Hal ini disebabkan karena jika banyaknya kategori terlampau banyak akan membuat visualisasi data yang terbentuk akan tidak bisa ditangkap secara langsung sehingga perlu dipertimbangkan kapan menggunakan pie chart dan bar chart yang telah dijelaskan sebelumnya. Selain itu, jika jarak antara nilai pada masing-masing kategori tidak terlalu jauh maka pie chart juga tidak akan efektif digunakan.

Kita akan menggunakan dataframe “pop” yang menunjukkan total populasi pada negara-negara berbeda dari dataframe “GLA_World_Cities_2016.csv” pada bagian sebelumnya untuk mengilustrasikan pie chart.

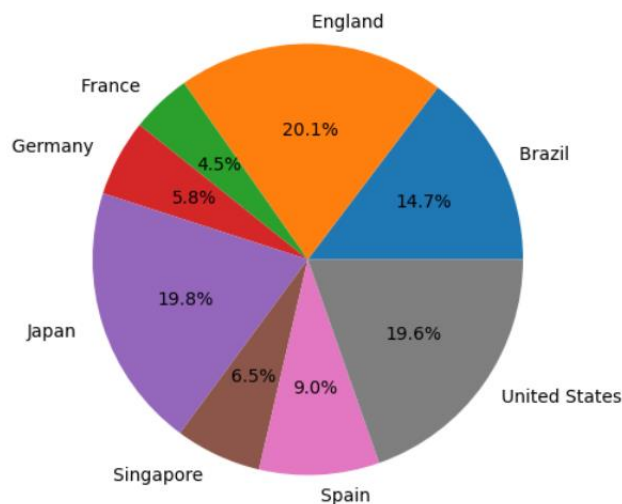
- **Menggunakan Matplotlib**

Untuk membuat pie chart menggunakan matplotlib kita menggunakan method `pie` lalu menentukan nilai dari masing-masing kategori yang diatur menggunakan parameter `labels`. Selanjutnya untuk memunculkan persentase dari masing-masing daerah kita gunakan parameter `autopct`

```
plt.pie(pop["Population"],
        labels=pop.index,
        autopct='%0.1f%%')

#Untuk menjamin diagram yang terbentuk adalah lingkaran
plt.axis("equal")

plt.show()
```



Perhatikan bahwa kita menggunakan nilai pada kolom “Population” pada dataframe “pop” selanjutnya melabeli masing-masing daerah sesuai index dari “pop”. Nilai pada parameter `autopct='%0.1f%%'` akan memunculkan persentase dengan satu angka di belakang koma dan menggunakan symbol %.

Library seaborn tidak memberikan fungsi atau method untuk membuat pie chart sehingga akan diberikan ilustrasi pembuatan pie chart dengan Plotly.

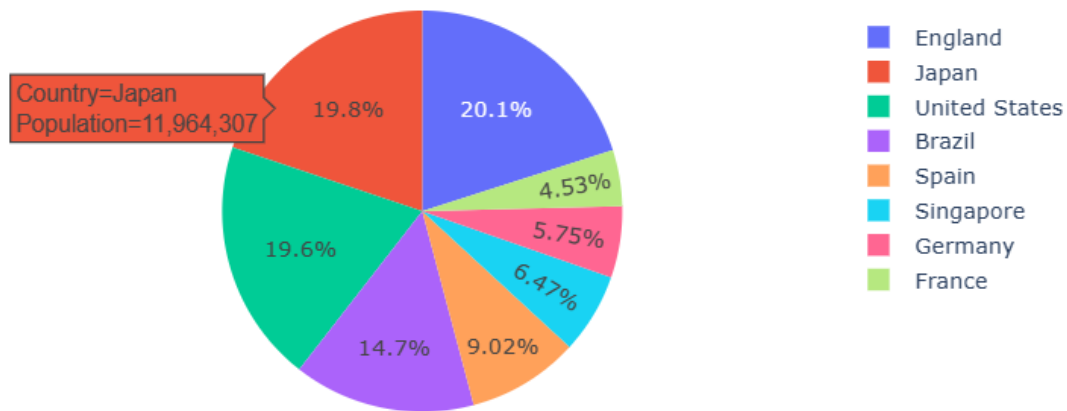
- **Menggunakan Plotly**

Untuk membuat pie chart dengan Plotly kita menggunakan method `pie` selanjutnya kita tentukan data yang akan digunakan dan menentukan nilai yang akan dibandingkan dengan menentukan parameter `values`. Dengan dua input ini kita hanya akan menampilkan persentase saja, namun untuk menampilkan label dari masing-masing kategori kita gunakan parameter `names`.

```
import plotly.express as px

fig = px.pie(pop,
             values = "Population",
             names = pop.index
            )

fig.show()
```



Pada sintaks diatas kita menggunakan data “pop” lalu kolom “Population” untuk menentukan luas daerah masing-masing kategori pada diagram lingkaran. Selanjutnya kita tampilkan label masing-masing daerah sesuai index dari dataframe “pop”.

Sebagai penutup dan pengingat pie chart direkomendasikan hanya jika diagram yang terbentuk tidak menimbulkan kebingungan pada pembaca plot.

3.9 Histogram

Dalam ilmu statistika kita mengenal distribusi dari suatu variabel random seperti distribusi uniform, binomial, normal, dan lain-lain. Terkadang kita tertarik untuk melihat distribusi dari data yang kita miliki apakah mendekati atau bahkan sama dengan plot dari distribusi-distribusi yang kita kenal. Salah satu cara untuk melihat distribusi dari data kita adalah dengan menggunakan histogram.

Histogram mirip dengan bar chart yang telah kita bahas sebelumnya yaitu menggunakan rangkaian persegi panjang untuk merepresentasikan data kita. Pada histogram kita juga menggunakan panjang dari masing-masing persegi panjang untuk menunjukkan frekuensi dari sebuah observasi pada suatu interval, sedangkan pada bar chart setiap persegi panjang mewakili frekuensi dari satu objek sehingga lebar masing-masing persegi panjang tidak mempunyai arti khusus berbeda dengan histogram.

Kita akan mencoba membuat ilustrasi dari histogram dengan menggunakan data pengujian model-model mobil pada tahun 1974 yang dilakukan oleh Rogel-Salazar dan dapat didownload pada link <https://doi.org/10.6084/m9.figshare.3122005.v1>. Kita akan membuat histogram dari variabel “mpg” (miles per gallon). Mula-mula kita impor data kedalam Jupyter notebook dengan sintaks berikut

```
#Membuat path dari file yang kita punya
path = "Downloads/pelatihan/cars.csv"
df = pd.read_csv(path)

#Menampilkan 5 data teratas
df.head()
```

	Car	mpg	cyl	dis	hp	drat	wt	qsec	vs	am	gear	carb
0	Mazda RX4	21.0	6	160.0	110	3.90	2.620	16.46	0	1	4	4
1	Mazda RX4 Wag	21.0	6	160.0	110	3.90	2.875	17.02	0	1	4	4
2	Datsun 710	22.8	4	108.0	93	3.85	2.320	18.61	1	1	4	1
3	Hornet 4 Drive	21.4	6	258.0	110	3.08	3.215	19.44	1	0	3	1
4	Hornet Sportabout	18.7	8	360.0	175	3.15	3.440	17.02	0	0	3	2

Selanjutnya kita akan membuat histogram dengan library-library yang telah kita jelaskan sebelumnya.

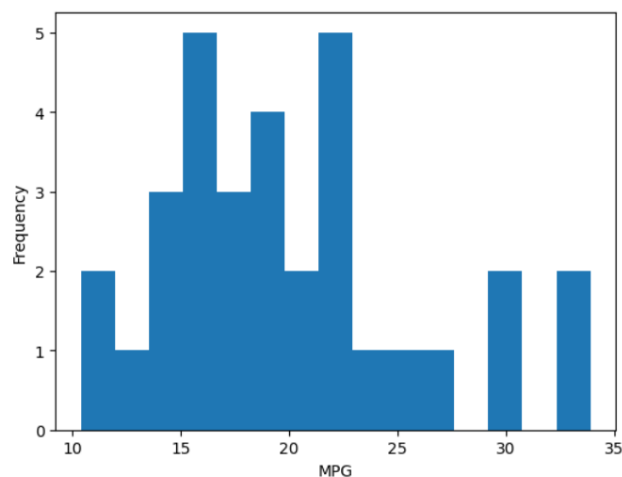
- **Menggunakan Matplotlib**

Untuk membuat histogram dengan library matplotlib kita gunakan method `hist` lalu memasukkan data yang akan kita lihat distribusinya selanjutnya kita tentukan banyaknya rentang yang kita inginkan dengan memasukkan nilai pada parameter `bins`.

```
#Membuat histogram dari kolom "mpg"
# dengan jumlah interval sebanyak 15
plt.hist(df["mpg"],
         bins = 15)

plt.xlabel("MPG")
plt.ylabel("Frequency")

plt.show()
```

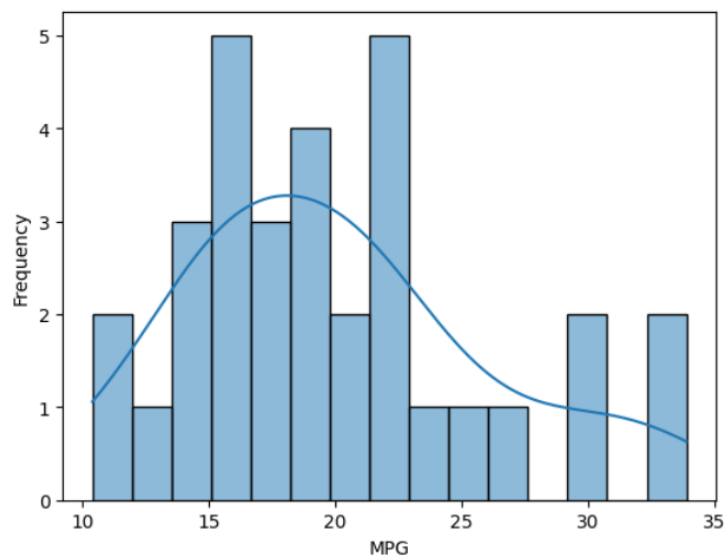


Pada sintaks diatas kita menggunakan kolom “mpg” pada dataframe “df” sebagai data yang ingin kita lihat representasi histogramnya, selanjutnya kita menggunakan interval sebanyak 15.

- **Menggunakan Seaborn**

Untuk membuat histogram dengan library seaborn kita gunakan method `histplot` lalu memasukkan dataframe yang akan kita lihat distribusinya pada parameter `data`, selanjutnya kita tentukan variabel yang kita inginkan pada parameter `x` dan menentukan banyaknya rentang dengan memasukkan nilai pada parameter `bins`.

```
sns.histplot(data = df,  
             x = "mpg",  
             bins = 15,  
             kde = True)  
  
plt.xlabel("MPG")  
plt.ylabel("Frequency")  
  
plt.show()
```

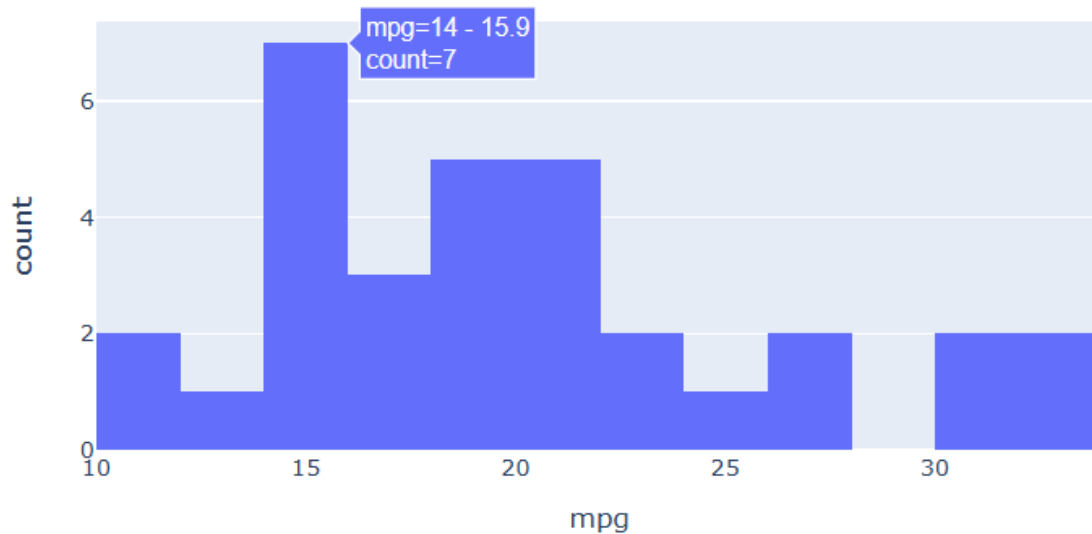


Pada sintaks diatas kita menggunakan kolom “mpg” pada dataframe “df” sebagai data yang ingin kita lihat representasi histogramnya, selanjutnya kita menggunakan interval sebanyak 15. Perhatikan bahwa kita juga menampilkan *kernel density estimation* (kde) dari variabel “mpg” dengan memasukkan nilai `True` pada parameter `kde`.

- **Menggunakan Plotly**

Untuk membuat histogram dengan library plotly kita gunakan method `histogram` lalu memasukkan dataframe yang akan kita lihat distribusinya, selanjutnya kita tentukan variabel yang kita inginkan pada parameter `x` dan menentukan banyaknya rentang dengan memasukkan nilai pada parameter `nbins`.

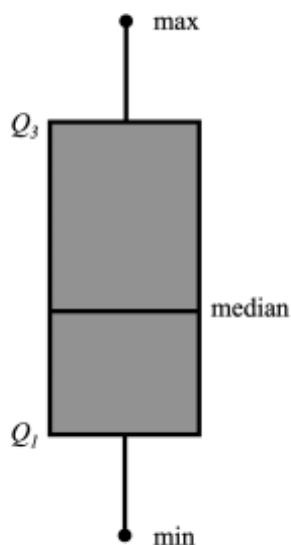

```
fig = px.histogram(df,
                    x = "mpg",
                    nbins = 15)
fig.show()
```



Pada sintaks diatas kita menggunakan kolom “mpg” pada dataframe “df” sebagai data yang ingin kita lihat representasi histogramnya, selanjutnya kita menggunakan interval sebanyak 15.

3.10 Box Plot

Pada pembahasan sebelumnya kita telah melihat bagaimana cara melihat distribusi dari data kita dengan bantuan histogram. Namun histogram bukan satu-satunya cara untuk memvisualisasikan informasi dari data kita. Cara lainnya yang akan dibahas disini adalah *Box Plot*.



Gambar diatas menyajikan anatomi dari box plot. Sesuai namanya, box plot diwakili oleh sebuah persegi panjang dengan sisi horizontalnya menunjukkan kuartil dari data kita dan juga dua garis dengan

ujung bawah adalah nilai minimum data dan ujung atasnya adalah nilai maksimum. Selain itu di dalam persegi panjang tersebut terdapat garis horizontal yang menunjukkan median dari data.

Selanjutnya kita akan membuat box plot dengan memanfaatkan dataframe “df” yang telah dipakai pada pembahasan sebelumnya.

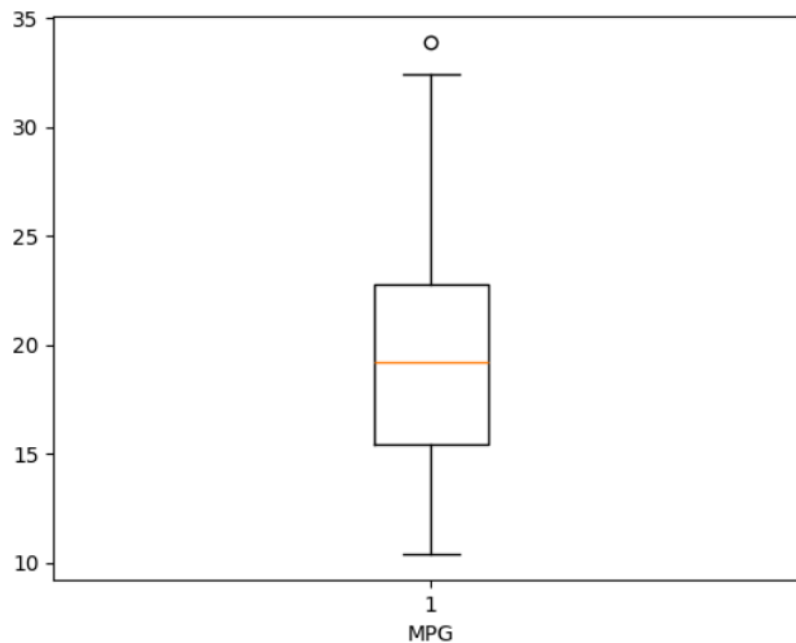
- **Menggunakan Matplotlib**

Untuk membuat box plot dengan library matplotlib kita gunakan method `boxplot` lalu memasukkan data yang akan kita lihat distribusinya.

```
#Membuat boxplot dari kolom "MPG" dataframe "df"
plt.boxplot(df["mpg"])

#Menamai sumbu horizontal dengan "MPG"
plt.xlabel("MPG")

plt.show()
```



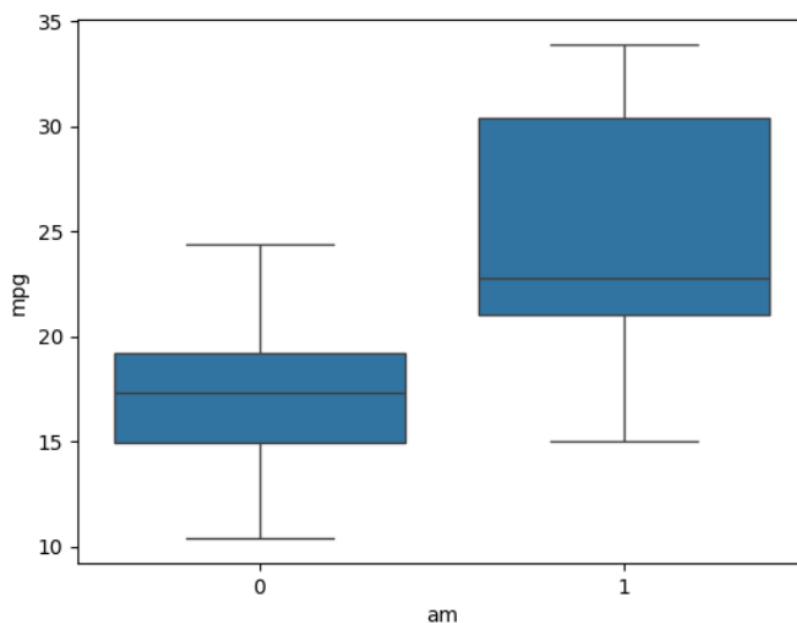
Pada sintaks diatas kita menggunakan kolom “mpg” pada dataframe “df” sebagai data yang ingin kita lihat representasi box plotnya. Perhatikan bahwa kita bisa melihat ringkasan dari data kita dengan satu buah grafik.

