

Classification Model

Team Algoritma

December 25, 2023

Coursebook: Classification in Machine Learning

- Bagian 5 Audit Analytics untuk Bank Rakyat Indonesia
- Durasi: 7 Jam
- *Last Updated*: December 2023

-
- *Coursebook* ini disusun dan dikurasi oleh tim produk dan instruktur dari [Algoritma Data Science School](#)

1 Background

Coursebook ini merupakan bagian dari **BRI Audit Analytics** yang disiapkan oleh [Algoritma](#). *Coursebook* ini ditujukan hanya untuk khalayak terbatas, yaitu individu dan organisasi yang menerima *coursebook* ini langsung dari organisasi pelatihan. Tidak boleh direproduksi, didistribusikan, diterjemahkan, atau diadaptasi dalam bentuk apapun di luar individu dan organisasi ini tanpa izin. Algoritma adalah pusat pendidikan *data science* yang berbasis di Jakarta. Kami menyelenggarakan *workshop* dan program pelatihan untuk membantu para profesional dan mahasiswa dalam menguasai berbagai sub-bidang *data science* yaitu: *data visualization*, *machine learning*, statistik, dan lain sebagainya.

2 Classification in Machine Learning

2.1 Training Objectives

Pada *workshop* ini, Anda akan mempelajari salah satu *case* yang cukup umum diselesaikan dengan *machine learning*, yaitu klasifikasi. Klasifikasi merupakan aplikasi dari *supervised learning* (cabang dari *machine learning*), dengan model diharapkan dapat memberikan *output* berdasarkan kategori yang sudah ditentukan. Misalnya:

- Mengidentifikasi apakah suatu *email* merupakan *spam* atau tidak.
- Mengidentifikasi apakah pelanggan akan membeli suatu produk atau tidak.
- Mengidentifikasi objek buah-buahan pada suatu gambar, apakah objek tersebut merupakan apel, pisang, atau mangga.

Anda akan mempelajari siklus proses dalam membangun model *machine learning* untuk klasifikasi dan konsep matematika di belakangnya. *Workshop* ini akan berfokus pada 3 model *machine learning*: *logistic regression*, *decision tree*, dan *random forest*. Dengan durasi selama 7 jam, pembelajaran pada *workshop* ini terbagi dalam beberapa modul:

- **Data Preprocessing**
 - Target and Predictors Splitting
 - Dummy Variables for Categorical Predictor
 - Cross Validation
- **Classification Concepts for Logistic Regression**
 - Probability concept
 - Understanding log of odds
 - Understanding logit function
- **Logistic Regression Implementation**
 - Logistic regression with discrete predictor variables
 - Logistic regression with one continuous predictor variables
 - Logistic regression with multiple predictors variable
 - Assumption of logistic regression
- **Performance Evaluation and Model Selection**
 - Confusion Matrix
 - * Accuracy
 - * Sensitivity
 - * Recall
 - * Specificity
- **K - Nearest Neighbors**
 - Euclidean distance
 - Choosing an appropriate K
 - Features rescaling
 - * Min-max normalization
 - * Z-score standardization
 - Characteristics of K-NN

```
[1]: # Package
'''
Package dapat dibayangkan sebagai sekumpulan program yang telah ditulis
↳ seseorang sehingga
dapat digunakan oleh orang lain yang ingin menyelesaikan permasalahan yang
↳ serupa.
'''

import warnings

# package untuk perhitungan matematika
import math
import numpy as np

# package untuk persiapan data
import pandas as pd

# package untuk visualisasi data
import matplotlib.pyplot as plt
import seaborn as sns
```

```
# package untuk keperluan modeling
import statsmodels.api as sm
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, precision_score, recall_score
from sklearn.preprocessing import StandardScaler
from sklearn.neighbors import KNeighborsClassifier
```

```
[2]: # pandas display setup
pd.set_option('display.max_rows', 500)
pd.set_option('display.max_columns', 500)
pd.set_option('display.width', 1000)

# suppress scientific notation
np.set_printoptions(suppress=True) # numpy output
pd.set_option('display.float_format', lambda x: '%.5f' % x) # pandas output

# suppress warning
warnings.filterwarnings("ignore")
%matplotlib inline
```

3 Data Preparation and Exploratory Data

Di dunia nyata, sering kali terdapat permasalahan pada data mentah yang dapat mengganggu proses analisis, seperti kesalahan *input*, nilai yang hilang, atau format penulisan yang tidak konsisten. Untuk mengatasi permasalahan tersebut, dilakukan preparasi data.

