Integration R Quarto & Python Language for Dynamic Reproting

Victor Nugraha

Pendahuluan

Materi ini diproduksi oleh tim dari Algoritma untuk *DSS - Integration R Quarto & Python Language for Dynamic Reproting*. Materi berikut hanya ditujukan untuk kalangan terbatas, meliputi individu/personal yang menerima materi ini secara langsung dari lembaga pelatihan. Materi ini dilarang untuk direproduksi, didistribusikan, diterjemahkan, atau diadaptasikan dalam bentuk apapun di luar izin dari individu dan organisasi yang berkepentingan.

Algoritma adalah pusat pendidikan Data Science di Jakarta. Kami mengadakan workshop dan program pelatihan untuk membantu para profesional dan pelajar untuk mendapatkan keahlian dalam berbagai bidang dalam ruang lingkup Data Science: *data visualization, machine learning, data modeling, statistical inference*, dan lain-lainnya.

Sebelum masuk ke dalam materi dan menjalankan kode-kode di dalam materi ini, silakan anda melihat bagian **Library and Setup** untuk melihat dan memastikan semua persyaratan dasar untuk mengikuti materi ini sudah terpenuhi termasuk package-package yang diperlukan. Pada bagian **Tujuan Pembelajaran** anda dapat melihat secara umum apa saja yang akan dipelajari dalam modul materi ini. Kami harap materi ini akan bermanfaat bagi karir ataupun menambah keahlian peserta.

Inline Code Dynamic Reporting

Inline Code Dynamic Reporting adalah metode pembuatan pelaporan di mana kode pemrograman ditulis langsung di dalam laporan untuk memproses, menganalisis, dan menampilkan data secara dinamis. Dengan pendekatan ini, pengguna dapat memasukkan atau mengubah kode di dalam laporan untuk menyesuaikan *output* sesuai kebutuhan spesifik, seperti menghitung metrik baru, memfilter data berdasarkan kondisi tertentu, atau membuat visualisasi khusus. *Inline code* ini biasanya ditulis dalam bahasa pemrograman yang mendukung analisis data, seperti Python atau R, dan dieksekusi secara langsung saat laporan diakses.

Beberapa manfaat yang didapatkan dari metode *inline code dynamic reporting* adalah

- **Fleksibilitas**: Memungkinkan pengguna untuk membuat laporan yang sepenuhnya dapat disesuaikan dan sesuai dengan kebutuhan analisis yang berubah-ubah tanpa memerlukan laporan baru atau alat tambahan.
- **Efisiensi**: Mengurangi kebutuhan untuk membuat laporan secara manual atau memodifikasi laporan yang sudah ada karena semua perubahan dapat dilakukan secara langsung di dalam laporan dengan menyesuaikan kode.
- **Interaktivitas**: Pengguna dapat berinteraksi dengan data secara lebih mendalam dengan menambahkan atau mengubah kode untuk melihat bagaimana berbagai skenario atau parameter memengaruhi hasil laporan.

Integrasi antara R & Python

Seperti yang disampaikan di atas, Python **atau** R dapat dimanfaatkan untuk membuat sebuah *inline code dynamic reporting*, tapi kenapa kita tidak memanfaatkan Python **dan** R secara bersamaan?

Dengan banyaknya kebutuhan untuk melakukan otomasi untuk efisiensi pekerjaan, membuat para pengembang teknologi harus saling membahu agar dapat menyajikan alat bantu yang lebih baik dari waktu ke waktu. Alasan tersebutlah yang menjadi bahan bakar untuk developer Python & Posit, mengintegrasikan beberapa produk mereka untuk melengkapi antara satu dengan yang lainnya.

Bahasa Python digemari dan digunakan oleh banyak individu dalam industri pekerjaan untuk mengolah data karena kemampuannya yang kuat dalam menangani berbagai jenis data, didukung oleh *library* yang memudahkan analisis, manipulasi, dan visualisasi data. Selain itu, sintaksis Python yang sederhana dan intuitif memungkinkan pengguna untuk menulis kode yang efisien dan mudah dipahami, sehingga mempercepat proses pengolahan data dan pengambilan keputusan berbasis data.

Sedangkan R, yang sekarang dikembangkan oleh **posit**, menjadi salah opsi yang bisa juga dimanfaatkan untuk melakukan tugas yang sama dengan Python, akan tetapi R memiliki sebuah format file yang memungkinkan penggunanya untuk menghubungkan hasil kode pemograman ke narasi dan format file yang dimaksud adalah file **R Quarto Markdown**. Walaupun format file tersebut dibesarkan oleh perusahaan yang mengembangkan R, format file tersebut bisa menerima dan mengolah bahasa pemograman Python.

> "At the end of the day, we want to get the work done - not worry about tools."

Tujuan Pembelajaran

Tujuan utama dari *workshop* ini adalah untuk memberikan pengenalan yang komprehensif mengenai tools dan perangkat lunak yang digunakan untuk melakukan *Inline Code Dynamic Reporting*, yakni dua open-source ternama: R & Python. Adapun materi ini akan mencakup:

- Dasar R Studio & Bahasa Python
  - Perkenalan ke software RStudio
  - Integrasi antara Python & RStudio
  - Perkenalan ke bahasa Python
  - Pemprosesan data dan manipulasi data dengan pandas
  - Pembuatan visualisasi sederhana dengan matplotlib
- Format File Quarto Markdown
  - Perbedaan Quarto dengan R Markdown
  - Perkenalan ke YAML
  - Quarto Markdown Python chunk
  - *Inline code*
  - *Render* Quarto Markdown
- Dynamic Reporting
  - Pemanfaatan *inline code*
  - Mengatur tampilan laporan
  - *Exporting dynamic reporting*