- **Menggunakan Seaborn**

Untuk membuat box plot dengan library seaborn kita gunakan method `boxplot` lalu memasukkan data yang akan kita lihat distribusinya. Selain melihat box plot berdasarkan nilai satu kolom kita juga bisa membuat box plot dari suatu data numerik berdasarkan kategorinya. Sebagai contoh kita mempunyai data nilai dari mata pelajaran Matematika suatu sekolah, kita bisa melihat distribusi nilai dari siswa laki-laki dan perempuan dengan menggunakan box plot.

Pada dataframe “df” kita mempunyai kolom numerik “mpg” yang menyatakan jarak tempuh mobil per gallon dan juga kolom kategorik “am” yang menyatakan jenis transmisi dari mobil yang diuji (apakah merupakan mobil matic (label 0) atau manual (label 1)). Kita akan melihat distribusi dari jarak tempuh mobil per gallon dilihat dari jenis transmisinya dengan menggunakan box plot.

```
sns.boxplot(x = "am",  
            y = "mpg",  
            data = df)  
plt.show()
```

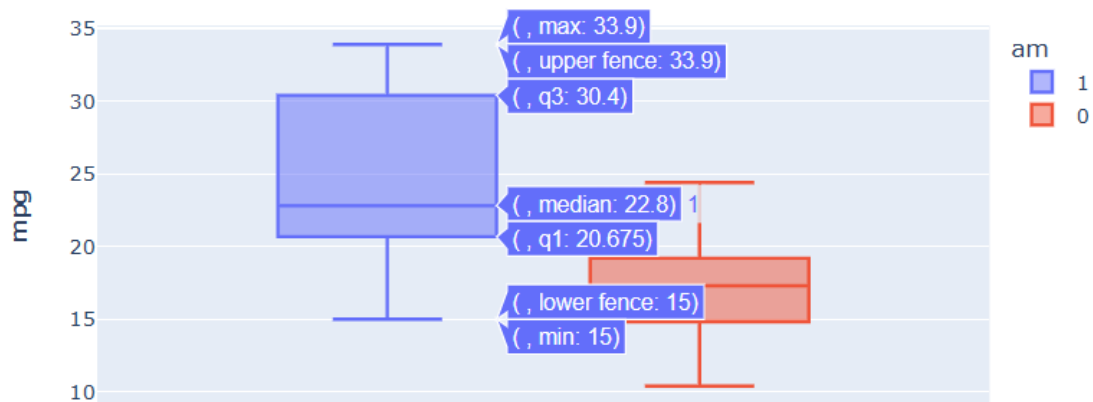


Pada sintaks diatas kita menggunakan kolom “am” sebagai sumbu horizontal dan kolom “mpg” sebagai sumbu vertikal. Terlihat bahwa dengan jumlah bahan bakar yang sama, menggunakan mobil dengan transmisi manual akan memberikan jarak tempuh yang cenderung lebih jauh daripada mobil automatic.

- **Menggunakan Plotly**

Untuk membuat box plot dengan library plotly kita gunakan method `box` lalu memasukkan dataframe yang kita inginkan. Selanjutnya kita tentukan variabel mana yang dijadikan sebagai sumbu vertikal pada parameter `y` dan kita tentukan parameter `color`.

```
fig = px.box(df,
             y = "mpg",
             color = "am")
fig.show()
```



Pada sintaks diatas kita menggunakan kolom “mpg” pada dataframe “df” sebagai data yang ingin kita lihat representasi box plotnya selanjutnya kita bedakan berdasarkan nilai pada kolom “am”.

3.11 Heatmap

Heatmap merupakan grafik yang menggambarkan nilai-nilai pada data dengan menggunakan warna yang bervariasi ketebalannya bergantung kepada nilai pada data yang berkorespondensi. Jika kita ingin menekankan kepada pembaca dengan cara menampilkan nilai-nilai pada data yang signifikan maka kita dapat menggunakan heatmap untuk membantu highlight nilai-nilai tersebut.

Untuk mengilustrasikan heatmap menggunakan dataframe “df” yang digunakan pada pembahasan sebelumnya, kita akan membuat kolom “transmission” yang akan bernilai “manual” jika “am” bernilai 0 dan “automatic” jika “am” bernilai “1”.

```
wrd = {0 : "manual", 1:"automatic"}
df["transmission"] = df["am"].map(wrd)
df.head()
```

	Car	mpg	cyl	disp	hp	drat	wt	qsec	vs	am	gear	carb	transmission
0	Mazda RX4	21.0	6	160.0	110	3.90	2.620	16.46	0	1	4	4	automatic
1	Mazda RX4 Wag	21.0	6	160.0	110	3.90	2.875	17.02	0	1	4	4	automatic
2	Datsun 710	22.8	4	108.0	93	3.85	2.320	18.61	1	1	4	1	automatic
3	Hornet 4 Drive	21.4	6	258.0	110	3.08	3.215	19.44	1	0	3	1	manual
4	Hornet Sportabout	18.7	8	360.0	175	3.15	3.440	17.02	0	0	3	2	manual

Selanjutnya kita akan membuat tabel pivot yang berisikan banyaknya mobil yang mempunyai cylinder 4,6, atau 8 berdasarkan jenis transmisinya.

```
#Mengelompokkan mobil-mobil berdasarkan transmisi dan jumlah cylinder
carcount = df.groupby(["transmission", "cyl"])

#Menghitung banyaknya mobil yang masuk tiap kategori
carcount = carcount.count()

#Mengambil kolom transmisi, cylinder, Car
carcount = carcount.filter(["transmission", "cyl", "Car"])

#Menghilangkan index
carcount = carcount.reset_index()
carcount
```

	transmission	cyl	Car
0	automatic	4	8
1	automatic	6	3
2	automatic	8	2
3	manual	4	3
4	manual	6	4
5	manual	8	12

Dari tabel diatas terlihat bahwa banyaknya mobil yang transmisinya automatic dengan jumlah cylinder 4 adalah 8 dan seterusnya. Selanjutnya kita akan membuat tabel dengan baris menyatakan banyaknya cylinder dan kolom menyatakan jenis transmisi.

```
carheatmap = carcount.pivot(index = "cyl", columns = "transmission")["Car"]
carheatmap
```

transmission	automatic	manual
cyl		
4	8	3
6	3	4
8	2	12

Tabel diatas masih memberikan informasi yang sama dengan sebelumnya namun dengan bentuk yang berbeda. Selanjutnya kita akan membuat heatmap dengan library-library yang telah kita bahas sebelumnya.

- **Menggunakan Matplotlib**

Untuk membuat heatmap dengan library matplotlib kita gunakan method `pcolor` lalu memasukkan data yang akan kita lihat distribusinya dan penggunaan warna pada heatmap kita.

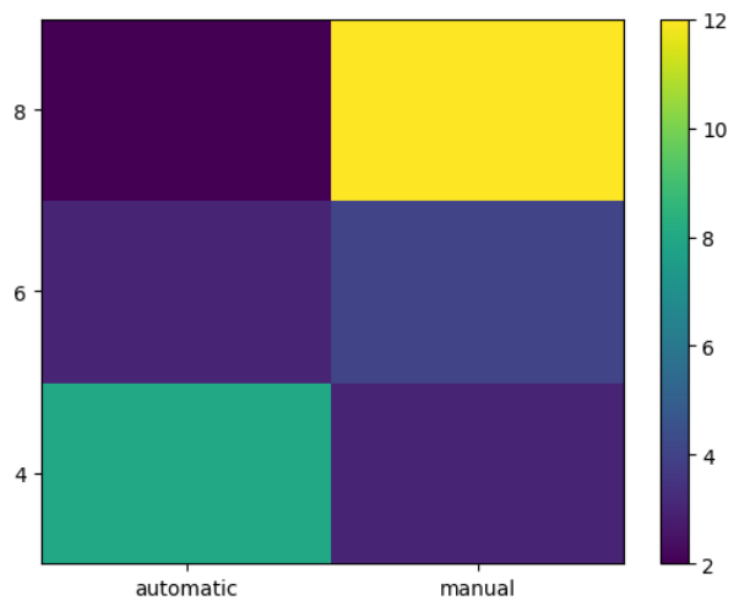
```
import numpy as np

plt.pcolor(carheatmap)

xt = np.arange(0.5, len(carheatmap.columns), 1)
yt = np.arange(0.5, len(carheatmap.index), 1)

plt.xticks(xt, carheatmap.columns)
plt.yticks(yt, carheatmap.index)
plt.colorbar()

plt.show()
```



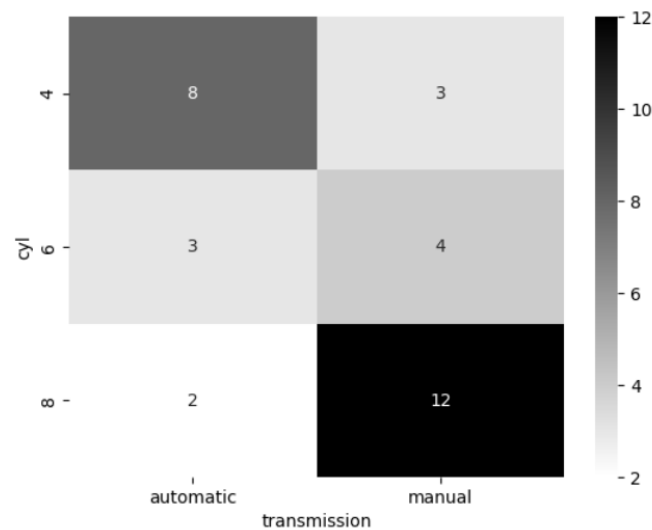
Perhatikan bahwa kita menggunakan tabel pivot untuk digunakan sebagai input pada method `pcolor`. Terlihat bahwa semakin terang warna pada heatmap maka semakin banyak mobil yang berada pada kategori tersebut.

- **Menggunakan Seaborn**

Untuk membuat heatmap dengan library seaborn kita gunakan method `heatmap` lalu memasukkan tabel pivot yang akan kita lihat distribusinya selanjutnya kita tentukan warna yang akan kita gunakan pada parameter `cmap`.

```
import seaborn as sns
sns.heatmap(carheatmap,
            cmap = "binary",
            annot = True)

plt.show()
```

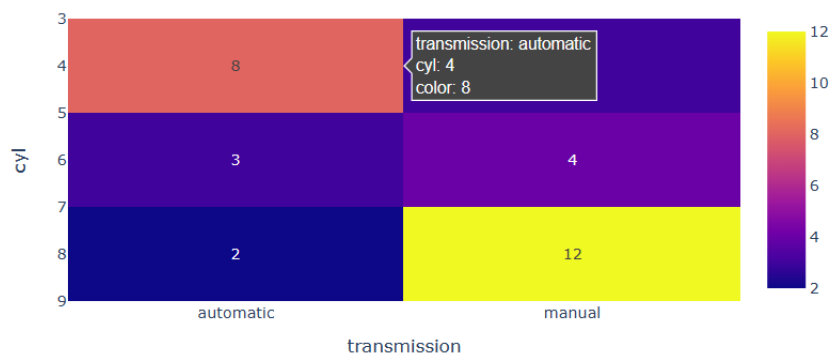


Perhatikan bahwa seaborn memberikan cara yang lebih mudah untuk membuat heatmap dibandingkan matplotlib. Pada sintaks diatas kita menggunakan tabel pivot “carheatmap”, lalu memberikan warna hitam putih dengan `cmap = “binary”` dan menampilkan nilai pada masing-masing kategori.

- **Menggunakan Plotly**

Untuk membuat heatmap dengan library plotly kita cukup menampilkan tabel pivot kita menggunakan method `imshow`

```
import plotly.express as px
fig = px.imshow(carheatmap,
                text_auto = True)
fig.show()
```



Sama halnya dengan Seaborn, plotly memberikan sintaks yang lebih mudah dibandingkan dengan matplotlib. Pada sintaks diatas kita cukup menggunakan tabel pivot “carheatmap” sebagai input lalu untuk menampilkan nilai pada setiap kategori kita gunakan parameter `text_auto`.

3.12 Pairplot

Salah satu grafik yang sering digunakan untuk eksplorasi data adalah *Pairplot*. Pair plot menampilkan hubungan antar pasangan-pasangan variabel pada data sehingga kita tidak perlu membuat plot antar variabel secara individu. Kita hanya akan menggunakan library seaborn untuk membuat pair plot dari variabel “mpg”, “disp”, “hp”, dan “drat” pada dataframe “df” yang kita bedakan berdasarkan kolom “am”.

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

#Membuat path dari file yang kita punya
path = "Downloads/pelatihan/cars.csv"
df = pd.read_csv(path)

#mengambil kolom-kolom yang dibutuhkan
mycars = df.filter(["mpg", "disp", "hp", "drat", "am"])
mycars.head()
```

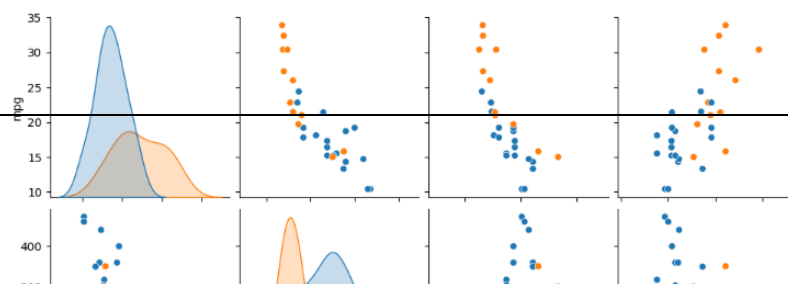
	mpg	disp	hp	drat	am
0	21.0	160.0	110	3.90	1
1	21.0	160.0	110	3.90	1
2	22.8	108.0	93	3.85	1
3	21.4	258.0	110	3.08	0
4	18.7	360.0	175	3.15	0

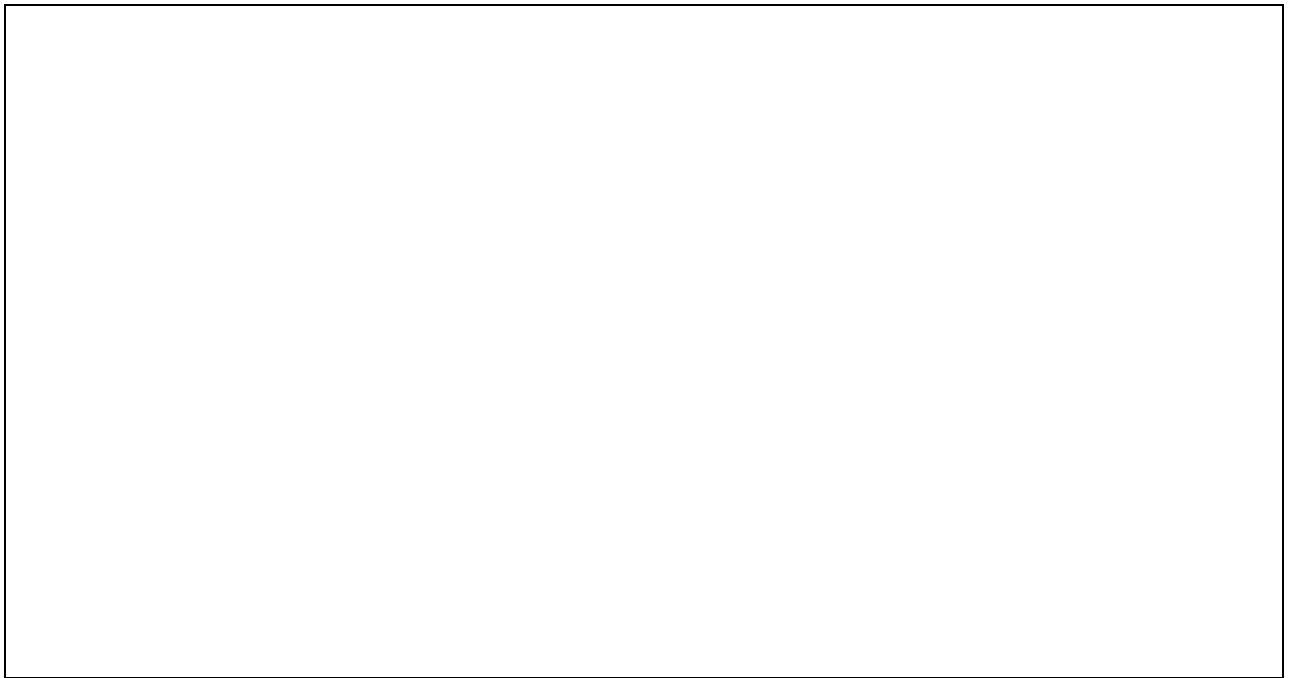
Untuk membuat pairplot dengan seaborn kita gunakan method `pairplot` lalu untuk menampilkan jenis plot antar variabel kita gunakan parameter `kind` yang dapat diisi dengan salah satu nilai ‘scatter’, ‘kde’, ‘hist’, atau ‘reg’. Selanjutnya kita gunakan parameter `hue` untuk membedakan warna sesuai kategori yang kita inginkan.