Preparasi data merupakan serangkaian proses mengubah data mentah menjadi format yang lebih siap dan sesuai untuk analisis lebih lanjut. Preparasi data merupakan proses yang penting dan harus dilakukan sebelum analisis dan pembuatan model *machine learning*.

Berikut beberapa *step* yang sering dilakukan pada tahap preparasi data:

1. Cek tipe data setiap kolom.
2. Cek data yang hilang/*missing values*.
3. *Feature selection*.

3.1 Read The Dataset

Pada *step* ini, kita akan memuat data mentah yang akan digunakan sepanjang proses analisis dan pembuatan model *machine learning*. Package **pandas** menyediakan banyak fungsi yang dapat kita manfaatkan untuk membaca data, tergantung format data yang kita miliki.

- Format **.xlsx** menggunakan `pd.read_excel()`.
- Format **.csv** menggunakan `pd.read_csv()`.
- Format data lainnya yang dapat dilihat [di sini](#).

Note: Selama proses persiapan dan pemrosesan data, kita akan banyak menggunakan fungsi yang terdapat pada package **pandas**.

Kita akan menggunakan *dataset* `credit100.csv`. Data ini merupakan data akun bank dari research paper metode evaluasi machine learning [berikut ini](#). Secara detailnya data ini bisa dilihat lebih lanjut pada [link GitHub ini](#).

Karena *dataset* ini memiliki format `.csv`, kita akan menggunakan fungsi `pd.read_csv()`.

```
[3]: credit = pd.read_csv("data_input/credit100.csv")
credit = credit.drop(columns=["device_fraud_count"])

# menampilkan 5 baris teratas
credit.head()
```

```
[3]: fraud_bool income name_email_similarity prev_address_months_count
current_address_months_count customer_age days_since_request
intended_balcon_amount payment_type zip_count_4w velocity_6h velocity_24h
velocity_4w bank_branch_count_8w date_of_birth_distinct_emails_4w
employment_status credit_risk_score email_is_free housing_status
phone_home_valid phone_mobile_valid bank_months_count has_other_cards
proposed_credit_limit foreign_request source session_length_in_minutes
device_os keep_alive_session device_distinct_emails_8w month
0 0 0.90000 0.61579 -1
51 40 0.01034 -0.69955 AB
2659 11007.70883 7448.79742 5576.30356 2030
7 CA 253 1 BA
1 1 9 1 1500.00000
0 INTERNET 13.07981 linux 0
1 1
1 0 0.60000 0.07246 -1
125 20 0.00697 -0.95947 AC
5599 6304.26284 4373.09359 4721.10002 1
13 CA 112 1 BE
1 0 -1 0 200.00000
0 INTERNET 6.66655 windows 0
1 4
2 1 0.50000 0.07659 -1
37 50 0.00632 -1.21993 AB
1352 1357.41355 6667.02966 6318.87626 15
6 CF 199 1 BC
1 1 28 0 1500.00000
0 INTERNET 4.49267 windows 1
2 0
3 0 0.30000 0.25543 23
19 20 0.00927 3.24224 AA
3743 5234.26230 6139.48958 4201.72592 1
14 CA 63 1 BC
0 1 16 0 200.00000
0 INTERNET 2.32189 other 1
1 3
```

4		0 0.90000		0.19236		-1
34		30		0.01924		-1.13156 AB
546	3589.15345	4574.39557	4318.25891		16	
5		CA	275		1	BC
1		1		1	1	1000.00000
0	INTERNET		4.84515	linux		1
1	5					

Penjelasan Dataset

Berikut adalah penjelasan setiap kolom yang terdapat pada *dataset*:

- **income** (numeric): *Annual income of the applicant (in decile form). Ranges between [0.1, 0.9].*
- **name_email_similarity** (numeric): *Metric of similarity between email and applicant's name. Higher values represent higher similarity. Ranges between [0, 1].*
- **prev_address_months_count** (numeric): *Number of months in previous registered address of the applicant, i.e. the applicant's previous residence, if applicable. Ranges between [-1, 380] months (-1 is a missing value).*
- **current_address_months_count** (numeric): *Months in currently registered address of the applicant. Ranges between [-1, 429] months (-1 is a missing value).*
- **customer_age** (numeric): *Applicant's age in years, rounded to the decade. Ranges between [10, 90] years.*
- **days_since_request** (numeric): *Number of days passed since application was done. Ranges between [0, 79] days.*
- **intended_balcon_amount** (numeric): *Initial transferred amount for application. Ranges between [-16, 114] (negatives are missing values).*
- **payment_type** (categorical): *Credit payment plan type. 5 possible (anonymized) values.*
- **zip_count_4w** (numeric): *Number of applications within same zip code in last 4 weeks. Ranges between [1, 6830].*
- **velocity_6h** (numeric): *Velocity of total applications made in last 6 hours i.e., average number of applications per hour in the last 6 hours. Ranges between [-175, 16818].*
- **velocity_24h** (numeric): *Velocity of total applications made in last 24 hours i.e., average number of applications per hour in the last 24 hours. Ranges between [1297, 9586].*
- **velocity_4w** (numeric): *Velocity of total applications made in last 4 weeks, i.e., average number of applications per hour in the last 4 weeks. Ranges between [2825, 7020].*
- **bank_branch_count_8w** (numeric): *Number of total applications in the selected bank branch in last 8 weeks. Ranges between [0, 2404].*
- **date_of_birth_distinct_emails_4w** (numeric): *Number of emails for applicants with same date of birth in last 4 weeks. Ranges between [0, 39].*
- **employment_status** (categorical): *Employment status of the applicant. 7 possible (anonymized) values.*
- **credit_risk_score** (numeric): *Internal score of application risk. Ranges between [-191, 389].*
- **email_is_free** (binary): *Domain of application email (either free or paid).*
- **housing_status** (categorical): *Current residential status for applicant. 7 possible (anonymized) values.*
- **phone_home_valid** (binary): *Validity of provided home phone.*
- **phone_mobile_valid** (binary): *Validity of provided mobile phone.*

- `bank_months_count` (numeric): *How old is previous account (if held) in months. Ranges between $[-1, 32]$ months (-1 is a missing value).*
- `has_other_cards` (binary): *If applicant has other cards from the same banking company.*
- `proposed_credit_limit` (numeric): *Applicant's proposed credit limit. Ranges between $[200, 2000]$.*
- `foreign_request` (binary): *If origin country of request is different from bank's country.*
- `source` (categorical): *Online source of application. Either browser (INTERNET) or app (TELEAPP).*
- `session_length_in_minutes` (numeric): *Length of user session in banking website in minutes. Ranges between $[-1, 107]$ minutes (-1 is a missing value).*
- `device_os` (categorical): *Operative system of device that made request. Possible values are: Windows, macOS, Linux, X11, or other.*
- `keep_alive_session` (binary): *User option on session logout.*
- `device_distinct_emails` (numeric): *Number of distinct emails in banking website from the used device in last 8 weeks. Ranges between $[-1, 2]$ emails (-1 is a missing value).*
- `device_fraud_count` (numeric): *Number of fraudulent applications with used device. Ranges between $[0, 1]$.*
- `month` (numeric): *Month where the application was made. Ranges between $[0, 7]$.*
- `fraud_bool` (binary): *If the application is fraudulent or not.*

3.2 Descriptive Statistics

Pemeriksaan statistik deskriptif merupakan analisis awal yang dilakukan untuk mendapatkan pemahaman tentang sifat dasar dari *dataset*. Seperti namanya, statistik deskriptif mencakup penghitungan ringkasan statistik yang menggambarkan ciri-ciri kumpulan data tersebut, seperti: ukuran pemusatan data (*mean* dan *median*) dan ukuran penyebaran data (standar deviasi).

Meskipun perhitungan yang ditampilkan cukup sederhana, statistik deskriptif bisa menjadi alternatif *tools* yang cukup *powerful*. Sebagai contoh, kita dapat mengetahui kecenderungan persebaran data dan melakukan deteksi awal apakah terdapat nilai pencilan (*outlier*) yang berbeda jauh dibandingkan nilai-nilai lainnya.