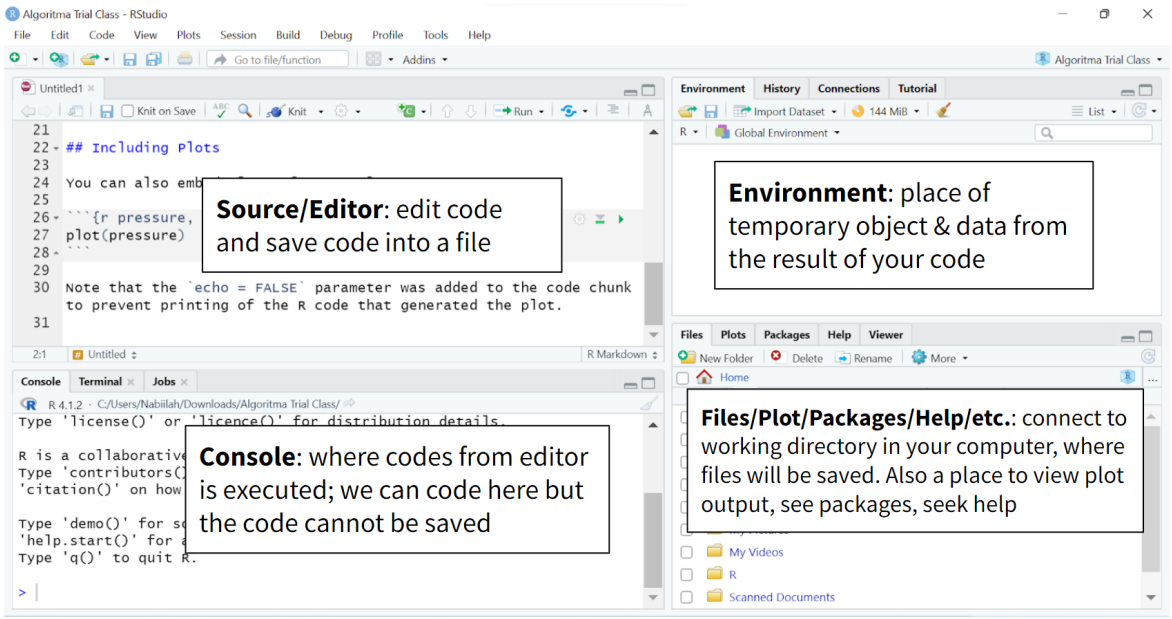
Dasar R Studio & Bahasa Python

Sebagai tahapan awal untuk memudahkan proses pembelajaran, mari kita coba berkenalan atau mengulang ingatan tentang R Studio & dasar bahasa Python.

Perkenalan ke *Software* RStudio

Komponen UI RStudio

```
# Pada R Studio kita juga bisa memasukan gambar dengan menggunakan code di bawah ini
from IPython.display import Image, display
display(Image(filename="assets/RStudio_UI.PNG"))
```



Terdapat 4 panel utama yang harus Anda pahami yaitu :

1. Panel Source : Panel ini merupakan fitur utama dari RStudio, panel ini menampilkan file yang sedang dibuka pada RStudio.

2. Panel Console : Panel ini menampilkan console asli dari R yang digunakan untuk berkomunikasi dengan R session. Terdapat beberapa tab lain seperti Terminal yang dapat digunakan untuk mengakses komputer Anda melalui Command Line Interface (CLI).

3. Panel Environment / History : Bagian ini menampilkan seluruh object R yang sudah dibuat selama session yang sama. Terdapat tab History yang berfungsi untuk melihat history dari kode yang sudah dijalankan sebelumnya.

4. Panel Files/Plot/Packages/Help :

- *Tab Files* : Daftar dari berkas (file) yang berada dalam *working directory*
- *Tab Plot* : Menampilkan visualisasi yang terbentuk
- *Tab Packages* : Berisi daftar packages yang sudah ter-*install*
- *Tab Help* : Menampilkan dokumentasi resmi dari setiap fungsi

Komponen *Tab Files* RStudio

Pada komponen *tab files* akan digunakan untuk membuka file ataupun script dari R. Dalam kesempatan kali ini, format file yang digunakan adalah Quarto Markdown. Penjelasan yang lebih komprehensif akan dijelaskan pada bagian bawah material ini, akan tetapi yang akan dijelaskan terlebih dahulu pada bagian ini adalah Quarto Markdown memiliki 2 bagian, yaitu

- **Bagian Warna Putih**: Berguna untuk menulis narasi. Pada bagian ini dapat ditambahkan heading dengan menambahkan hashtag `#` sebelum kalimat. Narasi yang tidak ditambahkan `#` sebelum kalimat akan menjadi paragraf biasa.
- **Bagian Warna Abu-Abu**: Sering disebut sebagai `chunk`. Chunk berguna untuk menulis **code**.

```
# Ini adalah contoh chunk
```

Disclaimer, pembuatan chunk dapat menggunakan *shortcut* `ctrl + alt + i`

Integrasi antara Python & RStudio

Untuk melakukan integrasi antara Python & RStduio terdapat beberapa hal yang perlu dipersiapkan.

1. **Mempersiapkan Python dengan Miniconda atau Anaconda**

- **Unduh dan Instal Miniconda/Anaconda**: Unduh Miniconda/Anaconda dari situs resmi Miniconda dan ikuti instruksi instalasi untuk sistem operasi Anda.
- **Buat Lingkungan Conda**: Setelah Miniconda terinstal, buat lingkungan Conda baru yang akan digunakan dengan RStudio.

```
conda create -n myenv python=3.9
conda activate myenv
```

Discalimer, gantilah `myenv` dengan nama *environtment* yang diinginkan.

- **Instal *Library* Python**: Instal *library* Python yang Anda perlukan dalam lingkungan Conda tersebut, seperti `pandas`, `numpy`, dan `matplotlib`.

```
conda install pandas numpy matplotlib
```

2. **Instal Paket reticulate di RStudio**

- **Buka RStudio**
- **Instal Paket reticulate**: Instal paket reticulate jika belum terinstal.

```
install.packages("reticulate")
```

Perkenalan ke bahasa Python

Python menawarkan alat yang efisien dan fleksibel untuk analisis data. Kemampuannya untuk mengintegrasikan berbagai sumber data, mendukung operasi data yang kompleks, serta memudahkan pembuatan laporan dan visualisasi menjadikannya pilihan utama di bidang data science dan analisis data.

Selanjutnya kita akan mengenal tentang dasar - dasar dalam pemrograman bahasa Python.

Variables

Di Python, variabel adalah tempat untuk menyimpan data yang dapat digunakan dan dimanipulasi dalam program. Variabel memungkinkan Anda untuk menyimpan nilai, seperti angka, teks, atau objek lainnya, dan memberi nama pada nilai-nilai tersebut untuk referensi selanjutnya.

```
# Menyimpan nilai ke variabel
materi = "Python Reporting"
```

```
# Memanggil isi variabel
materi
```

```
'Python Reporting'
```

Disclaimer, menjalankan chunk dapat menggunakan *shortcut* `ctrl + shift + enter`

Case Sensitive

Dalam Python, *case sensitive* berarti bahwa huruf besar dan huruf kecil dianggap berbeda. Ini berlaku pada penamaan variabel, fungsi, dan pengenal lainnya. Sebagai contoh, variabel `materi`, `Materi`, dan `MATERI` dianggap sebagai tiga variabel yang berbeda.

```
# Menyimpan nilai ke variabel
materi = "Python Reporting"
Materi = "Python"
MATERI = "Reporting"

# Memanggil isi variabel
materi # Output Python Reporting
Materi # Output Python
MATERI # Output Reporting
```

```
'Reporting'
```

Karena Python peka terhadap huruf besar dan kecil, penting untuk konsisten dalam penggunaan huruf saat mengakses variabel atau fungsi yang telah didefinisikan.

Overwrite

Dalam Python, *overwrite* terjadi ketika sebuah variabel yang sudah ada diberi nilai baru, sehingga nilai lama variabel tersebut akan digantikan oleh nilai baru.