```
#mengambil kolom-kolom yang dibutuhkan

mycars = df.filter(["mpg", "disp", "hp", "drat", "am"])

#Membuat pairplot dengan dataframe mycars
#Hubungan antar variabel menggunakan scatterplot
# Membedakan warna berdasarkan kolom "am"
sns.pairplot(mycars,
             kind = "scatter",
             hue = "am")
plt.show()
```





Pada sintaks diatas kita mengambil kolom “mpg”, “disp”, “hp”, “drat”, dan “am” pada dataframe “df” selanjutnya menyimpannya dengan nama “mycars. Kita menggunakan scatterplot untuk menggambarkan hubungan antar variabel lalu untuk membedakan data berdasarkan nilai pada variabel “am” kita gunakan parameter `hue = “am”`.

BAB IV

UJI HIPOTESIS DAN ANALISIS MEAN

Setelah membahas mengenai library-library pada Python dan penggunaannya untuk visualisasi data, selanjutnya kita akan membahas mengenai statistika inferensi khususnya terkait uji hipotesis dan analisis mean. Pada bab ini akan dibahas mengenai uji hipotesis, uji normalitas, uji hipotesis satu sampel, uji hipotesis dua sampel, dan analisis varian.

4.1 Sampel dan Populasi

Populasi merupakan seluruh objek/target utama penelitian. Seringkali untuk mendapatkan data dari keseluruhan populasi membutuhkan biaya dan tenaga yang sangat besar sehingga diperlukan cara untuk mendapatkan kesimpulan dari sebagian populasi saja. **Sampel** merupakan bagian dari populasi yang diharapkan dapat mewakili karakteristik populasi. Dengan demikian, diharapkan kita dapat menarik kesimpulan mengenai populasi dengan melihat karakteristik pada sampel.

Hipotesis statistik adalah pernyataan mengenai keseluruhan populasi. Untuk memberikan hipotesis statistik maka diperlukan pengambilan data populasi. Sebagai contoh, seorang kontraktor menyuplai tanaman pada kliennya. Satu ikat tanaman dikatakan bagus jika kurang dari 8% tanaman tersebut masih hidup. Untuk melakukan pengecekan pada seluruh tanaman memerlukan biaya yang sangat besar sehingga seorang klien ingin untuk mengambil sebagian tanaman sebagai sampel untuk memutuskan membelinya atau tidak. Dia akan membeli tanaman tersebut jika sampel yang dia ambil mengindikasikan bahwa tanaman dari kontraktor tersebut bagus. Dalam hal seluruh tanaman kontraktor berperan sebagai populasi dan tanaman yang dicek oleh klien sebagai sampel. Hipotesis statistik yang dipakai adalah tanaman yang masih hidup kurang/lebih dari sama dengan 8%.

4.2 Uji Hipotesis

Uji hipotesis dilakukan untuk mengetahui apakah dugaan tentang karakteristik populasi didukung kuat oleh sampel atau tidak. Terdapat dua jenis hipotesis, yakni hipotesis nol (H_0 , *null hypothesis*) dan hipotesis alternative (H_a , *alternative hypothesis*).

Hipotesis Nol dan Hipotesis Alternatif

H_0 adalah hipotesis yang dirumuskan dengan harapan untuk ditolak, umumnya berupa pernyataan tidak adanya perbedaan karakteristik populasi. Sedangkan H_a digunakan untuk menunjukkan pernyataan mendapat dukungan kuat dari sampel. Terdapat dua kemungkinan yang akan terjadi

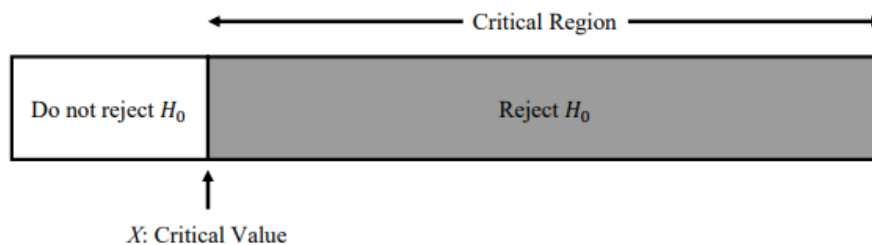
- H_0 ditolak dan H_a diterima karena terdapat bukti kuat untuk menyimpulkan adanya H_a
- H_0 tidak ditolak karena tidak adanya bukti kuat untuk menyimpulkan adanya H_a . Ingat H_0 tidak ditolak bukan berarti H_0 diterima.

Sebagai contoh, seorang kontraktor tanaman mengklaim bahwa hanya ada 5% tanaman yang dia miliki dalam kondisi rusak. Kita akan menginvestigasi peluang p bahwa tanaman yang dimiliki oleh kontraktor tersebut rusak sama dengan 5% atau lebih dari 5%. Sehingga hipotesis yang kita buat diformulasikan sebagai

$$H_0 : p = 0.05,$$

$$H_a : p > 0.05$$

Kita mengambil sampel sebanyak 100 tanaman dan melakukan pengujian kualitas dari tanaman yang diambil untuk menentukan apakah tanaman tersebut rusak atau tidak. Dimisalkan banyaknya tanaman yang rusak adalah X maka nilai ini disebut **statistik uji**. Misalkan kita akan menolak H_0 jika $X \geq 10$, maka nilai 10 disebut **nilai kritis**. Daerah yang membuat H_0 ditolak disebut **daerah kritis**.



Kesalahan

Melakukan penarikan kesimpulan mengenai populasi dari sampel bukan merupakan hal yang dipastikan kebenarannya. Jika kita menolak H_0 ketika H_a memang benar maka kita melakukan pengujian dengan baik. Namun, adakalanya kita menolak H_0 ketika H_0 bernilai benar. Terdapat dua tipe kesalahan yang akan terjadi dalam uji hipotesis yaitu **Tipe I** dan **Tipe II**. Kesalahan tipe I terjadi ketika kita menolak H_0 ketika H_0 seharusnya benar, sedangkan kesalahan tipe II terjadi ketika kita tidak menolak H_0 ketika H_a benar.

	H_0 benar	H_a benar
Tidak Menolak H_0	Keputusan benar	Kesalahan tipe II (β)
Menolak H_0	Kesalahan tipe I (α)	Keputusan benar

Tingkat signifikansi dan p-value

Tipe I terjadi ketika kita menolak H_0 padahal kenyataannya H_0 benar, ini seperti menghukum terdakwa padahal mereka tidak bersalah. Peluang kesalahan tipe I terjadi disebut **tingkat signifikansi** (α). Semakin kecil α maka akan semakin kecil terjadinya kesalahan tipe I, kita seringkali mengambil α lebih kecil dari sama dengan 0.05. Tipe II terjadi ketika kita tidak menolak H_0 padahal kenyataannya H_a yang benar. Peluang kesalahan tipe II terjadi dilambangkan dengan β dan $1 - \beta$ disebut sebagai **power uji**.

Selain penarikan kesimpulan melalui statistic uji, kita juga bisa melakukannya dengan memanfaatkan *p-value*. **p-value** adalah nilai terkecil dari tingkat signifikansi untuk menolak H_0 . Apabila $p - value < \alpha$ maka data mendukung penolakan H_0 .

Kesimpulan

Setelah menyusun uji hipotesis yang akan dilakukan dan mendapatkan data mengenai sampel suatu populasi maka kita dapat menarik kesimpulan terkait populasi dari nilai yang diperoleh pada sampel. Pada contoh sebelumnya seorang klien ingin melakukan pembelian kepada kontraktor tersebut jika peluang rusak kurang dari 8%. Sehingga dibentuk dua pernyataan sebagai hipotesis, yakni :

- $H_0 : p \geq 0.08$
- $H_a : p < 0.08$

Misal dari hasil sebelumnya diperoleh $p - value$ sebesar 0.021 dan klien tersebut menggunakan tingkat signifikansi 0.05 maka kesimpulan yang dapat ditarik dari uji ini adalah H_0 ditolak dan H_a diterima sehingga klien tersebut memutuskan untuk membeli tanaman dari kontraktor tersebut.

4.3 Uji Normalitas

Distribusi normal atau distribusi Gaussian merupakan salah satu distribusi yang sangat penting dalam statistika. Seringkali analisis statistika dimulai dengan eksplorasi data dan dilanjutkan dengan menguji apakah data yang kita punya berdistribusi normal sehingga kita bisa memilih uji hipotesis yang tepat untuk data kita.

Distribusi normal ditentukan oleh dua parameter yaitu rata-rata (μ) dan standard deviasi (σ^2). Rata-rata dari suatu data menunjukkan titik dimana data kita berkumpul dan standard deviasi menyatakan seberapa jauh data-data kita dari nilai rata-rata. Kita akan memulai uji normalitas dengan menggunakan plot yang disebut Q-Q plot dan akan dilanjutkan dengan beberapa uji formal. Uji yang dilakukan adalah terhadap hipotesis

H_0 : Sampel/data berasal dari distribusi normal

H_a : Sampel/data tidak berasal dari distribusi normal

Q-Q Plot

Q-Q Plot atau Quantile-Quantile plot merupakan salah satu cara untuk mengecek apakah data kita berdistribusi normal menggunakan grafik. Ide dari plot ini adalah melihat distribusi data pada setiap quantil. Jika quantil dari dua data datang dari distribusi yang sama maka akan membentuk garis lurus. Semakin jauh bentuk plot dari garis lurus maka semakin kecil kemungkinan mereka mempunyai distribusi yang sama. Yang perlu diingat adalah Q-Q plot hanya menjadi uji secara visual dan direkomendasikan untuk melakukan uji formal yang akan dibahas nantinya.

Kita akan menggunakan data “normal_skew.csv” yang mengandung data berdistribusi normal dan membandingkannya dengan data yang tidak berdistribusi normal. Data ini dapat didownload pada link <https://doi.org/10.6084/m9.figshare.17306285.v1>.

```
import pandas as pd
df = pd.read_csv("Downloads/pelatihan/normal_skew.csv")
df.head()
```

	normal_example	skewed_example
0	0.068676	2.023029
1	-0.143016	0.330508
2	-0.316584	0.140008
3	0.979202	0.255954
4	-0.739245	0.122830

Selanjutnya untuk membuat Q-Q plot dari setiap kolom kita membutuhkan method `probplot` dari library `stats`. Method ini akan memberikan kita Q-Q plot dengan distribusi awalnya adalah normal yang bisa diatur dengan parameter `dist="norm"`. Mula-mula kita akan menampilkan QQ plot dari kolom “normal_example”.

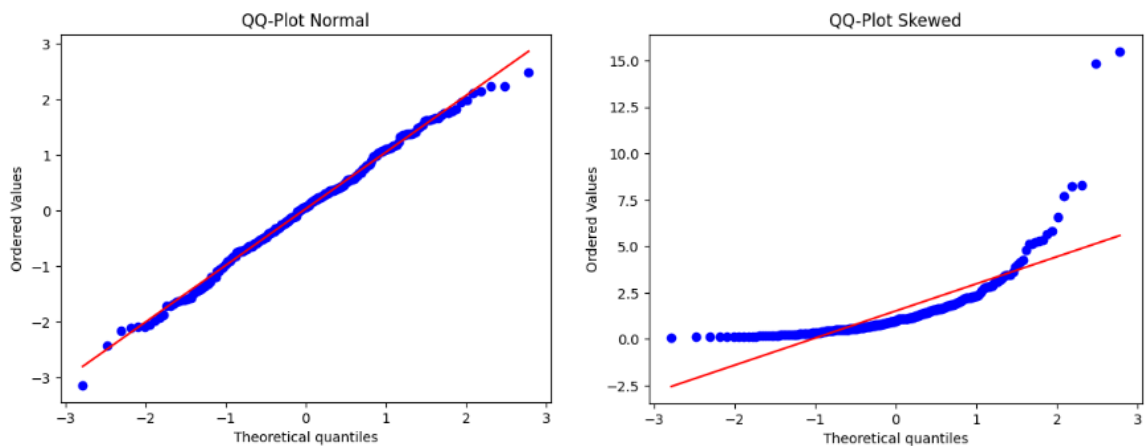
```
from scipy import stats
import matplotlib.pyplot as plt

stats.probplot(df["normal_example"],
               dist = "norm",
               plot = plt)
plt.title("QQ-Plot Normal")
plt.show()
```

Selanjutnya kita tampilkan QQ plot dari kolom “skewed_example”

```
stats.probplot(df["skewed_example"],
               dist = "norm",
               plot = plt)
plt.title("QQ-Plot Skewed")
plt.show()
```

diperoleh plot sebagai berikut



Semakin jauh titik pada QQ plot dari garis maka akan semakin besar kemungkinan data tersebut tidak mempunyai distribusi yang sama dengan distribusi acuan yang dipakai (dalam hal ini distribusi normal). Terlihat bahwa QQ plot dari kolom “normal_example” mendekati garis lurus sedangkan “skewed_example” jauh dari garis lurus sehingga tidak berdistribusi normal.

Uji Shapiro – Wilk

Merupakan salah satu metode uji normalitas nonparametric yang memiliki power yang besar, khususnya untuk sampel berukuran relatif kecil (< 2000). Jika $p - value$ yang diperoleh dari uji ini kurang dari tingkat signifikansi α yang telah kita tentukan maka H_0 ditolak dalam artian data tidak berdistribusi normal. Python menyediakan fungsi untuk uji Shapiro-Wilk yaitu `shapiro` yang memberikan statistik uji W dan $p - value$ hasil dari uji Shapiro – Wilk pada data yang kita masukkan. Untuk mengilustrasikan uji Shapiro – Wilk, kita akan menggunakan data “TaiwanEstate.csv” dan menguji apakah variabel `UsiaBangunan` berdistribusi normal.

```
df = pd.read_csv("Downloads/pelatihan/TaiwanEstate.csv",
                 sep = ";")
```

```
df.head()
```

	No	PeriodeTransaksi	UsiaBangunan	Kebisingan	JarakStasiunMRT	NumMinimarket	Humiditas	Latitude	Longitude	Harga	Harga_Lain
0	114	EARLY	14.8	26.9	393.2606	6	49	24.96172	121.53812	234.8	7.6
1	348	LATE	17.4	26.9	6488.0210	1	56	24.95719	121.47353	235.5	11.2
2	163	LATE	16.0	27.2	4066.5870	0	58	24.94297	121.50342	235.6	11.6
3	117	EARLY	30.9	22.7	6396.2830	1	52	24.94375	121.47883	244.0	12.2
4	227	EARLY	16.5	29.0	4082.0150	0	27	24.94155	121.50381	248.3	12.8

```
from scipy.stats import shapiro
```

```
shapiro(df["UsiaBangunan"])
```

```
ShapiroResult(statistic=0.94673911984696,
               pvalue=4.7983326720435985e-11)
```

Nilai p-value yang lebih kecil dari 5% menunjukkan H_0 ditolak pada tingkat signifikansi 5% sehingga data UsiaBangunan tidak berdistribusi normal.

Uji Jarque Bera (JB)

Uji JB merupakan salah satu jenis uji yang populer di kalangan komunitas ahli ekonometri dalam melakukan uji normalitas terhadap data. Statistik uji ini didefinisikan sebagai

$$JB = \frac{n}{6} \left(S_k^2 + \frac{(K - 3)^2}{4} \right)$$

Dimana n menunjukkan banyaknya observasi, S_k adalah estimasi dari skewness dan K menunjukkan estimasi dari kurtosis, yang didefinisikan sebagai

$$S_k = \frac{\frac{1}{n} \sum_{i=1}^n (x_i - \bar{x})^3}{\left(\frac{1}{n} \sum_{i=1}^n (x_i - \bar{x})^2 \right)^{\frac{3}{2}}}$$

Dan

$$K = \frac{\frac{1}{n} \sum_{i=1}^n (x_i - \bar{x})^4}{\left(\frac{1}{n} \sum_{i=1}^n (x_i - \bar{x})^2 \right)^2}$$

Dengan $\bar{x} = \frac{\sum_{i=1}^n x_i}{n}$. Di dalam Python uji Jarque Bera disediakan oleh submodule `stats` dalam bentuk method `jarque_bera` yang memberikan statistik uji JB dan $p - value$ hasil dari uji Jarque Bera pada data yang kita masukkan.