Untuk menampilkan statistik deskriptif pada data `credit`, kita dapat menggunakan fungsi `.describe()`.

```
[4]: credit.describe()
```

```
[4]:      fraud_bool      income  name_email_similarity
prev_address_months_count  current_address_months_count  customer_age
days_since_request  intended_balcon_amount  zip_count_4w  velocity_6h
velocity_24h  velocity_4w  bank_branch_count_8w
date_of_birth_distinct_emails_4w  credit_risk_score  email_is_free
phone_home_valid  phone_mobile_valid  bank_months_count  has_other_cards
proposed_credit_limit  foreign_request  session_length_in_minutes
keep_alive_session  device_distinct_emails_8w      month
count 100000.00000 100000.00000      100000.00000
100000.00000      100000.00000 100000.00000      100000.00000
100000.00000 100000.00000 100000.00000 100000.00000 100000.00000
100000.00000      100000.00000      100000.00000 100000.00000
```

100000.00000	100000.00000	100000.00000	100000.00000	
100000.00000	100000.00000	100000.00000	100000.00000	
100000.00000	100000.00000			
mean	0.11029	0.57243	0.48334	
15.67167		89.09414	34.34520	1.02909
8.05770	1579.65136	5622.74948	4751.70038	4846.51788
178.94636		9.31030	135.47810	0.54335
0.39658	0.88576	10.76559	0.20875	
545.64380	0.02764	7.64760		0.55407
1.02486	3.30663			
std	0.31325	0.29097	0.29139	
43.04344		88.27788	12.31084	5.39503
19.80150	1003.51929	3006.95237	1476.12014	926.53103
455.26869		5.05074	72.37795	0.49812
0.48919	0.31810	12.18756	0.40642	
513.04025	0.16394	8.28959		0.49707
0.20006	2.22212			
min	0.00000	0.10000	0.00006	
-1.00000		-1.00000	10.00000	0.00000
-14.06498	1.00000	8.67828	1320.28399	2863.78334
0.00000		0.00000	-170.00000	0.00000
0.00000	0.00000	-1.00000	0.00000	
190.00000	0.00000	-1.00000		0.00000
-1.00000	0.00000			
25%	0.00000	0.30000	0.21060	
-1.00000		22.00000	20.00000	0.00706
-1.18451	899.00000	3388.04839	3578.88819	4260.40529
1.00000		5.00000	85.00000	0.00000
0.00000	1.00000	-1.00000	0.00000	
200.00000	0.00000	3.11938		0.00000
1.00000	1.00000			
50%	0.00000	0.60000	0.47765	
-1.00000		56.00000	30.00000	0.01492
-0.83992	1272.00000	5272.14754	4734.07444	4904.60572
9.00000		9.00000	126.00000	1.00000
0.00000	1.00000	5.00000	0.00000	
200.00000	0.00000	5.09977		1.00000
1.00000	3.00000			
75%	0.00000	0.80000	0.74887	
11.00000		133.00000	40.00000	0.02607
-0.19746	1963.00000	7644.29673	5733.29879	5483.66273
23.00000		13.00000	185.00000	1.00000
1.00000	1.00000	25.00000	0.00000	
1000.00000	0.00000	8.87309		1.00000
1.00000	5.00000			
max	1.00000	0.90000	1.00000	
365.00000		424.00000	90.00000	76.57727

```

112.25311    6521.00000  16665.35953    9506.89660    6994.76420
2322.00000                                37.00000            378.00000            1.00000
1.00000                1.00000                32.00000            1.00000
2100.00000                1.00000                                82.00473            1.00000
2.00000        7.00000

```

`.describe()` akan mengeluarkan perhitungan:

1. Banyaknya baris pada setiap kolom (`count`).
2. Rata-rata setiap kolom (`mean`).
3. Standar deviasi setiap kolom (`std`).
4. Nilai minimum (`min`) dan maksimum (`max`) setiap kolom.
5. Kuartil bawah (25%), median (50%), dan kuartil atas (75%) setiap kolom.

3.3 Data Types Inspection and Preparation

Pemeriksaan tipe data adalah proses memverifikasi jenis data dalam setiap kolom dari *dataset* untuk memastikan bahwa setiap kolom sesuai dengan jenis data yang diharapkan. Misalnya, kolom yang berisi tanggal harus diidentifikasi sebagai tipe data tanggal/waktu, sementara kolom yang berisi teks harus diidentifikasi sebagai *string*, dan kolom dengan angka harus diidentifikasi sebagai bilangan bulat (*integer*) atau bilangan berkoma (*floating point*) tergantung pada kebutuhan analisis.