Misalnya, jika Anda mendeklarasikan variabel `materi` dengan nilai tertentu, dan kemudian memberikan nilai baru pada variabel `materi`, maka nilai sebelumnya akan hilang, dan variabel `materi` hanya akan menyimpan nilai baru tersebut.

```
# Menyimpan nilai ke variabel
materi = "Python Reporting"
materi = "Python"

# Memanggil isi variabel
materi
```

```
'Python'
```

Pada contoh di atas, nilai awal variabel `materi` adalah `Python Reporting`, tetapi setelah di-overwrite dengan `Python`, nilai sebelumnya tidak lagi dapat diakses. Ini adalah perilaku default Python yang memungkinkan variabel untuk berubah nilai seiring waktu.

Do & Don't

Berikut ini adalah beberapa hal yang **boleh** dan **tidak boleh** dilakukan terkait dengan penamaan variabel di Python:

Do

1. Penamaan Variabel Tidak Harus 1 Kata

Dalam penamaan variabel, ada kalanya harus menggunakan lebih dari satu kata. Untuk memungkinkan hal tersebut, diantara katanya bisa diberikan simbol *underscore*.

```
# Menyimpan nilai ke variabel dengan 2 kata
dss_sep = "Python Reporting"

# Memanggil isi variabel
dss_sep
```

```
'Python Reporting'
```

2. Angka Diletakan Di Belakang

```
# Menyimpan nilai ke variabel dengan angka
dss_sep_2024 = "Dynamic Reporting"

# Memanggil isi variabel
dss_sep_2024
```

```
'Dynamic Reporting'
```

Don't

1. Special Character `!, $ , &, dll` Tidak Dapat Digunakan

2. Tidak Boleh Menggunakan Keyword Python

Keywords adalah kata kunci yang sudah ditetapkan oleh Python sebagai nama yang tidak bisa dipakai baik untuk penamaan fungsi, variabel, dan lainnya. Keyword ditulis dalam lower-case (huruf kecil semua) kecuali keyword `True`, `False`, dan `None`. Sejauh ini keyword yang ada pada Python adalah sebagai berikut:

```
# Cek daftar keyword
import keyword
keyword.kwlist
```

```
['False',
 'None',
 'True',
 'and',
 'as',
 'assert',
 'async',
 'await',
```

```
'break',
'class',
'continue',
'def',
'del',
'elif',
'else',
'except',
'finally',
'for',
'from',
'global',
'if',
'import',
'in',
'is',
'lambda',
'nonlocal',
'not',
'or',
'pass',
'raise',
'return',
'try',
'while',
'with',
'yield']
```

Tipe Data Python

Pada Python terdapat beberapa tipe data berdasarkan peruntuhannya. Berikut beberapa jenis tipe data pada python:

1. ***String***

Dalam bahasa Python, tipe data **String** merepresentasikan teks. Tipe data ini terdiri dari serangkaian karakter yang juga dapat mengandung spasi dan angka. Sebagai contoh, kata "hamburger" dan frasa "I ate 3 hamburgers" keduanya merupakan string. Ada beberapa cara untuk membuat objek string di Python:

- Menggunakan tanda kutip tunggal -> 'Hello World!'
- Menggunakan tanda kutip ganda -> "I'm a programmer"
- Menggunakan tanda kutip tiga -> ''' "I'm smart!", he said '''

Disclaimer, tanda kutip tiga dalam Python sangat membantu dengan memungkinkan string merentang ke beberapa baris.

```
judul_dss = "Integration R Quarto & Python Language for Dynamic Reproting"
judul_dss
```

```
'Integration R Quarto & Python Language for Dynamic Reproting'
```

Kita dapat melakukan pengecekan tipe data dengan menggunakan fungsi `type()`, seperti yang akan dituliskan di bawah ini.

```
# Mengecek bentukan data
type(judul_dss)
```

```
str
```

2. ***Number***

Python memiliki 2 bentukan untuk data yang berupa angka, yaitu *integer* & *float*.

- ***Integer* (int64):**
  - Tipe data yang digunakan untuk menyimpan bilangan bulat, baik positif, negatif, maupun nol.
  - Tidak memiliki bagian desimal.
  - Contoh: -5, 0, 42

```
integer = 10
integer
```

```
10
```

```
type(integer)
```

```
int
```

- ***Float* (float64):**
  - Tipe data yang digunakan untuk menyimpan bilangan pecahan atau desimal.
  - Dapat menyimpan angka dengan bagian desimal.
  - Contoh: 3.14, -0.001, 2.0

```
desimal = 10.12
desimal
```

```
10.12
```

```
type(desimal)
```

```
float
```

3. ***Compound***

Di Python, *compound types* adalah tipe data yang dapat menyimpan lebih dari satu nilai di dalamnya. Ada satu jenis compound types yang sering digunakan, yaitu `list`.

`List` merupakan salah satu tipe data paling dasar dan serbaguna yang digunakan untuk menyimpan koleksi elemen. List sangat berguna karena sifatnya yang fleksibel, memungkinkan penyimpanan berbagai jenis data dan manipulasi elemen dengan mudah.

List dibuat dengan menempatkan elemen-elemen di dalam tanda kurung siku `[]`, dipisahkan oleh koma.

```
# Membuat list
bahasa = ["Python", "R", "C"]
tahun = [2021, 2022, 2023, 2024]

# Menampilkan list
bahasa
tahun
```

```
[2021, 2022, 2023, 2024]
```

```
type(bahasa)
```

```
list
```

Struktur Kontrol `if-else`

Pernyataan `if-else` di Python adalah salah satu struktur kontrol yang paling dasar dan penting dalam pemrograman. Ini digunakan untuk membuat keputusan di dalam kode berdasarkan kondisi tertentu. Intuisinya mirip dengan pengambilan keputusan dalam kehidupan sehari-hari: `"Jika suatu kondisi benar, lakukan sesuatu; jika tidak, lakukan hal lain."`

Berikut adalah struktur dasar dari pernyataan if-else:

```
if kondisi:
    # Blok kode ini akan dijalankan jika kondisi bernilai True
else:
    # Blok kode ini akan dijalankan jika kondisi bernilai False
```

- **`Kondisi if:`**:

Python mengevaluasi kondisi ini terlebih dahulu. Jika hasilnya `True`, blok kode yang ada di bawah `if` (yang memiliki indentasi) akan dijalankan.

- **`Kondisi else:`**:

Jika kondisi pada `if` bernilai `False`, maka Python akan menjalankan blok kode yang ada di bawah `else`. Blok `else` bersifat opsional; artinya, Anda bisa hanya menggunakan `if` tanpa `else` jika tidak diperlukan.

Selain dari kedua kondisi di atas, terdapat satu kondisi lagi yaitu `elif` (Else If), kondisi tersebut memungkinkan pengguna untuk menggunakan lebih dari 1 kondisi.

```python
# Nilai yang mau dicek
score = 85

# Struktur kondisi
if score >= 90: # Kondisi 1
    print("Grade: A") # Hasil kondisi 1
elif score >= 80: # Kondisi 2
    print("Grade: B") # Hasil kondisi 2
elif score >= 70: # Kondisi 3
    print("Grade: C") # Hasil kondisi 3
else: # Kondisi 4
    print("Grade: F") # Hasil kondisi 4
```

Grade: B

Penjelasan:

- Kondisi 1: Jika ***score* lebih besar atau sama dengan 90**, program mencetak **Grade: A**.
- Kondisi 2: Jika ***score* lebih besar atau sama dengan 80 tapi kurang dari 90**, program mencetak **Grade: B**.
- Kondisi 3: Jika ***score* lebih besar atau sama dengan 70 tapi kurang dari 80**, program mencetak **Grade: C**.
- Kondisi 4: Jika semua kondisi di atas *False*, program mencetak **Grade: F**.