```
from scipy.stats import jarque_bera
jarque_bera(df["UsiaBangunan"])

SignificanceResult(statistic=23.433784595527165,
                    pvalue=8.154894182448043e-06)
```

Nilai p-value yang lebih kecil dari 5% menunjukkan H_0 ditolak pada tingkat signifikansi 5% sehingga data UsiaBangunan tidak berdistribusi normal.

Beberapa uji normalitas tersedia juga dalam paket `stats` diantaranya Anderson-Darling, Kolmogorov-Smirnov dan D'Agostino K-squared. Sebagai rangkuman berikut tabel uji normalitas yang disediakan oleh `stats` beserta sintaks dan hasil $p - value$ pada data UsiaBangunan.

Uji	Sintaks	$p - value$
Shapiro - Wilk	<code>shapiro(df["UsiaBangunan"])</code>	4.7983326720435985e-11
Jarque - Bera	<code>jarque_bera(df["UsiaBangunan"])</code>	8.154894182448043e-06
Anderson-Darling	<code>anderson(df["UsiaBangunan"])</code>	-
Kolmogorov-Smirnov	<code>kstest(df["UsiaBangunan"], "norm")</code>	0.0

D'Agostino K-squared	<code>normaltest(df["UsiaBangunan"])</code>	8.253756825553312e-13
----------------------	---	-----------------------

Untuk dapat menggunakan semua method diatas tanpa mengimport satu-satu dapat menggunakan perintah berikut terlebih dahulu

```
from scipy.stats import *
```

4.4 Uji Hipotesis dengan Satu Sampel

Setelah mengetahui bagaimana uji hipotesis bekerja, kita akan mencoba permasalahan lain terkait uji hipotesis yaitu ketika kita ingin mengecek apakah parameter hasil estimasi dari populasi berbeda secara signifikan dengan nilai hipotesa. Permasalahan ini disebut **uji hipotesis satu sampel**.

Uji Hipotesis Mean Sampel Besar Populasi Berdistribusi Sembarang

- **Estimator Titik untuk Mean**

$$\bar{X} = \frac{\sum_{i=1}^n x_i}{n}$$

- **Estimator Interval untuk Mean**

Untuk kasus ukuran sampel n besar ($n > 30$), berdasarkan Teorema Limit Pusat, maka variabel random

$$Z = \frac{\bar{X} - \mu}{\sigma / \sqrt{n}}$$

Akan mendekati distribusi normal standar sehingga uji ini biasa disebut sebagai **uji Z**.

Interval konfidensi $(1 - \alpha)100\%$ untuk μ diturunkan dengan menggunakan sifat dari variabel random Z diatas, dan didapat :

$$\bar{X} - Z_{\alpha/2} \frac{\sigma}{\sqrt{n}} \leq \mu \leq \bar{X} + Z_{\alpha/2} \frac{\sigma}{\sqrt{n}}$$

Dalam perhitungan, biasanya σ^2 (variansi dari populasi) tidak diketahui, tetapi dapat diganti dengan variansi sampel $S^2 = \frac{1}{n-1} \sum_{i=1}^n (x_i - \bar{x})^2$.

- **Uji Hipotesis untuk Mean**

Jika ingin menguji bahwa mean suatu populasi sama dengan nilai tertentu μ_0 , dengan n besar ($n > 30$). Langkah-langkah uji hipotesis ini adalah sebagai berikut :

1. Tentukan Hipotesis

A. $H_0 : \mu = \mu_0$ (uji dua sisi)

$$H_a : \mu \neq \mu_0$$

B. $H_0 : \mu \leq \mu_0$ (uji satu sisi)

$$H_a : \mu > \mu_0$$

C. $H_0 : \mu \geq \mu_0$ (uji satu sisi)

$$H_a : \mu < \mu_0$$

2. Tentukan tingkat signifikansi α
3. Statistik Penguji Z
4. Daerah kritis : H_0 ditolak bila :
 - A. $Z < -Z_{\frac{\alpha}{2}}$ atau $Z > Z_{\frac{\alpha}{2}}$
 - B. $Z > Z_{\alpha}$
 - C. $Z < -Z_{\alpha}$
5. Berdasarkan langkah 4 dan hasil penghitungan statistic penguji langkah 3, diambil kesimpulan apakah H_0 ditolak atau tidak ditolak pada tingkat signifikansi α .

Komputasi Uji Mean Sampel Besar Populasi Berdistribusi Sembarang dengan Python

Akan diberikan ilustrasi uji hipotesis mean satu sampel dari populasi yang berdistribusi sembarang dengan n besar. Untuk melakukan uji Z dengan Python kita perlu menginstall library “statsmodels”, langkah instalasinya tinggal mengetikkan `pip3 install statsmodels` pada Command Prompt. Library ini memberikan fungsi `ztest` untuk melakukan uji Z (dengan variansi populasi tidak diketahui). Jika variansi populasi diketahui maka kita dapat melakukan penghitungan dengan cara manual. Terdapat 3 jenis hipotesis alternatif sebagai nilai pada parameter `alternative`.

- “two-sided” untuk $H_a : \mu \neq \mu_0$
- “larger” untuk $H_a : \mu > \mu_0$
- “smaller” untuk $H_a : \mu < \mu_0$

Berikut ini contoh dari penggunaan uji mean dengan $n > 30$ dan variansi populasi tidak diketahui. Misalkan kita ingin menguji apakah nilai ujian mata pelajaran Matematika suatu sekolah kurang dari 50. Untuk keperluan tersebut kita mengambil sampel sebesar nilai dari 100 siswa (untuk simplisitas kita akan menggenerate angka dari 0-100 menggunakan numpy).

```
from numpy import random
data = random.randint(100, size=(100))
data
array([89, 99,  0, 41, 97, 68, 27, 14, 35, 36, 28, 23,  7, 33, 25, 76, 27,
        9, 70, 87, 18, 82, 52, 65, 32, 23,  3, 63, 39, 41, 17, 85, 83, 65,
       37, 25, 24, 11, 49, 80, 45, 88, 51, 20, 66, 76, 95, 63, 61, 72, 57,
       64, 89, 79, 46, 50, 19, 10, 77,  0, 42, 21, 14, 63, 93,  6, 61, 17,
       36, 69, 66, 13,  5, 92, 53, 19, 67, 27, 35, 48,  5, 25, 90,  1, 34,
       53, 17, 75, 55, 73, 59, 85,  7, 17, 91, 90, 12, 52, 24, 68])
```

Karena data sudah cukup besar maka untuk keperluan simplisitas kita dapat mengasumsikan bahwa data sudah berdistribusi normal. Oleh karena itu tidak diperlukan uji normalitas. Misalkan μ menunjukkan rata-rata nilai seluruh siswa dan $\mu_0 = 50$.

1. Menentukan hipotesis.

$H_0 : \mu \leq 50$ (rata-rata nilai siswa kurang dari sama dengan 50)

$H_a : \mu > 50$ (rata-rata nilai siswa lebih dari 50)

2. Kita gunakan tingkat signifikansi $\alpha = 5\%$.
3. Untuk memperoleh nilai statistic uji kita gunakan fungsi `ztest` dari library “statsmodels”.

```
from statsmodels.stats.weightstats import ztest
#Dicari nilai z yang memberikan luas 0.05 di sebelah kanan
ztest(data, value=50, alternative = "larger")
```

yang memberikan output (-1.0704796012319149, 0.8577982569711657). Artinya nilai statistic uji $Z = -1.0705$ dan $p - value = 0.8578$.

Atau bisa dengan menghitung statistic uji Z secara manual yang akan memberikan nilai yang sama.

```
import numpy as np
n = 100
xbar = data.mean()
s = np.std(data, ddof = 1)
mu = 50
z = (xbar-mu) / (s/np.sqrt(n))
print(z)
#pval adalah luas wilayah daerah kritis
pval = 1-norm.cdf(z)
print(pval)
```

4. Daerah kritis : H_0 ditolak jika $Z > Z_{\alpha}$. Untuk memperoleh nilai $Z_{0.05}$ kita gunakan sintaks berikut

```
alpha = 0.05
print(norm.ppf(1-alpha))
```

yang memberikan nilai $Z_{0.05} = 1.64$

5. Kesimpulan. Karena $Z < Z_{0.05}$ maka H_0 tidak ditolak yang artinya belum cukup bukti untuk menyimpulkan nilai rata-rata siswa sekolah tersebut lebih dari 50. Selain itu kita juga bisa menggunakan nilai $p - value$ yang diperoleh pada langkah 3 yaitu $p - value = 0.8578 > \alpha$ sehingga H_0 tidak ditolak.

Uji Hipotesis Mean Populasi Normal

- **Estimator Interval untuk Mean Populasi Normal**

Misalkan X_1, X_2, \dots, X_n adalah sampel random yang diambil dari populasi normal dengan mean μ dan variansi σ^2 maka interval konfidensi $(1 - \alpha)100\%$ untuk μ adalah

A. $\bar{X} - Z_{\frac{\alpha}{2}} \frac{\sigma}{\sqrt{n}} \leq \mu \leq \bar{X} + Z_{\frac{\alpha}{2}} \frac{\sigma}{\sqrt{n}}$ (jika σ^2 diketahui)

B. $\bar{X} - t_{(n-1; \frac{\alpha}{2})} \frac{s}{\sqrt{n}} \leq \mu \leq \bar{X} + t_{(n-1; \frac{\alpha}{2})} \frac{s}{\sqrt{n}}$ (jika σ^2 tidak diketahui)

C. $\bar{X} - Z_{\frac{\alpha}{2}} \frac{s}{\sqrt{n}} \leq \mu \leq \bar{X} + Z_{\frac{\alpha}{2}} \frac{s}{\sqrt{n}}$ (jika σ^2 tidak diketahui dan n besar)

- **Uji Hipotesis untuk Mean Populasi Normal**

Jika ingin menguji bahwa mean suatu populasi normal sama dengan nilai tertentu μ_0 , langkah-langkah uji hipotesis ini adalah sebagai berikut :

1. Tentukan Hipotesis

- A. $H_0 : \mu = \mu_0$ (uji dua sisi)

$$H_a : \mu \neq \mu_0$$

- B. $H_0 : \mu \leq \mu_0$ (uji satu sisi)

$$H_a : \mu > \mu_0$$

- C. $H_0 : \mu \geq \mu_0$ (uji satu sisi)

$$H_a : \mu < \mu_0$$

2. Tentukan tingkat signifikansi α

3. Statistik Penguji Z

4. Daerah kritis : H_0 ditolak bila :

- A. $Z < -Z_{\frac{\alpha}{2}}$ atau $Z > Z_{\frac{\alpha}{2}}$ (untuk uji Z)

$$t < -t_{(n-1; \frac{\alpha}{2})} \text{ atau } t > t_{(n-1; \frac{\alpha}{2})} \text{ (untuk uji t)}$$

- B. $Z > Z_{\alpha}$ (untuk uji Z)

$$t > t_{(n-1; \alpha)} \text{ (untuk uji t)}$$

- C. $Z < -Z_{\alpha}$ (untuk uji Z)

$$t < -t_{(n-1; \alpha)} \text{ (untuk uji t)}$$

5. Berdasarkan langkah 4 dan hasil penghitungan statistic penguji langkah 3, diambil kesimpulan apakah H_0 ditolak atau tidak ditolak pada tingkat signifikansi α .

Komputasi Uji Mean Populasi Normal dengan Python

Akan diberikan ilustrasi uji hipotesis mean satu sampel dari populasi yang berdistribusi normal menggunakan Python. Pada Python tersedia fungsi `ttest_1samp` untuk menghitung statistic uji dan p – *value* dari uji-t. Terdapat 3 jenis hipotesis alternatif sebagai nilai pada parameter `alternative`.

- A. “two-sided” untuk $H_a : \mu \neq \mu_0$

- B. “greater” untuk $H_a : \mu > \mu_0$

- C. “less” untuk $H_a : \mu < \mu_0$

Akan digunakan dua data untuk mengilustrasikan uji hipotesis mean satu sampel dari populasi yang berdistribusi normal.

- **Data TaiwanEstate**

Misalkan ingin diketahui apakah harga rata-rata bangunan di Taiwan lebih tinggi dari 340000/ping.

Normalitas

Mula-mula akan dicek apakah data memenuhi asumsi yaitu berdistribusi normal. Akan dilakukan uji normalitas Shapiro – Wilk dengan hipotesis

H_0 : Harga rata-rata bangunan per ping di Taiwan berasal dari populasi yang berdistribusi Normal

H_a : Harga rata-rata bangunan per ping di Taiwan tidak berasal dari populasi yang berdistribusi Normal

Akan digunakan tingkat signifikansi 5%. Selanjutnya akan digunakan bantuan Python untuk mendapatkan p – $value$ dari uji Shapiro-Wilk pada variabel “Harga”.

```
import pandas as pd
from scipy.stats import *

df = pd.read_csv("Downloads/pelatihan/TaiwanEstate.csv",
                 sep = ";")
shapiro(df["Harga"])

ShapiroResult(statistic=0.994888889142379,
               pvalue=0.18744587664238638)
```

Diperoleh hasil p – $value > 5\%$ sehingga H_0 tidak ditolak pada tingkat signifikansi 5% (harga bangunan berdistribusi normal)

Uji Mean Satu Sampel

1. Menentukan hipotesis.

Misal μ menunjukkan rata-rata harga bangunan per ping di Taiwan (dalam ribuan TWD) dan μ_0 adalah harga proyeksi yaitu 340. Hipotesis yang diajukan adalah

$$H_0: \mu \leq 340$$

$$H_a: \mu > 340$$

2. Akan digunakan tingkat signifikansi $\alpha = 5\%$.
3. Karena variansi populasi tidak diketahui maka akan dilakukan uji-t. Akan digunakan bantuan Python untuk menentukan p – $value$ dari uji-t.

```
ttest_1samp(df["Harga"],
            popmean = 340,
            alternative = "greater")

TtestResult(statistic=2.593198829839057,
             pvalue=0.004923244814715349, df=413)
```

4. Kesimpulan. Diperoleh p – $value = 0.0049 < 0.05$ sehingga data mendukung penolakan H_0 . Dengan demikian, rata-rata harga bangunan di Taiwan lebih tinggi dari proyeksi pada taraf signifikansi 5%.

- **Data Karyawan**

Sebuah perusahaan sedang melakukan evaluasi kinerja karyawannya. Disinyalir bahwa keterlambatan karyawan merupakan faktor penting yang memengaruhi perusahaan. Jika rata – rata keterlambatan seluruh karyawan lebih dari 7 menit, akan diberlakukan kebijakan baru. Diambil sampel 50 karyawan dan diperoleh data keterlambatan. Apakah perusahaan perlu menetapkan kebijakan baru?

Normalitas

Mula-mula akan dicek apakah data memenuhi asumsi yaitu berdistribusi normal. Akan dilakukan uji normalitas Shapiro – Wilk dengan hipotesis

H_0 : Rata-rata keterlambatan berasal dari populasi yang berdistribusi Normal

H_a : Rata-rata keterlambatan tidak berasal dari populasi yang berdistribusi Normal

Akan digunakan tingkat signifikansi 5%. Selanjutnya akan digunakan bantuan Python untuk mendapatkan $p - value$ dari uji Shapiro-Wilk pada variabel “Menit Terlambat”.

```
import pandas as pd
from scipy.stats import *

df = pd.read_csv("Downloads/pelatihan/Karyawan.csv",
                 sep = ";")
shapiro(df["Menit Terlambat"])

ShapiroResult(statistic=0.980222706372278,
               pvalue=0.5613943675189909)
```

Diperoleh hasil $p - value > 5\%$ sehingga H_0 tidak ditolak pada tingkat signifikansi 5% (data “Menit Terlambat” berdistribusi normal).

Uji Mean Satu Sampel

1. Menentukan hipotesis.

Misal μ menunjukkan rata-rata lama keterlambatan karyawan. Hipotesis yang diajukan adalah

$$H_0: \mu \leq 7$$

$$H_a: \mu > 7$$

2. Akan digunakan tingkat signifikansi $\alpha = 5\%$.
3. Karena variansi populasi tidak diketahui maka akan dilakukan uji-t. Akan digunakan bantuan Python untuk menentukan $p - value$ dari uji-t.