Untuk memeriksa tipe setiap kolom, kita bisa menggunakan `.dtypes`.

```
[5]: credit.dtypes
```

```

[5]: fraud_bool                int64
income                        float64
name_email_similarity         float64
prev_address_months_count    int64
current_address_months_count int64
customer_age                 int64
days_since_request           float64
intended_balcon_amount        float64
payment_type                  object
zip_count_4w                  int64
velocity_6h                   float64
velocity_24h                  float64
velocity_4w                   float64
bank_branch_count_8w          int64
date_of_birth_distinct_emails_4w int64
employment_status             object
credit_risk_score              int64
email_is_free                 int64
housing_status                object
phone_home_valid              int64
phone_mobile_valid            int64
bank_months_count             int64
has_other_cards               int64

```



```

proposed_credit_limit      float64
foreign_request            int64
source                    object
session_length_in_minutes  float64
device_os                  object
keep_alive_session         int64
device_distinct_emails_8w  int64
month                     int64
dtype: object

```

Beberapa kolom yang bertipe kategorikal berdasarkan deskripsi data sebelumnya perlu kita ubah terlebih dahulu. Pengubahan tipe data ini menggunakan fungsi `.astype()`. Detail lebih lanjut terkait fungsi `.astype()`, Anda dapat mengunjungi [referensi berikut](#).

```

[6]: # mengubah tipe data sebagai kategorikal

cat_cols = [
    'payment_type', 'employment_status', 'housing_status', 'source', 'device_os']

credit[cat_cols] = credit[cat_cols].astype('category')

```

```

[7]: # cek tipe data setelah diubah

credit.dtypes

```

```

[7]: fraud_bool      int64
income              float64
name_email_similarity  float64
prev_address_months_count  int64
current_address_months_count  int64
customer_age        int64
days_since_request  float64
intended_balcon_amount  float64
payment_type        category
zip_count_4w        int64
velocity_6h         float64
velocity_24h        float64
velocity_4w         float64
bank_branch_count_8w  int64
date_of_birth_distinct_emails_4w  int64
employment_status    category
credit_risk_score     int64
email_is_free        int64
housing_status        category
phone_home_valid      int64
phone_mobile_valid    int64
bank_months_count     int64
has_other_cards       int64

```

proposed_credit_limit	float64
foreign_request	int64
source	category
session_length_in_minutes	float64
device_os	category
keep_alive_session	int64
device_distinct_emails_8w	int64
month	int64
dtype:	object

Note: perlu diingat kembali bahwa terdapat dua tipe data: numerikal dan kategorikal. Khusus kolom kategorik, terdapat pemrosesan tambahan yang akan dibahas lebih lanjut pada bagian *Data Preprocessing*.

3.4 Check Missing Values

Pemeriksaan nilai *null/missing value* adalah proses untuk mendeteksi keberadaan nilai *null* dalam sebuah *dataset*. Nilai *null* adalah entri yang tidak memiliki data di dalamnya. Hal ini bisa berarti bahwa informasi tersebut tidak diketahui, tidak ada, atau tidak berlaku. Dalam konteks pemrograman dan analisis data, nilai *null* ini sering kali diwakili dengan nilai khusus seperti `NULL`, `None`, `NaN` (*Not a Number*).

Untuk melihat apakah terdapat nilai *null* pada data, kita dapat menggunakan sintaks berikut.

```
[8]: credit.isna().sum()
```

```
[8]: fraud_bool      0
     income           0
     name_email_similarity  0
     prev_address_months_count  0
     current_address_months_count  0
     customer_age      0
     days_since_request  0
     intended_balcon_amount  0
     payment_type      0
     zip_count_4w       0
     velocity_6h        0
     velocity_24h       0
     velocity_4w        0
     bank_branch_count_8w  0
     date_of_birth_distinct_emails_4w  0
     employment_status  0
     credit_risk_score  0
     email_is_free      0
     housing_status     0
     phone_home_valid    0
     phone_mobile_valid  0
     bank_months_count   0
```

```

has_other_cards          0
proposed_credit_limit    0
foreign_request          0
source                  0
session_length_in_minutes 0
device_os                0
keep_alive_session       0
device_distinct_emails_8w 0
month                   0
dtype: int64

```

Fungsi `.isna().sum()` digunakan untuk melihat banyaknya nilai *null* di setiap kolom. Terlihat pada *output* yang ditampilkan bahwa data kita tidak memiliki nilai *null* sehingga tidak perlu dilakukan *handling missing values*.