Pemprosesan Data & Manipulasi Data Dengan Pandas

Persiapan *Library*

Library Python adalah kumpulan modul (file berisi kode Python) yang menyediakan fungsi, kelas, dan metode yang dapat digunakan untuk menyelesaikan tugas-tugas tertentu tanpa harus menulis kode dari nol. *Library* membantu pengembang untuk menghemat waktu dan usaha dengan menyediakan solusi siap pakai untuk berbagai kebutuhan, seperti manipulasi data, jaringan, pemrosesan gambar, analisis statistik, dan banyak lagi.

Untuk menjalankan atau menggunakan *library* di Python, Anda perlu mengikuti beberapa langkah dasar, mulai dari instalasi hingga impor *library* tersebut ke dalam kode Python Anda. Berikut adalah langkah-langkahnya:

1. **Menginstal Library**

Jika *library* yang ingin Anda gunakan adalah *library* eksternal (tidak termasuk dalam Python standard *library*), penggunanya perlu menginstalnya terlebih dahulu menggunakan `pip`, yang merupakan *package manager* untuk Python.

Contoh Instalasi dengan pip:

```
pip install nama_library
```

2. **Mengimpor Library**

Setelah *library* diinstal, Anda perlu mengimpor library tersebut ke dalam skrip Python Anda sebelum bisa menggunakannya.

Contoh Pengimporan:

```
import nama_library
```

Pandas *Library*

Pandas digunakan untuk manipulasi dan analisis data, memungkinkan operasi yang mudah pada data berstruktur seperti tabel.

```
import pandas as pd
print(pd.__version__)
```

2.2.2

Disclaimer, `as pd` adalah alias yang digunakan untuk memudahkan penggunaan nama *library* yang panjang.

Membaca *DataFrame*

Membaca data ke dalam *DataFrame* menggunakan Pandas adalah salah satu langkah awal yang umum dilakukan dalam analisis data. Pandas menyediakan berbagai fungsi untuk membaca data dari berbagai sumber, seperti file CSV, Excel dan lainnya, ke dalam *DataFrame*.

Dalam kesempatan ini, format data yang akan digunakan adalah CSV, dan fungsi yang dapat digunakan adalah `pd.read_csv()`.

```
sales = pd.read_csv('data/sales.csv', index_col =0)
sales["Date"] = pd.to_datetime(sales["Date"], dayfirst = True, errors = 'coerce')
sales["Date"] = sales["Date"].sort_values()
sales = sales[(sales['Date'] >= "2021-01-01") & (sales['Date'] <= "2022-12-31")]
sales.to_csv("data/sales_2021_2022.csv", header =True, index = False)
```

```
sales
```

	Date	Q-P1	Q-P2	Q-P3	Q-P4	S-P1	S-P2	S-P3	S-P4
3838	2021-01-01	7693	2394	2384	1367	24386.81	15177.96	12921.28	9746.71
3839	2021-01-02	5418	1294	1071	1492	17175.06	8203.96	5804.82	10637.96
3840	2021-01-03	6718	649	4878	1380	21296.06	4114.66	26438.76	9839.40
3841	2021-01-04	6964	1122	2730	428	22075.88	7113.48	14796.60	3051.64
3842	2021-01-05	3963	1848	1563	944	12562.71	11716.32	8471.46	6730.72
...	...	...	...	...	...	...	...	...	...
4561	2022-12-26	7600	662	4510	988	24092.00	4197.08	24444.20	7044.44
4562	2022-12-27	7114	2948	681	700	22551.38	18690.32	3691.02	4991.00
4563	2022-12-28	7759	356	1834	1142	24596.03	2257.04	9940.28	8142.46
4564	2022-12-29	6457	1851	3369	669	20468.69	11735.34	18259.98	4769.97

	Date	Q-P1	Q-P2	Q-P3	Q-P4	S-P1	S-P2	S-P3	S-P4
4565	2022-12-30	7284	1417	788	1369	23090.28	8983.78	4270.96	9760.97

```
# Membaca data dan menyimpannya ke variabel
sales = pd.read_csv('data/sales_2021_2022.csv')
```

Untuk mempermudah proses penampilan data, pandas menyediakan dua fungsi yaitu `head()`& `tail()`. Fungsi tersebut akan menampilkan bagian awal atau akhir dari data yang dimiliki

```
# Menampilkan 5 baris pertama
sales.head()
```

	Date	Q-P1	Q-P2	Q-P3	Q-P4	S-P1	S-P2	S-P3	S-P4
0	2021-01-01	7693	2394	2384	1367	24386.81	15177.96	12921.28	9746.71
1	2021-01-02	5418	1294	1071	1492	17175.06	8203.96	5804.82	10637.96
2	2021-01-03	6718	649	4878	1380	21296.06	4114.66	26438.76	9839.40
3	2021-01-04	6964	1122	2730	428	22075.88	7113.48	14796.60	3051.64
4	2021-01-05	3963	1848	1563	944	12562.71	11716.32	8471.46	6730.72

Data yang ditampilkan merupakan data penjualan pada sebuah perusahaan dari awal tahun 2021 sampai dengan akhir tahun 2022. Ditambah juga dengan keterangan jumlah barang yang terjual beserta nominal sales yang didapatkan, untuk 4 jenis barang.

Deskripsi data:

- `Date`: Tanggal transaksi
- `Q-P1`: Jumlah kuantitas produk 1 yang terjual
- `Q-P2`: Jumlah kuantitas produk 2 yang terjual
- `Q-P3`: Jumlah kuantitas produk 3 yang terjual
- `Q-P4`: Jumlah kuantitas produk 4 yang terjual
- `S-P1`: Jumlah sales produk 1 yang terjual
- `S-P2`: Jumlah sales produk 2 yang terjual
- `S-P3`: Jumlah sales produk 3 yang terjual
- `S-P4`: Jumlah sales produk 4 yang terjual

Persiapan Data

Mengubah Nama Kolom

Dari nama kolom di awal, setiap nama kolomnya dirasa kurang intuitif karena merupakan singkatan dan nantinya ketika ingin ditampilkan dalam bentuk laporan akan kurang rapi. Maka dari itu, akan lebih baik untuk diubah dengan nama yang lebih intuitif.