```
ttest_1samp(df["Menit Terlambat"],
            popmean = 7,
            alternative = "greater")

TtestResult(statistic=1.355505334855727,
            pvalue=0.09073595738287162, df=49)
```

4. Kesimpulan. Diperoleh $p - value = 0.09 > 0.05$ sehingga H_0 tidak ditolak. Artinya tidak cukup bukti bahwa rata-rata keterlambatan karyawan lebih dari 7 menit.

Uji Proporsi Satu sampel

Kita telah membahas bagaimana cara melakukan uji mean populasi satu sampel. Kegunaan lain dari uji satu sampel ini adalah untuk menguji proporsi sukses dalam satu populasi sama dengan nilai tertentu p_0 . Sebagai contoh seorang kontraktor tanaman mengklaim bahwa di dalam kebunnya hanya terdapat 5% tanaman yang rusak. Kita mengambil beberapa sampel tanaman dan menguji apakah tanaman tersebut rusak atau tidak. Tentu saja persentase yang kita dapat akan berbeda dengan klaim dari kontraktor tersebut dan kita ingin menguji secara statistik apakah terdapat perbedaan yang signifikan antara proporsi sampel kita dengan klaim dari kontraktor.

- **Statistik uji**

$$Z = \frac{\bar{p} - p_0}{\sqrt{\frac{p_0(1 - p_0)}{n}}}$$

Dengan \bar{p} merupakan estimasi proporsi dan n merupakan banyaknya sampel.

- **Uji Hipotesis untuk Proporsi Satu Sampel**

Jika ingin menguji bahwa proporsi suatu populasi sama dengan nilai tertentu p_0 , langkah-langkah uji hipotesis ini adalah sebagai berikut :

1. Tentukan Hipotesis. Hanya terdapat satu H_0 yang mungkin yaitu $H_0: p = p_0$ dan terdapat tiga kemungkinan H_a yaitu
 - A. $H_a: p \neq p_0$
 - B. $H_a: p > p_0$
 - C. $H_a: p < p_0$
2. Tentukan tingkat signifikansi α
3. Statistik Penguji Z
4. Daerah kritis : H_0 ditolak jika
 - A. $Z < -Z_{\frac{\alpha}{2}}$ atau $Z > Z_{\frac{\alpha}{2}}$
 - B. $Z > Z_{\alpha}$
 - C. $Z < -Z_{\alpha}$
5. Berdasarkan langkah 4 dan hasil penghitungan statistik penguji langkah 3, diambil kesimpulan apakah H_0 ditolak atau tidak ditolak pada tingkat signifikansi α .

Komputasi Uji Proporsi Satu Sampel dengan Python

Akan diberikan ilustrasi uji proporsi satu sampel menggunakan Python. Pada Python tersedia fungsi `proportions_ztest` dari submodule `proportion` untuk menghitung statistik uji dan $p - value$ dari uji proporsi. Terdapat beberapa parameter yang harus dimasukkan pada fungsi `proportions_ztest`.

- A. `count` – Banyaknya sukses dari percobaan yang dilakukan
- B. `nobs` – Banyaknya percobaan yang dilakukan
- C. `value` – Nilai yang digunakan untuk H_0
- D. `alternative` – Terdapat tiga pilihan yaitu “two-sided”, “smaller”, atau “larger” bergantung hipotesis alternatif yang kita punya.

Akan digunakan dua data untuk mengilustrasikan uji proporsi satu sampel.

- **Data TaiwanEstate**

Dari data harga bangunan di Taiwan, ingin diketahui apakah jual beli terjadi secara merata di pertengahan awal dan akhir tahun. Kolom “PeriodeTransaksi” mencatat periode transaksi (Januari-Juni atau Juli-Desember). Akan kita lihat banyaknya periode transaksi yang dilakukan secara “EARLY” atau “LATE”.

```
import pandas as pd

df = pd.read_csv("Downloads/pelatihan/TaiwanEstate.csv",
                 sep = ";")

#Menghitung kemunculan nilai-nilai berbeda pd kolom PeriodeTransaksi
df["PeriodeTransaksi"].value_counts()
```

PeriodeTransaksi	count
EARLY	265
LATE	149

Name: count, dtype: int64

Apabila diperhatikan, transaksi properti lebih banyak dilakukan di pertengahan pertama tahunan (EARLY). Uji proporsi didesain untuk melihat apakah perbedaan ini hanya “kebetulan” saja atau tidak. Hipotesis yang digunakan adalah

$$H_0: p_{EARLY} = 0.5$$

$$H_a: p_{EARLY} \neq 0.5$$

Penggunaan hipotesis di atas dengan alasan karena kita ingin menguji apakah proporsi transaksi EARLY dan LATE sama sehingga diperlukan $p_{EARLY} = 0.5$. Diperoleh `count = 265`, `nobs = 265+149`, `value = 0.5`, `alternative = “two-sided”`. Dengan menggunakan fungsi `proportions_ztest` diperoleh hasil sebagai berikut

```
from statsmodels.stats.proportion import proportions_ztest
proportions_ztest(count = 265,
                  nobs = 265+149,
                  value = 0.5,
                  alternative = "two-sided")

(5.938983147870953, 2.8679534174923876e-09)
```

Dari hasil diatas diperoleh $p - value = 2.8 \times 10^{-9} < 0.05$ sehingga data mendukung penolakan H_0 . Artinya proporsi transaksi pada EARLY berbeda signifikan dari 0.5, atau transaksi properti tidak merata sepanjang tahun.

- **Data Jaksa**

Seorang jaksa daerah ingin mencalonkan diri untuk jabatan jaksa kota. Dia memutuskan akan melepas jabatan lamanya jika lebih dari 90% pemilih partai mendukungnya. Untuk maksud tersebut, manajer kampanye dari partai tersebut melakukan survey terhadap 70 orang anggota partainya diambil secara random, untuk ditanyai setuju atau tidak terkait pencalonan jaksa tersebut. Didapat 54 orang menjawab setuju. Apa keputusan terbaik yang diambil jaksa tersebut?

Misalkan suatu kejadian dikatakan sukses jika pemilih mendukung jaksa. Hipotesis yang digunakan untuk permasalahan ini adalah

$$H_0: p \leq 0.9$$

$$H_a: p > 0.9$$

Dengan asumsi diatas diperoleh `count = 54`, `nobs = 70`, `value = 0.9`, `alternative = "larger"`. Dengan menggunakan fungsi `proportions_ztest` diperoleh hasil sebagai berikut

```
from statsmodels.stats.proportion import proportions_ztest
proportions_ztest(count = 54,
                  nobs = 70,
                  value = 0.9,
                  alternative = "larger")

(-2.5617376914898995, 0.9947925026704684)
```

Dari hasil diatas diperoleh $p - value = 0.995 > 0.05$ sehingga belum ada cukup bukti untuk menolak H_0 . Artinya pendukung jaksa tersebut kurang dari 90% dari partainya. Ia tidak disarankan untuk maju kampanye dan tetap pada jabatan lamanya.

4.5 Uji Dua Sampel

Pada pembahasan sebelumnya kita berfokus untuk menguji apakah parameter hasil estimasi berbeda secara signifikan dengan nilai hipotesa. Pada banyak kasus, kita tertarik untuk membandingkan dua populasi berbeda seperti mengestimasi selisih antara rata-rata dua populasi atau menentukan apakah perbedaan kedua nilai tersebut berbeda secara signifikan.

Salah satu pertimbangan yang harus diketahui adalah bagaimana data dari setiap populasi ini diambil. Pengambilan sampel dikatakan **independen** jika pengambilan sampel dari satu populasi tidak mempengaruhi populasi lain. Di sisi lain, pengambilan sampel dikatakan **berpasangan** jika pengambilan pada populasi satu berpengaruh pada pengambilan pada populasi lain. Berikut diberikan contoh dari pengambilan sampel independen dan berpasangan.

1. Pengukuran efektivitas lensa antiradiasi. Dalam penelitian ini, setiap orang mengenakan lensa biasa dan lensa antiradiasi dan dicatat lama waktu hingga mata memerah.
 - Data biasa dan antiradiasi diperoleh dari individu yang sama sehingga ia disebut **sampel berpasangan**.
 - Apabila setiap individu hanya diminta untuk mengenakan satu jenis lensa saja, ia disebut **sampel independen**.
2. Nilai pre-test (sebelum) dan post-test (sesudah) dicatat dari individu yang sama. Nilai post-test akan dipengaruhi oleh nilai pre-test. Data tersebut **berpasangan**.
3. Penghasilan seseorang 5 tahun dan 10 tahun setelah kelulusan adalah data **berpasangan**.
4. Kadar oksigen yang dihasilkan tanaman Anggrek di dua humiditas berbeda (tinggi dan rendah). Masing-masing Anggrek hanya dikenakan satu jenis humiditas dan tidak memengaruhi Anggrek di lokasi lain. Data ini **independen**.

Uji Proporsi Dua Sampel Independen

Terkadang kita ingin menguji apakah dua proporsi pada dua populasi sama. Sebagai contoh, kita ingin mengetahui proporsi dokter spesialis di Yogyakarta sama dengan proporsi dokter spesialis di Surabaya atau ingin mengetahui apakah proporsi perokok yang meninggal karena kanker lebih dari proporsi bukan merokok yang meninggal karena kanker.

Misalkan p_i menyatakan proporsi pada populasi $i \in \{1,2\}$. Pada uji ini kita ingin menguji

$$H_0: p_1 = p_2$$

Dengan H_a adalah salah satu dari pernyataan $p_1 < p_2$, $p_1 > p_2$, $p_1 \neq p_2$. Dimisalkan $p = p_1 - p_2$ tentu saja untuk menguji hipotesis sebelumnya sama halnya dengan menguji hipotesis

$$H_0: p = 0$$

Dengan H_a adalah salah satu dari pernyataan $p < 0$, $p > 0$, $p \neq 0$.

- **Estimator gabungan proporsi**

$$\hat{p} = \frac{x_1 + x_2}{n_1 + n_2}$$

Dengan x_i menyatakan banyaknya sukses pada sampel dari populasi i dan n_i menyatakan banyaknya sampel dari populasi i .

- **Statistik Uji**

$$z = \frac{\hat{p}_1 - \hat{p}_2}{\sqrt{\hat{p}\hat{q}\left(\frac{1}{n_1} + \frac{1}{n_2}\right)}}$$

Dengan $\hat{q} = 1 - \hat{p}$.

- **Uji Hipotesis untuk Proporsi Dua Sampel**

Jika ingin menguji apakah proporsi dua kelompok dari dua populasi berbeda adalah sama ($p_1 = p_2$), langkah-langkah uji hipotesis ini adalah sebagai berikut :

1. Tentukan Hipotesis. Hanya terdapat satu H_0 yang mungkin yaitu $H_0: p_1 = p_2$ dan terdapat tiga kemungkinan H_a yaitu
 - A. $p_1 \neq p_2$,
 - B. $p_1 > p_2$,
 - C. $p_1 < p_2$.
2. Tentukan tingkat signifikansi α
3. Statistik Penguji Z
4. Daerah kritis : H_0 ditolak jika
 - A. $Z < -Z_{\frac{\alpha}{2}}$ atau $Z > Z_{\frac{\alpha}{2}}$
 - B. $Z > Z_{\alpha}$
 - C. $Z < -Z_{\alpha}$
5. Berdasarkan langkah 4 dan hasil penghitungan statistic penguji langkah 3, diambil kesimpulan apakah H_0 ditolak atau tidak ditolak pada tingkat signifikansi α .

Komputasi Uji Proporsi Dua Sampel Independen Dengan Python

Akan diberikan ilustrasi uji proporsi dua sampel independ menggunakan Python. Pada Python tersedia fungsi `proportions_ztest` dari submodule `proportion` untuk menghitung statistik uji dan $p - value$ dari uji proporsi. Terdapat beberapa parameter yang harus dimasukkan pada fungsi `proportions_ztest`.

- A. `count` – Banyaknya sukses dari percobaan yang dilakukan, dalam kasus ini karena terdapat dua sampel maka dimasukkan dalam bentuk list $[c_1, c_2]$
- B. `nobs` – Banyaknya percobaan yang dilakukan $[n_1, n_2]$
- C. `value` – Nilai yang digunakan untuk H_0
- D. `alternative` – Terdapat tiga pilihan yaitu “two-sided”, “smaller”, atau “larger” bergantung hipotesis alternatif yang kita punya.

Akan digunakan dua data untuk mengilustrasikan uji proporsi dua sampel.

- **Data Karyawan**

Data Karyawan merupakan data keterlambatan karyawan di dua cabang berbeda. Seorang manajer tertarik untuk melihat apakah terdapat perbedaan proporsi pria yang terlambat di dua cabang tersebut. Untuk membuat tabel dengan indeks berupa kode cabang dan kolom berupa jenis kelamin kita gunakan method `crosstab` dari `pandas`.

```
import pandas as pd
from scipy.stats import *

df = pd.read_csv("Downloads/pelatihan/Karyawan.csv",
                 sep = ";")
pd.crosstab(index=df['Kode Cabang'], columns=df['Jenis Kelamin'])
```

Jenis Kelamin	Pria	Wanita
Kode Cabang		
A	11	13
B	12	14

Misal p_A menunjukkan proporsi karyawan pria terlambat di cabang A dan p_B di cabang B. Dari tabel tersebut, diperoleh

$$x_A = 11, x_B = 12$$

$$n_A = 24, n_B = 26$$

Hipotesis yang digunakan adalah

$$H_0: p_A = p_B$$

$$H_a: p_A \neq p_B$$

Selanjutnya kita gunakan `alternative = "two-sided"`. Dengan menggunakan fungsi `proportions_ztest` diperoleh hasil sebagai berikut

```
from statsmodels.stats.proportion import proportions_ztest
proportions_ztest(count = [11,12],
                  nobs = [24,26],
                  value = 0,
                  alternative = "two-sided")

(-0.022718356114153942, 0.9818749335525118)
```

Dari hasil diatas diperoleh $p - value = 0.98 > 0.05$ sehingga tidak cukup data untuk mendukung penolakan H_0 . Artinya . Proporsi karyawan pria yang terlambat identik di kedua cabang pada taraf 5%. Dari *sample estimates*, proporsi karyawan pria cabang A yang terlambat adalah 45.8%, tidak jauh berbeda dari cabang B sebesar 46.2%.

- **Data Pembelajaran Daring**

Seorang peneliti ingin membandingkan efektivitas pembelajaran daring dan luring. Peneliti tersebut memilih 125 siswa dari sebuah kelas berisi 300 siswa untuk belajar secara daring, sisanya dengan tatap muka. Di akhir pembelajaran, seluruh siswa diminta mengisi sebuah kuis dengan hasil yang dirangkum di tabel di bawah.

Hasil	Daring	Luring
Lulus	61	112
Gagal	64	63
Total	125	175

Untuk mengetahui apakah pembelajaran daring menurunkan proporsi siswa yang lulus daripada pembelajaran tatap muka, peneliti mengasumsikan bahwa kedua kelompok siswa saling independen (antara kelompok daring dan luring). Hipotesis yang digunakan adalah sebagai berikut.

Misal p_{Daring} menunjukkan proporsi siswa lulus di kelompok daring p_{Luring} di kelompok luring. Dari tabel tersebut, diperoleh

$$x_{Daring} = 61, x_{Luring} = 112$$

$$n_{Daring} = 125, n_{Luring} = 175.$$

Hipotesis yang digunakan adalah

$$H_0: p_{Daring} = p_{Luring}$$

$$H_a: p_{Daring} < p_{Luring}$$

Akan digunakan tingkat signifikansi $\alpha = 5\%$

Selanjutnya kita gunakan `alternative = "smaller"`. Dengan menggunakan fungsi `proportions_ztest` diperoleh hasil sebagai berikut

```
from statsmodels.stats.proportion import proportions_ztest
proportions_ztest(count = [61,112],
                   nobs = [125,175],
                   value = 0,
                   alternative = "smaller")

(-2.6269592387953256, 0.00430758266569063)
```

Diperoleh nilai $p - value = 0.004 < \alpha = 5\%$, artinya data mendukung penolakan H_0 . Artinya pembelajaran secara daring secara signifikan menurunkan proporsi siswa yang lulus kuis. Peneliti juga perlu menggaris bawahi bahwa hasil ini terbatas pada satu kali pembelajaran dan satu kuis. Pada

kenyataannya, untuk membandingkan efektivitas pembelajaran daring dan luring, diperlukan analisis yang lebih menyeluruh.

Uji Homogenitas (Kesamaan Varian)

Salah satu asumsi untuk menggunakan uji-t adalah kedua populasi harus mempunyai variansi yang sama. Keadaan dimana dua populasi mempunyai variansi yang sama disebut **homogenitas** dua populasi. Misalkan variansi kedua populasi masing-masing adalah σ_1^2 dan σ_2^2 . Hipotesis yang digunakan adalah

$$H_0: \sigma_1^2 = \sigma_2^2$$

$$H_a: \sigma_1^2 \neq \sigma_2^2$$

Salah satu pengujian yang paling sering digunakan untuk mengetahui dua populasi (baik normal atau tidak) adalah uji Levene. Python menyediakan fungsi untuk uji Levene yaitu `levене` yang memberikan statistik uji W dan p – *value* hasil dari uji Levene pada data yang kita masukkan. Fungsi ini dapat diimport dari library `scipy`, dan membutuhkan dua input data (bisa berbeda panjang). Fungsi ini menyediakan parameter `center` yang bisa bernilai “mean”, “median”, “trimmed” sebagai fungsi yang digunakan untuk melakukan pengujian. Karena nilai default dari parameter tersebut adalah median dan seringkali kita menggunakan nilai mean maka kita perlu mengatur nilai `center=“mean”`.

Untuk mengilustrasikan uji Levene, kita akan menggunakan data “TaiwanEstate.csv” dan menguji apakah variansi usia bangunan antara dua periode transaksi (EARLY dan LATE) homogen atau tidak. Perbedaan dalam variansi dapat menandakan bahwa kedua periode tersebut memiliki sebaran usia yang berbeda.

$$H_0: \sigma_{EARLY}^2 = \sigma_{LATE}^2$$

$$H_a: \sigma_{EARLY}^2 \neq \sigma_{LATE}^2$$

Kita gunakan tingkat signifikansi 5%.