[**Optional**] Selain menggunakan `.dtypes()` untuk memeriksa tipe data dan `.isna().sum()` untuk melihat banyak *missing values*, kita dapat menggunakan `.info()` untuk menampilkan *summary* yang lebih lengkap.

```
[9]: credit.info()
```

```

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 100000 entries, 0 to 99999
Data columns (total 31 columns):
 #   Column                                Non-Null Count  Dtype
---  -
 0   fraud_bool                            100000 non-null  int64
 1   income                                100000 non-null  float64
 2   name_email_similarity                 100000 non-null  float64
 3   prev_address_months_count            100000 non-null  int64
 4   current_address_months_count          100000 non-null  int64
 5   customer_age                         100000 non-null  int64
 6   days_since_request                   100000 non-null  float64
 7   intended_balcon_amount                100000 non-null  float64
 8   payment_type                         100000 non-null  category
 9   zip_count_4w                         100000 non-null  int64
10   velocity_6h                          100000 non-null  float64
11   velocity_24h                         100000 non-null  float64
12   velocity_4w                          100000 non-null  float64
13   bank_branch_count_8w                 100000 non-null  int64
14   date_of_birth_distinct_emails_4w     100000 non-null  int64
15   employment_status                    100000 non-null  category
16   credit_risk_score                     100000 non-null  int64
17   email_is_free                        100000 non-null  int64
18   housing_status                       100000 non-null  category
19   phone_home_valid                     100000 non-null  int64
20   phone_mobile_valid                   100000 non-null  int64
21   bank_months_count                    100000 non-null  int64
22   has_other_cards                      100000 non-null  int64

```

```

23 proposed_credit_limit      100000 non-null float64
24 foreign_request            100000 non-null int64
25 source                     100000 non-null category
26 session_length_in_minutes  100000 non-null float64
27 device_os                  100000 non-null category
28 keep_alive_session         100000 non-null int64
29 device_distinct_emails_8w  100000 non-null int64
30 month                      100000 non-null int64
dtypes: category(5), float64(9), int64(17)
memory usage: 20.3 MB

```

Fungsi `.info()` mengeluarkan *output*:

- Banyak baris pada *dataset*: data `credit` terdiri dari 100000 baris.
- Banyak kolom pada *dataset*: data `credit` terdiri dari 31 kolom.
- Nama setiap kolom dan tipe datanya.
- Banyaknya observasi yang tidak *null*. Kita dapat mengidentifikasi kolom dengan nilai *null* dari output ini. Data `credit` tidak memiliki *missing values*.

3.5 Feature Selection

Feature selection atau seleksi fitur merupakan proses pemilihan *subset* fitur (variabel atau atribut) yang paling relevan untuk pembuatan model *machine learning*. *Feature selection* bertujuan untuk memilih fitur-fitur yang paling sesuai dan mengabaikan fitur-fitur yang tidak relevan atau redundan.

Secara umum, terdapat 2 langkah untuk melakukan *feature selection*:

1. *Feature selection* berdasarkan intuisi bisnis

Artinya, kita memasukkan fitur/variabel/atribut yang menurut pandangan bisnis berpengaruh kuat dalam melakukan prediksi. Langkah ini dapat dilakukan apabila terdapat *domain knowledge*, pandangan ahli, atau kesepakatan yang dapat dijustifikasi kebenarannya.

2. *Feature selection* berdasarkan perhitungan matematika/statistika

Artinya, pemilihan fitur/variabel/atribut yang akan digunakan untuk membuat *machine learning* didasarkan pada perhitungan matematis. Sebagai contoh, *feature selection* yang berpedoman pada perhitungan korelasi setiap kolom.

Sekarang, kita akan berfokus kepada kolom-kolom numerikal data kita. Kita akan membuat sebuah *heatmap* yang menunjukkan korelasi antarkolom numerikal.