Untuk melakukan perubahan, terdapat sebuah metode yang dapat dimanfaatkan, yaitu dengan menggunakan atribut `.columns` dan membuat sebuah `list` nama kololom baru secara berurut untuk mengganti nama kolom yang lama.

```
# Mengganti nama kolom
sales.columns = ['Date', 'Quantity Product 1', 'Quantity Product 2', 'Quantity Product 3', '
```

```
# Mengecek hasil perubahan nama kolom
sales
```

	Date	Quantity Product 1	Quantity Product 2	Quantity Product 3	Quantity Product 4	Sales Produ
0	2021-01-01	7693	2394	2384	1367	243
1	2021-01-02	5418	1294	1071	1492	171
2	2021-01-03	6718	649	4878	1380	212
3	2021-01-04	6964	1122	2730	428	220
4	2021-01-05	3963	1848	1563	944	125
...	...	...	...	...	...	...
719	2022-12-26	7600	662	4510	988	240
720	2022-12-27	7114	2948	681	700	225
721	2022-12-28	7759	356	1834	1142	245
722	2022-12-29	6457	1851	3369	669	204
723	2022-12-30	7284	1417	788	1369	230

Tipe Data

Tipe data adalah klasifikasi yang menentukan jenis nilai yang dapat disimpan dan diproses dalam suatu variabel. Pada DataFrame di Python, yang biasanya dikelola dengan `library pandas`, setiap kolom dalam DataFrame dapat memiliki tipe data yang berbeda. Berikut adalah beberapa tipe data umum yang ditemukan dalam DataFrame:

- ***Integer (int64)***: Tipe data untuk bilangan bulat. Biasanya digunakan ketika kolom hanya berisi angka bulat tanpa desimal.
- ***Float (float64)***: Tipe data untuk bilangan desimal. Digunakan ketika kolom berisi angka dengan nilai desimal.
- ***Boolean (bool)***: Tipe data untuk nilai logika, yaitu True atau False.
- ***Datetime (datetime64)***: Tipe data untuk menyimpan tanggal dan waktu
- ***String (object)***: Tipe data untuk teks atau urutan karakter. Dalam DataFrame pandas, tipe data string biasanya diwakili sebagai object.

- ***Categorical (category)***: Tipe data untuk kolom yang hanya memiliki sejumlah nilai unik terbatas. Biasanya digunakan untuk menyimpan data kategorikal seperti label atau kelas.

Mengecek Tipe Data

Metode `.info()` adalah salah satu metode yang sangat berguna di pandas untuk mendapatkan ringkasan informasi tentang tipe data masing-masing kolom pada DataFrame. Selain dari itu, `.info()` juga memberikan gambaran umum mengenai struktur DataFrame, termasuk jumlah baris dan kolom, nama kolom, dan jumlah nilai non-null (tidak kosong) di setiap kolom.

```
# Mengecek tipe data
sales.info()
```

<class ‘pandas.core.frame.DataFrame’> RangeIndex: 724 entries, 0 to 723 Data columns (total 9 columns): # Column Non-Null Count Dtype
— —— ——— ——
0 Date 724 non-null object 1 Quantity Product 1 724 non-null int64
2 Quantity Product 2 724 non-null int64
3 Quantity Product 3 724 non-null int64
4 Quantity Product 4 724 non-null int64
5 Sales Product 1 724 non-null float64 6 Sales Product 2 724 non-null float64 7 Sales Product 3 724 non-null float64 8 Sales Product 4 724 non-null float64 dtypes: float64(4), int64(4), object(1) memory usage: 51.0+ KB

Tipe Data DateTime

Dalam melakukan transformasi tipe data objek menjadi DateTime, dapat memanfaatkan fungsi `astype()` dan mengisinya dengan `datetime64[ns]`.

Akan tetapi pandas juga menyediakan sebuah fungsi lainnya yang lebih flexibel membantu dalam berhadapan dengan data DateTime, yaitu fungsi `pd.to_datetime(x)`.

Untuk mengetahui apa perbedaan dari kedua fungsi tersebut, mari coba implementasikan pada data dummy di bawah ini.

```
# Dummy data dengan format awal tanggal-bulan-tahun
date = pd.Series(['01-02-2023', '02-02-2023', '03-02-2023', '04-02-2023'])
date
```

```
0    01-02-2023
1    02-02-2023
2    03-02-2023
```

```
3    04-02-2023
dtype: object
```

```
# Implementasi astype()
date.astype('datetime64[ns]')
```

```
0   2023-01-02
1   2023-02-02
2   2023-03-02
3   2023-04-02
dtype: datetime64[ns]
```

Additional Notes: Hasil yang dikeluarkan akan berbentuk Tahun-Bulan-Tanggal

Dikarenakan keterbatasan dari `astype()` yang harus memiliki format awal yang sesuai dengan format hasil akhir, maka dari itu fungsi `pd.to_datetime()` bisa menjadi jawabannya karena fungsi tersebut dibekali dengan beberapa parameter pendukung, seperti yang dilampirkan di bawah ini.

- Parameter `dayfirst`

Dengan adanya parameter ini, dapat memberikan informasi format awal pada data tanggal yang diawali oleh informasi mengenai tanggal.

```
pd.to_datetime(sales["Date"], dayfirst = True, errors = 'coerce')
```

```
0     2021-01-01
1     2021-02-01
2     2021-03-01
3     2021-04-01
4     2021-05-01
         ...
719          NaT
720          NaT
721          NaT
722          NaT
723          NaT
Name: Date, Length: 724, dtype: datetime64[ns]
```

- Parameter `format`

Dengan adanya parameter ini, dapat memberikan informasi format awal seperti apa dan bentukan yang cukup variatif.

Dokumentasi Python `strftime` cheatsheet

```
pd.to_datetime(sales["Date"], format = "%Y-%m-%d", errors = 'coerce')
```

```
0      2021-01-01
1      2021-01-02
2      2021-01-03
3      2021-01-04
4      2021-01-05
          ...
719    2022-12-26
720    2022-12-27
721    2022-12-28
722    2022-12-29
723    2022-12-30
Name: Date, Length: 724, dtype: datetime64[ns]
```

```
sales['Date'] = pd.to_datetime(sales["Date"], format = "%Y-%m-%d", errors = 'coerce')
sales['Date'].info()
```

<class ‘pandas.core.series.Series’> RangeIndex: 724 entries, 0 to 723 Series name: Date Non-Null Count Dtype

---

724 non-null datetime64[ns] dtypes: datetime64ns memory usage: 5.8 KB

Feature Engineering

Feature Engineering merupakan sebuah proses untuk mengembangkan dan memilih suatu fitur atau atribut (*features*) yang akan digunakan untuk melakukan analisis data.

Feature Engineering ini adalah merupakan salah satu tahap paling penting untuk melakukan sebuah proyek analisis data, hal ini dikarenakan kualitas fitur yang dihasilkan dapat digunakan untuk menghasilkan manfaat yang besar pada hasil analisis data yang dihasilkan.

Disclaimer: Tahapan ini baru bisa dilakukan untuk kolom yang sudah memiliki tipe data *datetime*.

Berikut adalah beberapa atribut atau fungsi yang dapat dimanfaatkan untuk melakukan *feature engineering* pada tipe data `datetime` yang sudah dimiliki.

Setiap atribut atau fungsi yang digunakan akan menggunakan accessor `.dt`.