```
from scipy.stats import levene
df = pd.read_csv("Downloads/pelatihan/TaiwanEstate.csv",
                 sep = ";")
#Mengambil data usia bangunan dengan periode transaksi bernilai early
early = df["UsiaBangunan"][df["PeriodeTransaksi"]=="EARLY"]
#Mengambil data usia bangunan dengan periode transaksi bernilai late
late = df["UsiaBangunan"][df["PeriodeTransaksi"]=="LATE"]
#Uji levene
levене(early,late,center = "mean")

LeveneResult(statistic=1.6521160954348881,
              pvalue=0.19939320857158294)
```

Nilai $p - value = 0.199 > 5\%$ menunjukkan H_0 tidak ditolak pada tingkat signifikansi 5%. Artinya, variansi usia bangunan kedua periode homogen (sama).

Uji Selisih Mean Dua Sampel Independen, Variansi Populasi Sama

Pada pembahasan kali ini akan diasumsikan kedua populasi saling independen dan berdistribusi normal dengan variansi sama.

- **Estimasi Interval Selisih Mean Dua Populasi**

Interval konfidensi $(1 - \alpha) \times 100\%$ untuk $\mu_1 - \mu_2$ adalah

$$B \leq \mu_1 - \mu_2 \leq A$$

A. Bila σ_1^2 dan σ_2^2 diketahui maka

$$B = (\bar{X}_1 - \bar{X}_2) - Z_{\frac{\alpha}{2}} \sqrt{\frac{\sigma_1^2}{n_1} + \frac{\sigma_2^2}{n_2}}$$

$$A = (\bar{X}_1 - \bar{X}_2) + Z_{\frac{\alpha}{2}} \sqrt{\frac{\sigma_1^2}{n_1} + \frac{\sigma_2^2}{n_2}}$$

B. Bila σ_1^2 dan σ_2^2 tidak diketahui maka

$$B = (\bar{X}_1 - \bar{X}_2) - t_{(n_1+n_2-2, \frac{\alpha}{2})} \sqrt{S_p^2 \left(\frac{1}{n_1} + \frac{1}{n_2} \right)}$$

$$A = (\bar{X}_1 - \bar{X}_2) + t_{(n_1+n_2-2, \frac{\alpha}{2})} \sqrt{S_p^2 \left(\frac{1}{n_1} + \frac{1}{n_2} \right)}$$

Dengan

$$S_p^2 = \frac{s_1^2(n_1 - 1) + s_2^2(n_2 - 1)}{n_1 + n_2 - 2}$$

- **Uji Hipotesis Selisih Mean Dua Populasi**

Jika ingin menguji bahwa mean selisih dua populasi adalah sama dengan nilai tertentu μ_0 , maka langkah-langkah uji hipotesis ini adalah sebagai berikut :

1. Tentukan Hipotesis

A. $H_0 : \mu_1 - \mu_2 = \mu_0$ (uji dua sisi)

$H_a : \mu_1 - \mu_2 \neq \mu_0$

B. $H_0 : \mu_1 - \mu_2 \leq \mu_0$ (uji satu sisi)

$H_a : \mu_1 - \mu_2 > \mu_0$

C. $H_0 : \mu_1 - \mu_2 \geq \mu_0$ (uji satu sisi)

$H_a : \mu_1 - \mu_2 < \mu_0$

2. Tentukan tingkat signifikansi α

3. Statistik Penguji

A. Bila σ_1^2 dan σ_2^2 diketahui maka

$$Z = \frac{(\bar{X}_1 - \bar{X}_2) - (\mu_1 - \mu_2)}{\sqrt{\frac{\sigma_1^2}{n_1} + \frac{\sigma_2^2}{n_2}}}$$

B. Bila σ_1^2 dan σ_2^2 tidak diketahui maka

$$t = \frac{(\bar{X}_1 - \bar{X}_2) - (\mu_1 - \mu_2)}{\sqrt{S_p^2 \left(\frac{1}{n_1} + \frac{1}{n_2} \right)}}$$

Dengan

$$S_p^2 = \frac{s_1^2(n_1 - 1) + s_2^2(n_2 - 1)}{n_1 + n_2 - 2}$$

4. Daerah kritis : H_0 ditolak berturut pada kasus A dan B bila :

A. $Z < -Z_{\frac{\alpha}{2}}$ atau $Z > Z_{\frac{\alpha}{2}}$ dan $t < -t_{(n_1+n_2-2, \frac{\alpha}{2})}$ atau $t > t_{(n_1+n_2-2, \frac{\alpha}{2})}$

B. $Z > Z_{\alpha}$ dan $t > t_{(n_1+n_2-2, \alpha)}$

C. $Z < -Z_{\alpha}$ dan $t < -t_{(n_1+n_2-2, \alpha)}$

5. Berdasarkan langkah 4 dan hasil penghitungan statistic penguji langkah 3, diambil kesimpulan apakah H_0 ditolak atau tidak ditolak pada tingkat signifikansi α .

Komputasi Uji Selisih Mean Dua Sampel Independen, Variansi Populasi Sama Dengan Python

Akan diberikan ilustrasi uji hipotesis selisih mean dua sampel dari populasi yang mempunyai variansi sama menggunakan Python. Pada Python tersedia fungsi `ttest_ind` untuk menghitung statistik uji dan $p - value$ dari uji-t. Untuk menggunakan fungsi ini kita masukkan dua sampel sebagai input, dan memberikan nilai `equal_var = True`. Terdapat 3 jenis hipotesis alternatif sebagai nilai pada parameter `alternative`.

A. “two-sided” untuk $H_a : \mu_1 - \mu_2 \neq \mu_0$

B. “greater” untuk $H_a : \mu_1 - \mu_2 > \mu_0$

C. “less” untuk $H_a : \mu_1 - \mu_2 < \mu_0$

Akan digunakan data TaiwanEstate untuk mengilustrasikan uji hipotesis selisih mean dua sampel dari populasi yang berdistribusi normal.

- **Data TaiwanEstate**

Peneliti ingin mengetahui apakah rata-rata harga bangunan di dua periode transaksi (EARLY dan LATE) berbeda secara signifikan. Dari data historis, peneliti menemukan bahwa harga properti umumnya meningkat di awal tahun karena adanya CNY.

Uji Normalitas

Mula-mula akan dicek apakah kedua sampel memenuhi asumsi berdistribusi normal. Akan dilakukan uji normalitas Shapiro – Wilk pada masing-masing sampel dengan hipotesis

H_0 : Harga rata-rata bangunan berasal dari populasi yang berdistribusi Normal

H_a : Harga rata-rata bangunan tidak berasal dari populasi yang berdistribusi Normal

Akan digunakan tingkat signifikansi 5%.

Selanjutnya akan digunakan bantuan Python untuk mendapatkan $p - value$ dari uji Shapiro-Wilk pada variabel “Harga” pada masing-masing sampel.

```
import pandas as pd
from scipy.stats import *

df = pd.read_csv("Downloads/pelatihan/TaiwanEstate.csv",
                 sep = ";")
early = df["Harga"][df["PeriodeTransaksi"] == "EARLY"]
late = df["Harga"][df["PeriodeTransaksi"] == "LATE"]

print("Early : ", shapiro(early))
print("Late : ", shapiro(late))

Early : ShapiroResult(statistic=0.9928848052031055,
                      pvalue=0.23739937240361064)
Late : ShapiroResult(statistic=0.9905142634760397,
                     pvalue=0.41657902979152084)
```

Diperoleh pada masing-masing sampel hasil $p - value > 5\%$ sehingga H_0 tidak ditolak pada tingkat signifikansi 5% (kedua populasi berdistribusi normal)

Uji Homogenitas (Levene)

Perbedaan dalam variansi dapat menandakan bahwa kedua periode tersebut memiliki sebaran harga yang berbeda. Akan diuji apakah kedua variansi sampel adalah sama.

$$H_0: \sigma_{EARLY}^2 = \sigma_{LATE}^2$$

$$H_a: \sigma_{EARLY}^2 \neq \sigma_{LATE}^2$$

Kita gunakan tingkat signifikansi 5%.

```
from scipy.stats import levene
#Uji levene
levene(early,late,center = "mean")

LeveneResult(statistic=0.15236962136322293,
              pvalue=0.6964828089329801)
```


Nilai $p - value = 0.696 > 5\%$ menunjukkan H_0 tidak ditolak pada tingkat signifikansi 5%. Artinya, variansi harga pada kedua periode transaksi homogen

Uji t Dua Sampel Independen

1. Menentukan hipotesis.

Misal dimiliki μ_{EARLY}, μ_{LATE} sebagai rata-rata masing-masing populasi. Hipotesis yang diajukan adalah

$$H_0: \mu_{EARLY} \leq \mu_{LATE}$$

$$H_a: \mu_{EARLY} > \mu_{LATE}$$

2. Akan digunakan tingkat signifikansi $\alpha = 5\%$.
3. Karena variansi populasi tidak diketahui namun sama maka akan dilakukan uji-t. Akan digunakan bantuan Python untuk menentukan $p - value$ dari uji-t.

```
from scipy.stats import ttest_ind
ttest_ind(early,
          late,
          alternative = "greater",
          equal_var = True)

TtestResult(statistic=1.3143081573849655,
            pvalue=0.09473687615160244, df=412.0)
```

4. Kesimpulan. Diperoleh $p - value = 0.0947 > 0.05$ sehingga data belum mendukung penolakan H_0 . Namun dengan menggunakan $\alpha = 10\%$ diperoleh H_0 ditolak, pemilihan ini bergantung pada preferensi peneliti mengenai model yang dia buat. Dengan demikian, rata-rata harga bangunan dengan periode EARLY lebih dari harga bangunan dengan periode LATE pada taraf signifikansi 10%.

Uji Selisih Mean Dua Sampel Independen, Variansi Populasi Berbeda (Uji Welch)

Pada pembahasan kali ini akan diasumsikan kedua populasi saling independen dan berdistribusi normal dengan variansi berbeda.

- **Estimasi Interval Selisih Mean Dua Populasi**

Interval konfidensi $(1 - \alpha) \times 100\%$ untuk $\mu_1 - \mu_2$ adalah

$$B \leq \mu_1 - \mu_2 \leq A$$

A. Bila σ_1^2 dan σ_2^2 diketahui maka

$$B = (\bar{X}_1 - \bar{X}_2) - Z_{\frac{\alpha}{2}} \sqrt{\frac{\sigma_1^2}{n_1} + \frac{\sigma_2^2}{n_2}}$$

$$A = (\bar{X}_1 - \bar{X}_2) + Z_{\frac{\alpha}{2}} \sqrt{\frac{\sigma_1^2}{n_1} + \frac{\sigma_2^2}{n_2}}$$

B. Bila σ_1^2 dan σ_2^2 tidak diketahui maka

$$B = (\bar{X}_1 - \bar{X}_2) - t_{(v, \frac{\alpha}{2})} \sqrt{\left(\frac{S_1^2}{n_1} + \frac{S_2^2}{n_2}\right)}$$

$$A = (\bar{X}_1 - \bar{X}_2) - t_{(v, \frac{\alpha}{2})} \sqrt{\left(\frac{S_1^2}{n_1} + \frac{S_2^2}{n_2}\right)}$$

Dengan

$$v = \frac{\left(\frac{s_1^2}{n_1} + \frac{s_2^2}{n_2}\right)^2}{\frac{\left(\frac{s_1^2}{n_1}\right)^2}{n_1 - 1} + \frac{\left(\frac{s_2^2}{n_2}\right)^2}{n_2 - 1}}$$

- **Uji Hipotesis Selisih Mean Dua Populasi**

Jika ingin menguji bahwa mean selisih dua populasi adalah sama dengan nilai tertentu μ_0 , maka langkah-langkah uji hipotesis ini adalah sebagai berikut :

1. Tentukan Hipotesis

A. $H_0 : \mu_1 - \mu_2 = \mu_0$ (uji dua sisi)

$H_a : \mu_1 - \mu_2 \neq \mu_0$

B. $H_0 : \mu_1 - \mu_2 \leq \mu_0$ (uji satu sisi)

$H_a : \mu_1 - \mu_2 > \mu_0$

C. $H_0 : \mu_1 - \mu_2 \geq \mu_0$ (uji satu sisi)

$H_a : \mu_1 - \mu_2 < \mu_0$

2. Tentukan tingkat signifikansi α

3. Statistik Penguji

A. Bila σ_1^2 dan σ_2^2 diketahui maka

$$Z = \frac{(\bar{X}_1 - \bar{X}_2) - (\mu_1 - \mu_2)}{\sqrt{\frac{\sigma_1^2}{n_1} + \frac{\sigma_2^2}{n_2}}}$$

B. Bila σ_1^2 dan σ_2^2 tidak diketahui maka

$$t = \frac{(\bar{X}_1 - \bar{X}_2) - (\mu_1 - \mu_2)}{\sqrt{\left(\frac{S_1^2}{n_1} + \frac{S_2^2}{n_2}\right)}}$$

4. Daerah kritis : H_0 ditolak berturut pada kasus A dan B bila :

D. $Z < -Z_{\frac{\alpha}{2}}$ atau $Z > Z_{\frac{\alpha}{2}}$ dan $t < -t_{(v, \frac{\alpha}{2})}$ atau $t > t_{(v, \frac{\alpha}{2})}$

E. $Z > Z_{\alpha}$ dan $t > t_{(v, \alpha)}$

F. $Z < -Z_{\alpha}$ dan $t < -t_{(v, \alpha)}$

5. Berdasarkan langkah 4 dan hasil penghitungan statistic penguji langkah 3, diambil kesimpulan apakah H_0 ditolak atau tidak ditolak pada tingkat signifikansi α .

Komputasi Uji Selisih Mean Dua Sampel Independen, Variansi Populasi Berbeda (Uji Welch) Dengan Python

Akan diberikan ilustrasi uji hipotesis selisih mean dua sampel dari populasi yang mempunyai variansi berbeda menggunakan Python. Pada Python tersedia fungsi `ttest_ind` untuk menghitung statistik uji dan p – value dari uji-t. Pada kasus ini untuk menggunakan fungsi `ttest_ind` kita masukkan dua sampel sebagai input, dan memberikan nilai `equal_var = False`. Terdapat 3 jenis hipotesis alternatif sebagai nilai pada parameter `alternative`.

- A. “two-sided” untuk $H_a : \mu_1 - \mu_2 \neq \mu_0$
- B. “greater” untuk $H_a : \mu_1 - \mu_2 > \mu_0$
- C. “less” untuk $H_a : \mu_1 - \mu_2 < \mu_0$

Akan digunakan data cars untuk mengilustrasikan uji hipotesis selisih mean dua sampel dari populasi yang berdistribusi normal dengan variansi berbeda

- **Data cars**

Peneliti ingin mengetahui apakah rata-rata penggunaan bahan bakar (mpg) pada mobil dengan jenis transmisi manual dan automatic berbeda secara signifikan.

Uji Normalitas

Mula-mula akan dicek apakah kedua sampel memenuhi asumsi berdistribusi normal. Akan dilakukan uji normalitas Shapiro – Wilk pada masing-masing sampel dengan hipotesis

H_0 : penggunaan bahan bakar berasal dari populasi yang berdistribusi Normal

H_a : penggunaan bahan bakar tidak berasal dari populasi yang berdistribusi Normal

Akan digunakan tingkat signifikansi 5%.

Selanjutnya akan digunakan bantuan Python untuk mendapatkan p – value dari uji Shapiro-Wilk pada variabel “mpg” pada masing-masing sampel.

```
import pandas as pd
from scipy.stats import *

df = pd.read_csv("Downloads/pelatihan/cars.csv")

auto = df["mpg"][df["am"] == 0]
manual = df["mpg"][df["am"] == 1]

print("Automatic : ", shapiro(auto))
print("Manual : ", shapiro(manual))

Automatic :  ShapiroResult(statistic=0.9767742647710409,
                           pvalue=0.8987357901905731)
Manual :    ShapiroResult(statistic=0.9458036604933976,
                           pvalue=0.536272885248448)
```

Diperoleh pada masing-masing sampel hasil $p - value > 5\%$ sehingga H_0 tidak ditolak pada tingkat signifikansi 5% (kedua populasi berdistribusi normal)

Uji Homogenitas (Levene)

Perbedaan dalam variansi dapat menandakan bahwa kedua periode tersebut memiliki sebaran harga yang berbeda. Akan diuji apakah kedua variansi sampel adalah sama.

$$H_0: \sigma_{auto}^2 = \sigma_{manual}^2$$

$$H_a: \sigma_{auto}^2 \neq \sigma_{manual}^2$$

Kita gunakan tingkat signifikansi 5%.