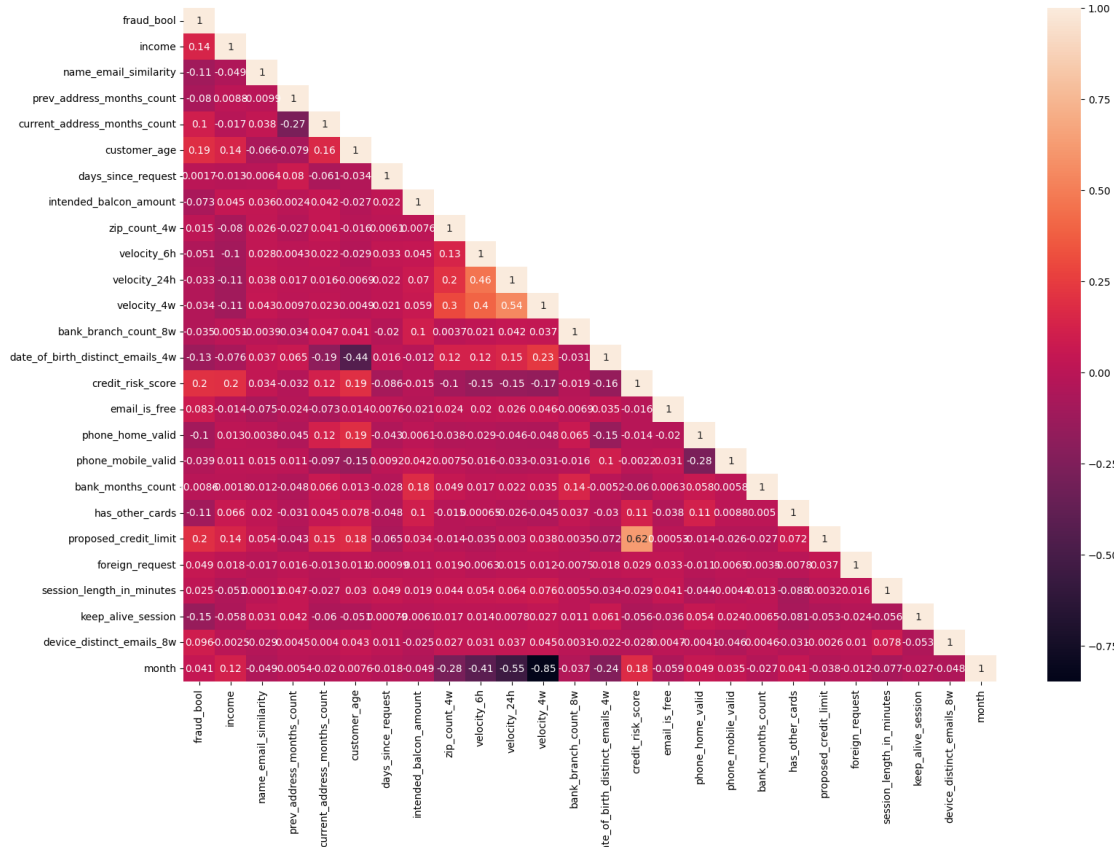
```

[10]: # cek korelasi

corr = credit.select_dtypes("number").corr()
matrix = np.triu(corr, k=1)

plt.figure(figsize = (18,12));
sns.heatmap(corr, annot=True, mask=matrix);

```



Berdasarkan gambar di atas, terlihat bahwa sebagian besar variabel memiliki korelasi yang rendah dengan variabel lainnya. Korelasi yang cukup kuat (sebesar -0.85) terdapat antara `month` dan `velocity_24h`. Idealnya, kita mengharapkan antarkolom numerikal di atas memiliki korelasi yang rendah.

Perlu diperhatikan bahwa beberapa model mengasumsikan antarprediktor tidak saling berkorelasi tinggi, seperti *logistic regression*. Akan tetapi, model yang cukup *robust*, seperti *decision tree* dan *random forest* tidak memiliki aturan yang berhubungan dengan korelasi.

Feature selection berdasarkan korelasi hanya merupakan salah satu alternatif dalam melakukan *feature selection*. Metode *feature selection* lainnya dapat dilihat pada [referensi berikut](#).