Date Component Numeric

- `.dt.year` -> Partisi tahun
- `.dt.month` -> Partisi bulan (angka)
- `.dt.day` -> Partisi day/tanggal (dalam angka)
- `.dt.dayofweek` -> Monday=0, Sunday=6

```
# Partisi tahun
sales["Date"].dt.year.head(3)
```

```
0    2021
1    2021
2    2021
Name: Date, dtype: int32
```

```
# Partisi bulan
sales["Date"].dt.month.head(3)
```

```
0    1
1    1
2    1
Name: Date, dtype: int32
```

```
# Partisi tanggal
sales["Date"].dt.day.head(3)
```

```
0    1
1    2
2    3
Name: Date, dtype: int32
```

```
# Partisi hari dalam angka
sales["Date"].dt.dayofweek.head(3)
```

```
0    4
1    5
2    6
Name: Date, dtype: int32
```

Date Component String

- `.dt.month_name()`-> Partisi bulan (nama)
- `.dt.day_name()`-> Partisi hari (nama)

```
# Partisi nama bulan
sales['Date'].dt.month_name().head(3)
```

```
0    January
1    January
2    January
Name: Date, dtype: object
```

```
# Partisi nama hari
sales['Date'].dt.day_name().head(3)
```

```
0      Friday
1    Saturday
2      Sunday
Name: Date, dtype: object
```

Date Transformation

- `.dt.to_period('W')` -> Transformasi ke format awal minggu - akhir minggu
- `.dt.to_period('M')` -> Transformasi ke format year-month
- `.dt.to_period('Q')` -> Transformasi ke format year-quartal

```
# Transformasi ke format awal minggu - akhir minggu
sales['Date'].dt.to_period('W').head(3)
```

```
0    2020-12-28/2021-01-03
1    2020-12-28/2021-01-03
2    2020-12-28/2021-01-03
Name: Date, dtype: period[W-SUN]
```

```
# Transformasi ke format tahun-bulan
sales['Date'].dt.to_period('M').head(3)
```

```
0    2021-01
1    2021-01
2    2021-01
Name: Date, dtype: period[M]
```

```
# Transformasi ke format tahun-quartal
sales['Date'].dt.to_period('Q').head(3)
```

```
0    2021Q1
1    2021Q1
2    2021Q1
Name: Date, dtype: period[Q-DEC]
```

Missing & Duplicates

Missing Value

Tahapan pertama dalam berhadapan dengan data *missing* adalah mengetahui apakah terdapat data *missing* atau tidak. Terdapat juga sebuah fungsi yang dapat digunakan secara spesifik untuk mengecek nilai *missing* pada data, yaitu dengan menggabungkan 2 fungsi `.isna()` & `.sum()`

```
# Mengecek apakah ada data missing atau tidak
sales.isna().sum()
```

```
Date                  0
Quantity Product 1    0
Quantity Product 2    0
Quantity Product 3    0
Quantity Product 4    0
Sales Product 1       0
Sales Product 2       0
Sales Product 3       0
Sales Product 4       0
dtype: int64
```

Dari hasil pengamatan data di atas, dapat disimpulkan bahwa pada data yang digunakan tidak memiliki nilai kosong sama sekali.

Akan tetapi jika terdapat data kosong, terdapat beberapa pendekatan yang dapat dilakukan, seperti:

1. **Hapus baris atau kolom**

Salah satu metode dalam berhadapan dengan nilai *missing* adalah dengan langsung menghapus saja setiap baris yang berisikan nilai *missing, dengan syarat nilai missing pada kolom tersebut **kurang dari 5%** dari total keseluruhan baris.

Fungsi yang akan digunakan adalah fungsi `.dropna()`.

2. ***Replace/Input* NA dengan nilai `mean`, `median`, dll**

Metode lainnya yang dapat digunakan dalam berhadapan dengan nilai *missing* adalah dengan melakukan imputasi atau pengisian dengan nilai dummy berdasarkan informasi data yang lainnya.

Hal tersebut dapat dilakukan dengan menggunakan metode `.fillna()`

3. **Tetap mempertahankan data kita**

Duplicates Value

Tahapan pertama dalam berhadapan dengan data *duplicate* adalah mengetahui apakah terdapat data *duplicate* atau tidak. Terdapat juga fungsi yang dapat digunakan secara spesifik untuk mengecek nilai *duplicate* pada data, yaitu dengan menggabungkan 2 fungsi `.duplicated()` & `.sum()`

```
# Mengecek apakah ada data duplikat atau tidak
sales.duplicated().sum()
```

```
0
```

Untuk menangani data yang *duplicate*, kita bisa menggunakan method `drop_duplicate()`. Cara ini membuat observasi yang duplicated terhapus dan bisa juga untuk mengatur observasi mana yang akan tetap disimpan.

Terdapat 2 macam cara untuk melakukan penghapusan pada nilai duplicate.

1. Dengan menambahkan parameter `keep='first'`, maka akan mempertahankan baris teratas dari nilai yang *duplicate*.
2. Dengan menambahkan parameter `keep='last'`, maka akan mempertahankan baris terbawah dari nilai yang *duplicate*.

Slicing

`[]` operator

Digunakan untuk melakukan subsetting dengan cara mengiris (slicing) index pada dataframe. Formula penulisannya adalah `[start:end]` dengan mengikuti aturan indexing pada python (dimulai dari 0) dimana `start` inclusive dan `end` exclusive.

- start -> inclusif -> urutan index baris masuk ke output
- end -> eksklusif -> urutan index baris enggak termasuk output

```
# Pada data, kita ingin mengambil dari urutan ke2-ke4
sales[1:4]
```

	Date	Quantity Product 1	Quantity Product 2	Quantity Product 3	Quantity Product 4	Sales Produ
1	2021-01-02	5418	1294	1071	1492	17175
2	2021-01-03	6718	649	4878	1380	21296
3	2021-01-04	6964	1122	2730	428	22075

`.loc` dan `.iloc`

Dengan menggunakan `.loc` dan `iloc` kita dapat melakukan pengirisan pada index **baris dan kolom**.

Perbedaan yang mendasar dari kedua operator ini adalah: - `.iloc` merujuk pada lokasi **index** baris atau kolomnya sehingga harus **integer**, sedangkan - `.loc` merujuk pada lokasi **nama** baris atau kolomnya

```
# iloc, mengambil baris 2 & kolom 3
sales.iloc[1, 2]
```

```
1294
```

```
# loc, mengambil baris 2 & kolom 3
sales.loc[1, "Quantity Product 2"]
```

```
1294
```

Conditional Subsetting

Conditional subsetting di `pandas` adalah teknik untuk mengekstrak subset data dari DataFrame atau Series berdasarkan kondisi tertentu. Ini memungkinkan untuk memilih/mengambil data yang memenuhi kriteria spesifik, seperti nilai dalam kolom yang melebihi ambang batas tertentu, teks yang mengandung kata tertentu, atau gabungan dari beberapa kondisi.

Syntax penulisan untuk conditional subsetting adalah:

```
df[df['column_name'] <comparison_operator> <value>]
```

Contoh comparison_operator adalah `==`, `!=`, `>`, `>=`, `<`, `<=`.