```
from scipy.stats import levene

#Uji levene
levene(auto,manual,center = "mean")

LeveneResult(statistic=5.920953829161345,
              pvalue=0.021133448421855064)
```

Nilai $p - value = 0.02113 < 5\%$ menunjukkan H_0 ditolak pada tingkat signifikansi 5%. Artinya, variansi penggunaan bahan bakar berdasarkan transmisi tidak sama sehingga tidak bisa digunakan uji yang sama dengan contoh sebelumnya. Oleh karena itu kita gunakan uji Welch untuk kasus ini.

Uji t Dua Sampel Independen Variansi berbeda

1. Menentukan hipotesis.

Misal dimiliki μ_{EARLY}, μ_{LATE} sebagai rata-rata masing-masing populasi. Hipotesis yang diajukan adalah

$$H_0: \mu_{auto} = \mu_{manual}$$

$$H_a: \mu_{auto} \neq \mu_{manual}$$

2. Akan digunakan tingkat signifikansi $\alpha = 5\%$.
3. Akan digunakan bantuan Python untuk menentukan $p - value$ dari uji-Welch.

```
from scipy.stats import ttest_ind
ttest_ind(auto,
          manual,
          alternative = "two-sided",
          equal_var = False)

TtestResult(statistic=-3.767123145144923,
            pvalue=0.0013736383330710345,
            df=18.332251638400464)
```

4. Kesimpulan. Diperoleh $p - value = 0.001374 < 0.05$ sehingga data mendukung penolakan H_0 . Dengan demikian, rata-rata penggunaan bahan bakar mobil akan berbeda secara signifikan berdasarkan jenis transmisinya.

Uji Selisih Mean Dua Sampel Berpasangan dari Populasi Normal (Paired t-test)

Dimiliki sampel random berupa pasangan data $(X_1, Y_1), (X_2, Y_2), \dots, (X_n, Y_n)$. Ingin dilakukan inferensi terkait μ_D (rata-rata selisih) tiap pasangan dari populasi, berdasarkan statistik $D_i = X_i - Y_i$. Dalam hal ini meskipun antara pasangan (X_i, Y_i) independen untuk semua $i = 1, 2, \dots, n$ namun X_i dan Y_i sendiri tidaklah independen karena Y_i diambil dari individu/objek yang sama terhadap X_i . Inferensi dua populasi pada pembahasan sebelumnya tidaklah cocok untuk data tipe ini. Dengan menganggap D_i adalah sampel random yang berasal dari distribusi normal dapat didefinisikan statistik t .

$$t = \frac{\bar{D} - \mu_D}{S_D / \sqrt{n}}$$

Dengan $\bar{D} = \frac{1}{n} \sum_{i=1}^n D_i$ dan $S_D = \frac{1}{n-1} \sum_{i=1}^n (D_i - \bar{D})^2$.

Interval konfidensi $(1 - \alpha) \times 100\%$ untuk μ_D adalah

$$B \leq \mu_D \leq A$$

Dengan

$$B = \bar{D} - t_{(n-1, \frac{\alpha}{2})} \frac{S_D}{\sqrt{n}}$$

$$A = \bar{D} + t_{(n-1, \frac{\alpha}{2})} \frac{S_D}{\sqrt{n}}$$

Ingin diuji suatu hipotesis bahwa mean selisih dua populasi μ_D sama dengan suatu nilai tertentu yaitu μ_0 . Langkah-langkah uji hipotesis dapat dirangkum sebagai berikut :

1. Tentukan Hipotesis

A. $H_0 : \mu_D = \mu_0$ (uji dua sisi)

$H_a : \mu_D \neq \mu_0$

B. $H_0 : \mu_D \leq \mu_0$ (uji satu sisi)

$H_a : \mu_D > \mu_0$

C. $H_0 : \mu_D \geq \mu_0$ (uji satu sisi)

$H_a : \mu_D < \mu_0$

2. Tentukan tingkat signifikansi α

3. Statistik Penguji

$$t = \frac{\bar{D} - \mu_D}{S_D/\sqrt{n}}$$

4. Daerah kritis : H_0 ditolak berturut pada kasus A dan B bila :

A. $t < -t_{(n-1, \frac{\alpha}{2})}$ atau $t > t_{(n-1, \frac{\alpha}{2})}$

B. $t > t_{(n-1, \alpha)}$

C. $t < -t_{(n-1, \alpha)}$

5. Berdasarkan langkah 4 dan hasil penghitungan statistic pengujian langkah 3, diambil kesimpulan apakah H_0 ditolak atau tidak ditolak pada tingkat signifikansi α .

Komputasi Uji Selisih Mean Dua Sampel Berpasangan dari Populasi Normal (Paired t-test) dengan Python

Akan diberikan ilustrasi uji hipotesis selisih mean dua sampel berpasangan menggunakan Python. Pada Python tersedia fungsi `ttest_rel` untuk menghitung statistik uji dan p – value dari uji- Paired t . Untuk menggunakan fungsi `ttest_rel` kita masukkan dua sampel sebagai input Terdapat 3 jenis hipotesis alternatif sebagai nilai pada parameter `alternative` .

A. “two-sided” untuk $H_a : \mu_1 \neq \mu_2$

B. “greater” untuk $H_a : \mu_1 > \mu_2$

C. “less” untuk $H_a : \mu_1 < \mu_2$

Akan digunakan data Kabezine untuk mengilustrasikan uji hipotesis selisih mean dua sampel berpasangan dari populasi yang berdistribusi normal.

• Data : Kabezine

Seorang peneliti tertarik untuk menguji pengaruh dari suatu zat bernama “Kabezine” yang diberikan kepada beberapa volunteer. Peneliti ingin mengetahui apakah nilai yang diperoleh tiap volunteer berubah setelah 30 menit. Dataset ini tersedia dan terdapat kolom “Pre-Treatment” dan “Post-Treatment” (indikator sampel pertama atau kedua).

Uji Normalitas

Mula-mula akan dicek apakah kedua sampel memenuhi asumsi berdistribusi normal. Akan dilakukan uji normalitas Shapiro – Wilk pada masing-masing sampel dengan hipotesis

H_0 : Nilai volunteer berasal dari populasi yang berdistribusi Normal

H_a : Nilai volunteer tidak berasal dari populasi yang berdistribusi Normal

Akan digunakan tingkat signifikansi 5%.

Selanjutnya akan digunakan bantuan Python untuk mendapatkan p – value dari uji Shapiro-Wilk pada masing-masing sampel.

```
import pandas as pd
from scipy.stats import *

df = pd.read_csv("Downloads/pelatihan/Kabezine.csv",
                 sep = ";")
```

```
pre = df["Pre-Treatment"]
post = df["Post-Treatment"]

print("Pre : ", shapiro(pre))
print("Post : ", shapiro(post))

Pre : ShapiroResult(statistic=0.9718993759657798,
                    pvalue=0.79438254969045)
Post : ShapiroResult(statistic=0.9453484962193525,
                    pvalue=0.3019354589712213)
```

Diperoleh pada masing-masing sampel hasil $p - value > 5\%$ sehingga H_0 tidak ditolak pada tingkat signifikansi 5% (kedua populasi berdistribusi normal).

Uji t Dua Sampel Berpasangan

1. Menentukan hipotesis.

Misal dimiliki μ_{Pre}, μ_{Post} sebagai rata-rata masing-masing populasi. Hipotesis yang diajukan adalah

$$H_0: \mu_{Pre} = \mu_{Post}$$

$$H_a: \mu_{Pre} \neq \mu_{Post}$$

2. Akan digunakan tingkat signifikansi $\alpha = 5\%$.
3. Akan digunakan bantuan Python untuk menentukan $p - value$ dari uji-Paired t.

```
from scipy.stats import ttest_rel

ttest_rel(pre,
          post,
          alternative = "two-sided",
          )

TtestResult(statistic=-3.443304412176149,
            pvalue=0.002723932312222769,
            df=19)
```

4. Kesimpulan. Diperoleh $p - value = 0.0027 < 0.05$ sehingga data mendukung penolakan H_0 . Dengan demikian pemberian Kabezine akan mempengaruhi nilai yang diperoleh oleh penerimanya secara signifikan.

BAB V

ANALISIS REGRESI

Di kehidupan sehari-hari kita seringkali menemukan keterkaitan antara beberapa hal. Jika ada orang yang tinggi maka biasanya dia mempunyai orang tua yang tinggi dan orang yang pendek biasanya orang tuanya juga pendek. Sekilas terdapat hubungan antara tinggi anak dan tinggi orang tua, namun tidak jarang juga orang tua yang tinggi mempunyai anak pendek dan seterusnya. Kita tertarik untuk melihat apakah ada hubungan antara tinggi orang tua dan tinggi anak.

Tentu saja kita tidak mungkin mengumpulkan semua data tinggi orang tua beserta anaknya yang ada di bumi. Jika kita dapat menentukan korelasi antara dua variabel, maka kita dapat membuat model regresi dari kedua variabel tersebut. Pembahasan pertama kita adalah membuat kuantifikasi dari korelasi dua variabel.

5.1 Korelasi Pearson

Salah satu cara untuk melihat hubungan antara dua variabel random adalah menghitung kovariannya. Misalkan kita mempunyai sampel $x = (x_1, x_2, \dots, x_n)$ dan $y = (y_1, y_2, \dots, y_n)$ maka kovarian dua data tersebut adalah

$$Cov(x, y) = \frac{1}{n} \sum_{i=1}^n (x_i - \bar{x})(y_i - \bar{y})$$

Jika $Cov(x, y) > 0$ maka dua variabel tersebut bergerak ke arah yang sama (semakin besar x maka semakin besar y), jika $Cov(x, y) < 0$ maka dua variabel tersebut bergerak ke arah berlawanan (semakin besar x maka semakin kecil y), dan jika $Cov(x, y) = 0$ maka tidak ada relasi antara x dan y .

Untuk selanjutnya diasumsikan x dan y **berdistribusi normal**. Diperoleh **koefisien korelasi Pearson**

$$\rho = \frac{Cov(x, y)}{\sigma_x \sigma_y}.$$

Diperhatikan bahwa $-1 \leq \rho \leq 1$. Kita ingin menguji signifikansi dari koefisien ρ yang diperoleh, diajukan hipotesis berikut

H_0 : Korelasi dua variabel adalah nol ($\rho = 0$)

H_a : Korelasi dua variabel tidak nol ($\rho \neq 0$)

Diperoleh statistik uji-t

$$t = \rho \frac{\sqrt{n-2}}{\sqrt{1-\rho^2}}$$

Berdistribusi t dengan derajat kebebasan $n - 2$.

Perlu diingat bahwa koefisien korelasi hanya mengukur adanya hubungan linear antar variabel. Seringkali kita juga menemukan dua variabel yang mempunyai hubungan namun tidak linear sehingga koefisien korelasi tidak cukup bagus. Selain itu, **adanya korelasi antar variabel belum tentu menunjukkan hubungan kausal (sebab-akibat).**

5.2 Komputasi Korelasi Pearson dengan Python

Untuk menguji koefisien korelasi Pearson dengan menggunakan Python dapat menggunakan fungsi `pearsonr` dari `scipy.stats`. Kita cukup memasukkan dua variabel yang ingin kita uji korelasi Pearson.

Sebagai ilustrasi digunakan data `TaiwanEstate`. Peneliti menduga bahwa semakin jauh jarak stasiun MRT, harga sebuah properti akan semakin murah, dan sebaliknya. Hal ini karena moda transportasi umum menjadi perhatian khusus bagi penduduk Taiwan.

```
import pandas as pd
import numpy as np
from scipy.stats import pearsonr

df = pd.read_csv("Downloads/pelatihan/TaiwanEstate.csv",
                 sep = ";")

pearsonr(df["JarakStasiunMRT"], df["Harga"])

PearsonRResult(statistic=-0.6833531599150267,
               pvalue=2.9442533862631152e-58)
```

Dari hasil diatas diperoleh $\rho = -0.683$ (korelasi kuat). Tanda negatif menunjukkan bahwa semakin tinggi jarak stasiun MRT, semakin rendah harga properti (sesuai dugaan peneliti). Nilai $p - value < \alpha = 5\%$ mengindikasikan korelasi signifikan antara jarak stasiun MRT dan harga property.

5.3 Regresi Linear

Sebelum membahas mengenai regresi linear, perlu diketahui istilah yang dimiliki oleh variabel-variabel pada data. **Variabel independen (bebas, prediktor, X)** adalah variabel yang dianggap tidak dipengaruhi oleh variabel lainnya. **Variabel dependen (terikat, respon, Y)** adalah variabel yang dianggap dipengaruhi variabel lainnya.

Selanjutnya diberikan beberapa asumsi yang harus dipenuhi untuk dapat melakukan regresi linear pada dua variabel.

1. Variabel respon harus berdistribusi normal.
2. Dua variabel mempunyai hubungan yang linear.
3. Setiap observasi independen satu sama lain. Asumsi ini sulit untuk dibuktikan, namun secara umum apabila pengambilan data dilakukan dengan sampel acak maka asumsi ini dapat dianggap dipenuhi.

Diberikan n observasi dengan variabel dependen $\mathbf{y} = (y_1, y_2, \dots, y_n)$ dan variabel independen $\mathbf{x}_i = (x_{i1}, x_{i2}, \dots, x_{iq})$ dengan $i = 1, 2, \dots, n$. Pada model regresi linear setiap y_i dinyatakan sebagai kombinasi linear \mathbf{x}_i ditambah random error ϵ_i sehingga model kita dapat dituliskan sebagai

$$\mathbf{y} = \mathbf{X}\boldsymbol{\beta} + \boldsymbol{\epsilon}$$

Dengan $\boldsymbol{\beta} = (\beta_0, \beta_1, \dots, \beta_q)^T$ dan $\mathbf{X} = (x_{ij}), 0 \leq i \leq n, 1 \leq j \leq q$ dengan $x_{0j} = 1$ untuk setiap j . Vektor $\boldsymbol{\beta}$ merupakan koefisien regresi yang ingin dicari dan $\boldsymbol{\epsilon}$ menyatakan residual yang diasumsikan independen, identik dan berdistribusi normal. Jika $q = 1$ maka model diatas disebut sebagai **regresi linear sederhana**, dan untuk $q > 1$ disebut **regresi berganda**

Salah satu cara untuk mencari koefisien regresi $\boldsymbol{\beta}$ meminimumkan jumlahan kuadrat residual

$$SSR = \epsilon^2 = (\mathbf{y} - \mathbf{X}\boldsymbol{\beta})^T (\mathbf{y} - \mathbf{X}\boldsymbol{\beta})$$

Dengan memanfaatkan uji turunan diperoleh koefisien regresi yang meminimumkan SSR adalah

$$\hat{\boldsymbol{\beta}} = (\mathbf{X}^T \mathbf{X})^{-1} \mathbf{X}^T \mathbf{Y}$$

Persamaan diatas disebut sebagi persamaan normal dari model regresi dan metode ini disebut sebagai **Ordinary Least Squares (OLS)**.

5.4 Komputasi Regresi Linear Sederhana Menggunakan Python

Untuk menghitung koefisien regresi dengan menggunakan Python kita dapat menggunakan fungsi `ols` yang disediakan oleh module `statsmodels.formula.api`. Secara umum sintaks untuk melakukan regresi linear sederhana pada variabel dependen Y dan variabel independen X pada dataframe “df” adalah sebagai berikut

$$\text{ols}(\text{“Y ~ X”, data = df})$$

Sebagai ilustrasi akan digunakan data “Harga” sebagai variabel dependen dan “JarakStasiunMRT” sebagai variabel independen pada data “TaiwanEstate”.

Uji Normalitas

Mula-mula akan dicek apakah variabel “Harga” memenuhi asumsi berdistribusi normal. Akan dilakukan uji normalitas Shapiro – Wilk dengan hipotesis

H_0 : Harga bangunan berasal dari populasi yang berdistribusi Normal

H_a : Harga bangunan tidak berasal dari populasi yang berdistribusi Normal

Akan digunakan tingkat signifikansi 5%.

Selanjutnya akan digunakan bantuan Python untuk mendapatkan p – value dari uji Shapiro-Wilk.

```
import pandas as pd
import numpy as np
from scipy.stats import *

df = pd.read_csv("Downloads/pelatihan/TaiwanEstate.csv",
                 sep = ";")
shapiro(df["Harga"])

ShapiroResult(statistic=0.994888889142379,
               pvalue=0.18744587664238638)
```

Diperoleh hasil p – value = 0.187 > 5% sehingga H_0 tidak ditolak pada tingkat signifikansi 5% (Harga bangunan berasal dari populasi berdistribusi normal).

Regresi Linear

Untuk dapat menggunakan fungsi `ols` kita import terlebih dahulu module `statsmodels.formula.api` selanjutnya kita definisikan variabel dependen dan variabel independen yang ingin kita regresikan.

```
import statsmodels.formula.api as smf
#Membangun model OLS
lr = smf.ols("Harga ~ JarakStasiunMRT", data = df).fit()

#Memunculkan hasil dari fitting model
lr.summary()
```

OLS Regression Results			
Dep. Variable:	Harga	R-squared:	0.467
Model:	OLS	Adj. R-squared:	0.466
Method:	Least Squares	F-statistic:	360.9
Date:	Tue, 16 Apr 2024	Prob (F-statistic):	2.94e-58
Time:	02:21:49	Log-Likelihood:	-2011.1
No. Observations:	414	AIC:	4026.
Df Residuals:	412	BIC:	4034.
Df Model:	1		
Covariance Type:	nonrobust		

Tabel 1

	coef	std err	t	P> t	[0.025	0.975]
Intercept	370.5155	2.024	183.054	0.000	366.537	374.494
JarakStasiunMRT	-0.0231	0.001	-18.998	0.000	-0.026	-0.021

Tabel 2

Omnibus:	32.856	Durbin-Watson:	0.606
Prob(Omnibus):	0.000	Jarque-Bera (JB):	57.431
Skew:	0.505	Prob(JB):	3.38e-13
Kurtosis:	4.520	Cond. No.	2.19e+03

Tabel 3

Perhatikan bahwa `lr = smf.ols("Harga ~ JarakStasiunMRT", data = df).fit()` mempunyai arti kita menggunakan kolom “Harga” sebagai variabel dependen dan variabel “JarakStasiunMRT” sebagai variabel independen (urutan ini tidak boleh ditukar). Kita memperoleh tiga tabel hasil regresi linear dua variabel tersebut. Berikut arti dari masing-masing nilai.

1. Tabel 1 : Koefisien determinasi

Tabel 1 dibagi menjadi dua kolom. Kolom pertama menyatakan deskripsi dari model yang dibangun.

- **Dep. Variable** : Informasi tentang apa yang menjadi variabel target.
- **Model** : Informasi tentang model yang telah disesuaikan, OLS adalah metode kuadrat terkecil biasa, istilah lain dari regresi linier.
- **Method** : Metode penyesuaian parameter (dalam hal ini kuadrat terkecil, metode perhitungan klasik)
- **No. Observations** : Jumlah pengamatan yang telah digunakan.
- **DF Residuals** : Derajat kebebasan residual, yaitu jumlah pengamatan dikurangi jumlah parameter.
- **DF Model**: Jumlah parameter yang diestimasi dalam model (tidak termasuk suku konstan dalam penghitungan).

Kolom kedua fokus untuk memberikan informasi seberapa bagus model regresi linear yang telah disesuaikan.

- **R-squared** : Koefisien determinasi, ukuran seberapa baik regresi berkinerja dibandingkan dengan mean sederhana. Model yang baik adalah model dengan R-squared tinggi.
- **Adj. R-squared** : Koefisien determinasi yang disesuaikan berdasarkan jumlah parameter dalam model dan jumlah observasi yang membantu membangunnya. Model yang baik adalah model dengan Adjusted R-squared tinggi
- **F-statistic** : Ukuran yang memberi informasi, semua koefisien yang diperoleh berbeda dari nol. Secara sederhana, ini memberikan informasi apakah regresi yang dilakukan benar-benar lebih baik daripada mean sederhana.
- **Prob (F-statistic)** : Ini adalah probabilitas bahwa F-statistik yang didapatkan tersebut hanya oleh kebetulan karena observasi yang digunakan (probabilitas seperti itu sebenarnya

disebut nilai p dari F-statistik). Jika cukup rendah, Kita yakin bahwa regresi yang dilakukan benar-benar lebih baik daripada mean sederhana.

- **AIC** : Ini adalah **Akaike Information Criterion**. AIC adalah skor yang mengevaluasi model berdasarkan jumlah observasi dan kompleksitas model itu sendiri. Semakin rendah skor AIC, semakin baik. Ini sangat berguna untuk membandingkan model yang berbeda dan untuk seleksi variabel statistik. Model yang baik adalah model dengan AIC rendah,
- **BIC** : Ini adalah **Bayesian Information Criterion**. Ini berfungsi seperti AIC, tetapi memberikan penalti yang lebih tinggi untuk model dengan lebih banyak parameter. Model yang baik adalah model dengan BIC rendah.

Untuk regresi linear sederhana hanya dua nilai pengukuran saja yang berguna yaitu **F-statistic** dan **R-squared**.

2. Tabel 2 : Signifikansi Koefisien

Tabel kedua memberikan informasi mengenai koefisien hasil regresi.

- **coef** : Koefisien hasil estimasi
- **std err** : Kesalahan standar dari estimasi koefisien; semakin besar nilainya, semakin tidak pasti estimasi koefisien tersebut
- **t** : Nilai statistik t, sebuah ukuran yang menunjukkan apakah nilai sebenarnya dari koefisien berbeda dari nol.
- **P > |t|** : *p – value* yang menunjukkan probabilitas bahwa koefisien berbeda dari nol.
- **[95.0% Conf. Interval]** : Nilai bawah dan atas dari koefisien, mempertimbangkan 95% dari semua kemungkinan dari observasi yang berbeda sehingga koefisien yang diestimasi pun berbeda.

Dari tabel ini diperoleh persamaan hasil regresi

$$\widehat{Harga} = 370.5155 - 0.0231 * JarakStasiunMRT$$

Dengan masing-masing koefisien signifikan terhadap persamaan. Persamaan diatas mengartikan bahwa kenaikan satu unit JarakStasiunMRT mengakibatkan Harga turun sebesar 0.0231.

3. Tabel 3 : Mengevaluasi nilai hasil penyesuaian

Tabel terakhir berkaitan dengan analisis residu dari regresi. Residu menggambarkan perbedaan antara nilai target sebenarnya dan nilai yang diprediksi.

- **Skewness** : Sebuah ukuran tentang seberapa simetris distribusi dari residu di sekitar rata-rata. Untuk distribusi residu yang simetris, nilai seharusnya mendekati nol. Nilai positif menunjukkan ekor panjang ke kanan; nilai negatif menunjukkan ekor panjang ke kiri.

- **Kurtosis** : Sebuah ukuran tentang bentuk distribusi dari residu. Distribusi yang berbentuk lonceng memiliki nilai nol. Nilai negatif menunjukkan distribusi yang terlalu datar; nilai positif menunjukkan puncak yang terlalu tinggi.
- **Omnibus D'Angostino's test** : Sebuah uji statistik gabungan untuk menilai skewness dan kurtosis.
- **Prob(Omnibus)**: Statistik Omnibus yang diubah menjadi probabilitas.
- **Jarque-Bera** : Sebuah uji lain untuk menilai skewness dan kurtosis.
- **Prob(JB)**: Statistik JB yang diubah menjadi probabilitas.
- **Durbin-Watson** : Sebuah uji untuk mengetahui keberadaan korelasi di antara residu (berkaitan saat menganalisis data berbasis waktu).
- **Cond. No** : Sebuah uji untuk mendeteksi multikolinearitas.

5.5 Komputasi Regresi Linear Berganda dengan Python

Ketika kita mempunyai satu variabel dependen yang dijelaskan oleh lebih dari satu variabel independen maka kita dapat memodelkan regresi linear berganda. Dengan kata lain kita mencari $\beta_0, \beta_1, \dots, \beta_q$ yang memenuhi

$$y_1 = \beta_0 + \beta_1 x_{11} + \beta_2 x_{12} + \dots + \beta_q x_{1q}$$

$$y_2 = \beta_0 + \beta_1 x_{21} + \beta_2 x_{22} + \dots + \beta_q x_{2q}$$

$$y_3 = \beta_0 + \beta_1 x_{31} + \beta_2 x_{32} + \dots + \beta_q x_{3q}$$

⋮

$$y_n = \beta_0 + \beta_1 x_{n1} + \beta_2 x_{n2} + \dots + \beta_q x_{nq}$$

Tahapan dari analisis regresi linear berganda adalah sebagai berikut.

1. **Uji asumsi Klasik.** Lakukan uji asumsi Normalitas dan Linearitas
2. **Pemodelan Regresi.** Dengan menggunakan fungsi `ols` yang disediakan oleh module `statsmodels.formula.api` kita bisa mengestimasi nilai-nilai beta diatas. Yang menjadi input pada fungsi `ols` adalah formula berikut

$$respon \sim prediktor_1 + prediktor_2 + \dots + prediktor_q$$

Apabila ingin menjalankan regresi tanpa konstanta maka ditambahkan -1 di akhir formula.

3. **Uji Overall.** Setelah mendapatkan model regresi berganda kita lakukan uji ANOVA untuk melihat kelayakan model regresi. Model dikatakan layak untuk digunakan apabila terdapat sekurang-kurangnya satu prediktor yang signifikan mempengaruhi respon. Dengan kata lain, terdapat minimal satu koefisien β_j yang signifikan berbeda dari 0.

Hipotesis yang digunakan adalah:

- $H_0: \beta_1 = \beta_2 = \dots = \beta_q = 0$ (model regresi tidak layak digunakan, tidak ada hubungan linear antara prediktor dan respon)
- $H_a: \text{Terdapat } \beta_j \neq 0, j = 1, 2, \dots, q$ (model regresi layak digunakan, ada hubungan linear antara prediktor dan respon).

Apabila uji ini tidak dipenuhi, Kita coba untuk menggunakan variabel prediktor lain atau melakukan transformasi variabel.

4. Uji Parsial Konstanta dan Koefisien

Uji parsial digunakan untuk mengetahui apakah konstanta dan koefisien masing-masing memberikan kontribusi signifikan terhadap model.

Uji ini dilakukan dengan hipotesis sebagai berikut.

- $H_0: \beta_i = 0$
- $H_a: \beta_i \neq 0$ (Konstanta ke- i tidak signifikan terhadap model regresi)

Apabila koefisien tidak signifikan, terdapat opsi untuk mengeluarkannya dari model regresi secara bertahap (satu per satu) dari koefisien paling tidak signifikan (nilai p – *value* terbesar), kemudian menjalankan tahap 2-4 kembali. Metode ini dinamakan backward-elimination. Apabila konstanta tidak signifikan, disarankan untuk tetap digunakan.

Untuk mengilustrasikan penggunaan Python untuk regresi berganda akan digunakan data TaiwanEstate. Setelah melakukan uji asumsi Normalitas dan Linearitas kita akan memodelkan variabel respon “Harga” berdasarkan beberapa prediktor yaitu “UsiaBangunan”, “Kebisingan”, “JarakStasiunMRT”, “NumMinimarket”, dan “Humiditas”.

```
import statsmodels.formula.api as smf
import pandas as pd
import numpy as np
from scipy.stats import *

df = pd.read_csv("Downloads/pelatihan/TaiwanEstate.csv",
                 sep = ";")

#Membangun model OLS
lr = smf.ols("Harga ~ UsiaBangunan + \
            Kebisingan + \
            JarakStasiunMRT + \
            NumMinimarket + \
            Humiditas", data = df).fit()

#Memunculkan hasil dari fitting model
```

```
lr.summary()
```

OLS Regression Results			
Dep. Variable:	Harga	R-squared:	0.570
Model:	OLS	Adj. R-squared:	0.565
Method:	Least Squares	F-statistic:	108.1
Date:	Tue, 16 Apr 2024	Prob (F-statistic):	1.78e-72
Time:	09:39:42	Log-Likelihood:	-1966.8
No. Observations:	414	AIC:	3946.
Df Residuals:	408	BIC:	3970.
Df Model:	5		
Covariance Type:	nonrobust		

Uji Overall

Hipotesis yang digunakan adalah

- $H_0 : \beta_{UsiaBangunan} = \beta_{Kebisingan} = \dots = \beta_{Humiditas} = 0$
- $H_a : \text{Terdapat } \beta_j \neq 0, j \in \{UsiaBangunan, Kebisingan, \dots, Humiditas\}$

Hasil uji overall dapat ditemukan pada kolom kedua tabel 1, yakni F-statistic = 108.1 dengan DF = 5 dan 408. Untuk kemudahan perhatian $\text{Prob}(F - \text{statistic}) = 1.78 \times 10^{-72} < 0.05$ sehingga H_0 ditolak. Artinya model regresi ini layak untuk digunakan pada taraf signifikansi $\alpha = 5\%$.

Uji Parsial

Digunakan hipotesis

- $H_0 : \beta_i = 0$
- $H_a : \beta_i \neq 0$ (Konstanta ke- i tidak signifikan terhadap model regresi)

Hasil uji parsial dapat ditemukan pada bagian tabel kedua. Diperoleh

	coef	std err	t	P> t	[0.025	0.975]
Intercept	376.5177	12.751	29.529	0.000	351.452	401.583
UsiaBangunan	-0.8730	0.122	-7.137	0.000	-1.114	-0.633
Kebisingan	-0.6913	0.428	-1.615	0.107	-1.533	0.150
JarakStasiunMRT	-0.0169	0.001	-12.209	0.000	-0.020	-0.014
NumMinimarket	4.2267	0.594	7.118	0.000	3.059	5.394
Humiditas	0.0706	0.134	0.527	0.598	-0.192	0.334

Parameter	Estimasi	P-value	Kesimpulan
Konstanta (β_0)	376.5177	0	H_0 ditolak. Konstanta signifikan
UsiaBangunan	-0.873	0	H_0 ditolak. Konstanta signifikan
Kebisingan	-0.6913	0.107	H_0 tidak ditolak. Kebisingan tidak signifikan
JarakStasiunMRT	-0.0169	0	H_0 ditolak. Konstanta signifikan
BanyaknyaMinimarket	4.2267	0	H_0 ditolak. Konstanta signifikan
Humiditas	0,0706	0.598	H_0 tidak ditolak. Humiditas tidak signifikan

Terdapat dua variabel prediktor yang tidak signifikan masuk model.

Persamaan regresi

Hasil diatas dapat dirangkum dalam sebuah persamaan regresi sebagai berikut

$$\text{Harga} = 376.52 - (0.873 * \text{Usia}) - (0.691 * \text{Kebisingan}) - (0.017 * \text{JarakMRT}) + (4.227 * \text{Minimarket}) + (0.071 * \text{Humiditas})$$

Penghapusan prediktor

Apabila ditemui satu atau beberapa variabel prediktor yang tidak signifikan pada Uji Parsial, kita dapat membentuk model regresi baru dengan mengeluarkan satu prediktor yang “paling tidak signifikan”. Artinya, prediktor dengan statistik uji F terkecil (atau P-value terbesar) dikeluarkan. Sebagai contoh pada model diatas variabel “Humiditas” mempunyai $p - value$ terbesar sehingga dihapus dari model.

```
#Membangun model OLS
lr2 = smf.ols("Harga ~ UsiaBangunan + \
              Kebisingan + \
              JarakStasiunMRT + \
              NumMinimarket", data = df).fit()

#Memunculkan hasil dari fitting model
lr2.summary()
```

	coef	std err	t	P> t	[0.025	0.975]
Intercept	379.0576	11.796	32.134	0.000	355.869	402.246
UsiaBangunan	-0.8722	0.122	-7.137	0.000	-1.112	-0.632
Kebisingan	-0.6796	0.427	-1.591	0.112	-1.519	0.160
JarakStasiunMRT	-0.0169	0.001	-12.220	0.000	-0.020	-0.014
NumMinimarket	4.2466	0.592	7.173	0.000	3.083	5.410

diperoleh variabel “Kebisingan” mempunyai $p - value > 5\%$ sehingga tidak signifikan. Kita lakukan pembuangan terhadap variabel “Kebisingan”.

```
#Membangun model OLS
lr3 = smf.ols("Harga ~ UsiaBangunan + \
              JarakStasiunMRT + \
              NumMinimarket", data = df).fit()

#Memunculkan hasil dari fitting model
lr3.summary()
```

	coef	std err	t	P> t	[0.025	0.975]
Intercept	361.5300	4.223	85.604	0.000	353.228	369.832
UsiaBangunan	-0.8644	0.122	-7.066	0.000	-1.105	-0.624
JarakStasiunMRT	-0.0169	0.001	-12.233	0.000	-0.020	-0.014
NumMinimarket	4.2858	0.593	7.232	0.000	3.121	5.451

diperoleh semua variabel sudah signifikan. Sehingga diperoleh model regresi ketiga

$$\text{Harga} = 361.53 - (0.864 * \text{Usia}) - (0.017 * \text{JarakMRT}) + (4.286 * \text{Minimarket})$$

Pemilihan Model

Setelah menjalankan regresi bertahap hingga seluruh prediktor signifikan, tahap selanjutnya adalah menentukan model regresi yang terbaik untuk Perlu diperhatikan bahwa model regresi dengan prediktor paling sedikit belum tentu terbaik. Hal ini dikarenakan beberapa prediktor yang tak signifikan mungkin memainkan peran penting dan sebaiknya tidak dihapus (misal: nilai Adjusted R2 lebih tinggi). Dari ketiga model diatas diperoleh ringkasan berikut

Model	R-squared	Adj R-squared	AIC	BIC	Log-likelihood
lr1	0.57	0.565	3946	3970	-1966.8
lr2	0.569	0.565	3944	3964	-1966.9
lr3	0.567	0.564	3944	3960	-1968.2

Diperoleh model dengan AIC terendah adalah lr2 dan lr3, namun dilihat dari Adjusted R-squared diperoleh model terbaik permasalahan diatas adalah model kedua yaitu

$$\text{Harga} = 379.058 - (0.872 * \text{Usia}) - (0.68 * \text{Kebisingan}) - (0.017 * \text{JarakMRT}) + (4.247 * \text{Minimarket})$$

DAFTAR PUSTAKA

- Johnson, R., & Bhattacharyya, G. (1987). *Statistics Principles and Methods*. New York: John Wiley & Sons.
- Massaron, L., & Boschetti, A. (2016). *Regression Analysis with Python*. Birmingham: PACKT.
- Rogel-Salazar, J. (2023). *Statistics and Data Visualisation with Python*. London: CRC Press.
- Rosadi, D. (2016). *Analisis Statistika dengan R*. Yogyakarta: Gadjah Mada University Press.
- Walpole, R. E., Myers, R. H., Myers, S. L., & Ye, K. (2016). *Probability & Statistics for Engineers & Scientist*. Boston: Pearson.