- `==`: untuk mengambil nilai yang sama
- `!=`: untuk mengambil nilai yang tidak sama (misal !=supermarket -> minimarket & hypermarket)
- `>`: untuk mengambil sebuah nilai angka yang lebih besar (misalkan > 200 -> 201)
- `>=`: untuk mengambil sebuah nilai angka yang sama dan lebih besar dari ketentuannya (misal >= 200 -> 200,201,..)
- `<`: untuk mengambil sebuah nilai angka yang lebih kecil (misalkan < 200 -> 199)
- `<=`: untuk mengambil sebuah nilai angka yang sama dan lebih kecil dari ketentuannya (misal <= 200 -> 200,199,..)

```
# Conditional Subsetting untuk mengambil data pada tanggal 2021-01-01
sales[sales['Date'] == "2021-01-1"]
```

	Date	Quantity Product 1	Quantity Product 2	Quantity Product 3	Quantity Product 4	Sales Produ
0	2021-01-01	7693	2394	2384	1367	24386

Additional Information:

Terdapat operator `&` (AND) dan `|` (OR) untuk melakukan subsetting lebih dari 1 kondisi. Misalnya hasil yang ingin diamati adalah data penjualan dari seorang pegawai bernama Moana yang jumlahnya lebih dari 5000, maka dapat menggunakan syntax:

```
sales[(sales.salesperson == 'Moana') & (sales.amount > 5000)]
```

Untuk subsetting dengan kondisi lebih dari 1, setiap kondisi diletakkan **di dalam tanda kurung `()`** atau bisa ditulis dengan syntax berikut:

```
df[(kondisi pertama) operator (kondisi kedua) operator (kondisi ketiga) dan seterusnya...]
```

Poin: - Operator `AND`: harus semua kondisi terpenuhi dalam satu baris agar muncul - Operator `OR`: salah satu kondisi saja sudah terpenuhi maka baris tersebut muncul

Contoh sederhana penggunaan OR dan AND

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

df
```

	angka	huruf
0	1	a
1	2	b
2	3	c

OPERATOR AND

Menampilkan baris dengan **angka** lebih besar dari 1 dan dengan **huruf** b

```
#code here
df[(df['angka'] > 1) & (df['huruf'] == 'b')]
```

	angka	huruf
1	2	b

OPERATOR OR

Menampilkan baris dengan **huruf** a dan b

```
#code here
df[(df['huruf'] == 'a') | (df['huruf'] == 'b')]
```

	angka	huruf
0	1	a
1	2	b

Aggregasi Data

Agregasi data adalah proses meringkas atau menggabungkan data dalam sebuah DataFrame, sehingga kita bisa mendapatkan informasi seperti total, rata-rata, jumlah, dan lainnya dari kelompok data tertentu.

`crosstab()` adalah fungsi di `pandas` yang digunakan untuk menghitung frekuensi atau meringkas data dengan cepat dalam bentuk tabel silang. Ini sangat mirip dengan fitur tabel pivot di Excel, tetapi lebih spesifik untuk menghitung frekuensi atau agregasi dengan dua atau lebih kategori.

Berikut adalah syntax:

```
pd.crosstab(index= nama_df['nama_kolom'], columns= nama_df['nama_kolom'])
```

Dimana tujuan dari setiap parameternya :

- `index` : kolom yang akan dijadikan index baris
- `columns` : kolom yang akan dijadikan index kolom

Sebagai contoh, analisa yang ingin dilakukan adalah berapakah total frekuensi dari setiap quarter untuk kurun waktu 2 tahun terakhir.

```
# Membuat kolom baru untuk informasi quarter
sales['Quarter'] = sales['Date'].dt.to_period('Q')
sales['Quarter'].head(3)
```

```
0    2021Q1
1    2021Q1
2    2021Q1
Name: Quarter, dtype: period[Q-DEC]
```

```
# Menghitung frekuensi
pd.crosstab(index = sales['Quarter'],
            columns = 'Frekuensi',
            colnames = [None]) # Parameter tambahan untuk menghilangkan nama kolom col_0
```

Quarter	Frekuensi
2021Q1	90
2021Q2	91
2021Q3	91
2021Q4	90

Quarter	Frekuensi
2022Q1	90
2022Q2	91
2022Q3	91
2022Q4	90

Selain dari memanfaatkan metode perhitungan `crosstab()`. Metode `groupby()` di `Pandas` dapat digunakan juga untuk mengelompokkan data berdasarkan satu atau lebih kolom. Setelah data dikelompokkan, kita dapat menerapkan berbagai operasi agregasi, seperti `sum()`, `mean()`, `count()`, `max()`, `min()`, dan lainnya pada setiap kelompok data.

Sebagai contoh, analisa yang ingin dilakukan adalah menghitung rata-rata penjualan untuk setiap produk yang ada.

```
# Melakukan perhitungan rata-rata
sales.groupby('Quarter').mean(numeric_only=True)[['Sales Product 1', 'Sales Product 2', 'Sale
```

Quarter	Sales Product 1	Sales Product 2	Sales Product 3	Sales Product 4
2021Q1	14210.123778	13993.788889	16763.999778	7787.227556
2021Q2	13492.948242	13354.548132	17372.231648	8669.609890
2021Q3	12842.715055	12830.000220	16462.565055	7878.258242
2021Q4	12029.375111	14512.753111	18960.906444	7640.983333
2022Q1	13062.830333	12576.798889	16093.365111	7914.300000
2022Q2	12764.614615	13531.441099	17091.285055	7411.830879
2022Q3	12348.438901	13916.718022	15793.880000	7587.260220
2022Q4	12556.193889	12970.512889	17250.294222	7591.073333

Selain dari menggunakan crosstab, `groupby` dan `agg` adalah dua fungsi yang dapat dimanfaatkan juga untuk mengelompokkan data dan melakukan agregasi (seperti `sum`, `mean`, `count`, dll.) pada setiap kelompok. Berikut ini penjelasan mengenai cara penggunaannya:

- `groupby` digunakan untuk mengelompokkan data berdasarkan satu atau lebih kolom.
- `agg` digunakan setelah `groupby` untuk menerapkan fungsi agregasi (seperti `sum`, `mean`, `min`, `max`, dll.) pada setiap kelompok.

```
# Membuat DataFrame
data = {'Kategori': ['A', 'B', 'A', 'B', 'A', 'B'],
        'Nilai': [10, 20, 30, 40, 50, 60],
        'Jumlah': [1, 2, 3, 4, 5, 6]}

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

	Kategori	Nilai	Jumlah
0	A	10	1
1	B	20	2
2	A	30	3
3	B	40	4
4	A	50	5
5	B	60	6

```
# Group by 'Kategori' dan lakukan agregasi
df.groupby('Kategori').agg(
    Total_Nilai=('Nilai', 'sum'),
    Rata_Rata_Jumlah=('Jumlah', 'mean')
)
```

Kategori	Total_Nilai	Rata_Rata_Jumlah
A	90	3.0
B	120	4.0

Pembuatan Sisualisasi Sederhana Dengan `matplotlib`

`Matplotlib` adalah library yang kuat dan fleksibel untuk membuat visualisasi data di Python. Dengan `Matplotlib`, dapat membantu untuk membuat berbagai jenis grafik untuk menganalisis dan memahami data.

```
# Persiapan library
import matplotlib.pyplot as plt
```

Pembuatan visualisasi dengan bantuan library `matplotlib`, fungsi yang dapat digunakan adalah

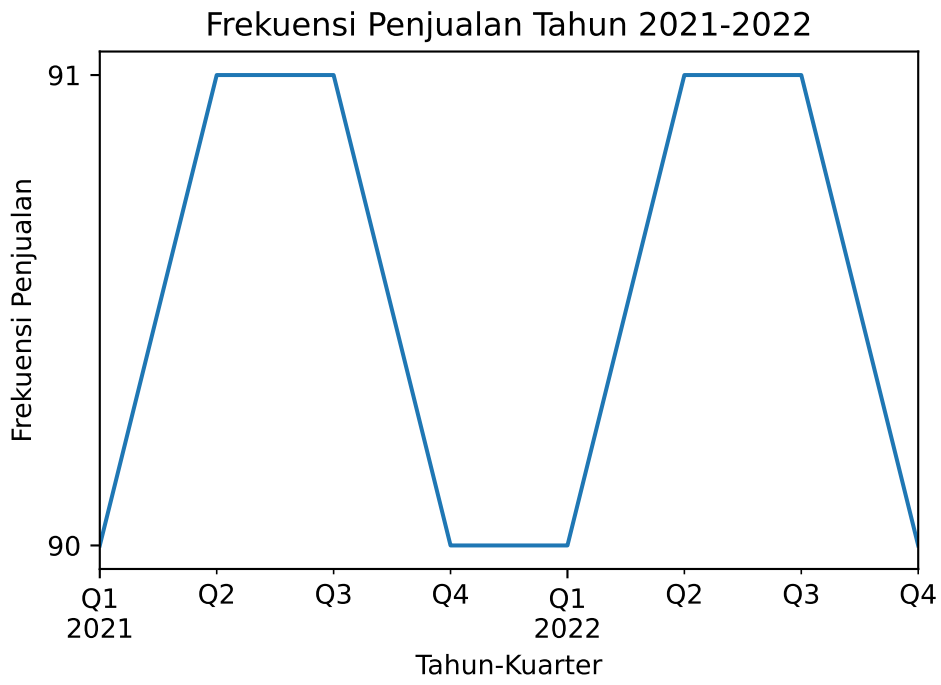
- Line Plot: `plt.plot(x, y)`
- Bar Plot: `plt.bar(categories, values)`
- Scatter Plot: `plt.scatter(x, y)`
- Histogram: `plt.hist(data, bins=10)`
- Box Plot: `plt.boxplot(data)`

```
# Mempersiapkan dataframe
frekuensi = pd.crosstab(index = sales['Quarter'],
                        columns = 'Frekuensi',
                        colnames = [None])

# Membuat visualisasi
frekuensi.plot()

# Melakukan kustomisasi
plt.title('Frekuensi Penjualan Tahun 2021-2022') # Menambahkan judul
plt.xlabel('Tahun-Kuarter') # Mengatur nama sumbu x
plt.ylabel('Frekuensi Penjualan') # Mengatur nama sumbu y
plt.yticks(range(90, 92, 1))  # Mengatur interval sumbu y
plt.legend().remove() # Menghilangkan legenda

# Menampilkan visualisasi
plt.show()
```



Format File Quarto Markdown

Quarto adalah sebuah sistem untuk membuat dokumen, laporan, artikel, buku, situs web, dan presentasi yang interaktif dan dinamis menggunakan markdown. Quarto dirancang untuk ilmuwan data, penulis teknis, dan peneliti yang bekerja dengan berbagai bahasa pemrograman seperti R, Python, dll.

Perbedaan Quarto dengan R Markdown

Quarto Markdown memiliki perbedaan yang paling signifikan dengan R Markdown pada fleksibilitas terhadap basaha pemrograman yang digunakan dan fitur yang lebih moderen.

1. **Dukungan Multi-bahasa yang Lebih Baik**

Walaupun R Markdown juga memiliki kapabilitas untuk digunakan dengan bahasa selain R, akan tetapi pemanfaatan Quarto Markdown untuk menjalankan bahasa pemograman lainnya selain R, hal itu dikarenakan Quarto Markdown memang mendukung untuk menjalankan bahasa Python, Julia, dan Observable (JavaScript) secara native, tidak seperti R Markdown yang secara khusus dipersiapkan untuk bahasa R saja ketika pertama kali dikembangkan.

2. **Fitur Modern What You See Is What You Mean**

Selain dari kemampuannya untuk bekerja dengan banyak bahasa, Quarto juga memiliki pengembangan pada proses pembuatan dokumen yang memanfaatkan konsep *What You See Is What You Mean.*

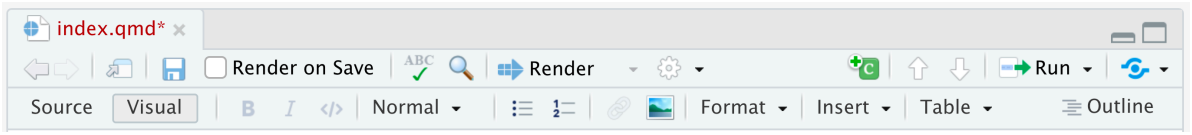
What You See Is What You Mean (WYSIWYM) adalah konsep dalam desain antarmuka pengguna dan pembuatan konten yang berfokus pada penyediaan umpan balik visual yang langsung dan akurat tentang struktur semantik dari informasi yang sedang dibuat atau diedit oleh pengguna. Dengan adanya konsep ini para pengguna harapannya akan lebih merasa produktif.

- **Mengganti Tampilan**

Dokumen markdown dapat diedit dalam mode sumber (source) atau visual. Untuk beralih ke mode visual pada dokumen tertentu, gunakan tombol Source atau Visual di bagian kiri atas toolbar dokumen (atau sebagai alternatif, gunakan pintasan keyboard F4):

Gambar cuplikan jendela RStudio yang menunjukkan bilah opsi di bagian atas dokumen Quarto.

```
display(Image(filename="assets/visual-editing-switch-modes.png"))
```



- **Keyboard Shortcuts**

Dengan memanfaatkan mode visual, pengguna Quarto Markdown diberikan tambahan fitur untuk menggunakan *keyboard shortcut* tanpa harus menggunakan pengaturan markdown yang relatif lebih konvensional.

Command	Keyboard Shortcut	Markdown Shortcut
Bold	Ctrl / ⌃ + B	**bold**
Italic	Ctrl / ⌃ + I	*italic*
Code	Ctrl / ⌃ + D	`code`
Header	Ctrl + Alt / + ⌃ + 1	Heading 1

Untuk referensi yang lebih lengkapnya bisa merujuk ke dokumentasi berikut ini.

- ***Editor Toolbar***