4 Data Preprocessing


Pada dasarnya, *data preprocessing* tidak berbeda jauh dengan *data preparation*. Keduanya sering digunakan secara bergantian dan sama-sama mengacu kepada serangkaian proses yang dilakukan untuk menghasilkan data dalam format yang siap digunakan untuk proses analisis lanjutan. Akan tetapi, biasanya *data preparation* memiliki cakupan yang lebih luas dan *data preprocessing* termasuk ke dalam cakupan tersebut.

Dalam konteks *course* ini, kita akan mengacu *data preprocessing* sebagai proses lanjutan yang secara spesifik mempersiapkan data untuk proses *modeling* menggunakan *machine learning*. Terdapat beberapa *step* yang umum dilakukan untuk *data preprocessing*:

1. Konversi variabel prediktor yang kategorikal menjadi *dummy variables*.
2. Pemisahan variabel prediktor dan variabel target.
3. *Train-test split*.

4.1 Dummy Variables for Categorical Predictor

Water Temperature	
A	Hot
B	Cold
C	Warm
D	Cold



Water Temperature		Dummy Variables		
		var_hot	var_warm	var_cold
A	Hot	1	0	0
B	Cold	0	0	1
C	Warm	0	1	0
D	Cold	1	0	0

Adakalanya, data kita memiliki prediktor yang kategorikal, seperti yang terlihat pada gambar. Terdapat kolom **Temperature** dengan tiga variasi nilai: **Hot**, **Cold**, dan **Warm**. Data dalam bentuk seperti ini tidak bisa langsung diproses oleh mesin untuk membuat model *machine learning*. Untuk mengatasi hal tersebut, kita perlu melakukan *processing* tambahan, yaitu *one-hot encoding*.

One-hot encoding merupakan teknik yang digunakan untuk membuat data kategorikal ke dalam representasi yang dapat dimengerti oleh model *machine learning*. *One-hot encoding* bekerja dengan cara berikut:

- Mengidentifikasi nilai unik yang terdapat pada kolom kategorik.
- Membuat kolom baru (*dummy variable*) untuk setiap nilai unik.
- Memasukkan nilai biner (1 atau 0) ke setiap kolom baru tersebut.

Perhatikan kembali gambar di atas. Awalnya terdapat tiga variasi nilai: **Hot**, **Cold**, dan **Warm**. Selanjutnya, dibuat tiga kolom baru untuk setiap variasi nilai: **var_hot**, **var_cold**, dan **var_warm**. Setelah itu, nilai 1 dan 0 dimasukkan ke setiap kolom berdasarkan kolom aslinya.

- Jika **Temperature** awalnya adalah **Hot**, maka **var_hot** = 1, **var_cold** = 0, dan **var_warm** = 0.
- Jika **Temperature** awalnya adalah **Cold**, maka **var_hot** = 0, **var_cold** = 1, dan **var_warm** = 0.
- Jika **Temperature** awalnya adalah **Warm**, maka **var_hot** = 0, **var_cold** = 0, dan **var_warm** = 1.

Untuk melakukan *one-hot encoding*, kita dapat memanfaatkan fungsi `pd.get_dummies()` dari `pandas`.

```
[11]: # contoh implementasi pd.get_dummies()
```

```
example_dummy = pd.DataFrame({
    "Daerah" : ['Jakarta', 'Bandung', 'Depok', 'Bekasi', 'Jakarta'],
    "Suhu" : [38, 36, 37.5, 38, 39],
})

example_dummy
```

```
[11]:   Daerah    Suhu
0  Jakarta  38.00000
1  Bandung  36.00000
2   Depok  37.50000
3  Bekasi  38.00000
4  Jakarta  39.00000
```

```
[12]: pd.get_dummies(example_dummy, dtype = int)
```

```
[12]:   Suhu  Daerah_Bandung  Daerah_Bekasi  Daerah_Depok  Daerah_Jakarta
0  38.00000             0              0             0              1
1  36.00000             1              0             0              0
2  37.50000             0              0             1              0
3  38.00000             0              1             0              0
```

```
4 39.00000      0      0      0      1
```

Detail lebih lanjut bagaimana menggunakan fungsi `pd.get_dummies()` dapat dilihat pada [referensi berikut](#).

Sekarang, kita akan melihat kembali bagaimana data `telco_clean` kita sebelumnya.

```
[13]: credit.head()
```

```
[13]:  fraud_bool  income  name_email_similarity  prev_address_months_count
current_address_months_count  customer_age  days_since_request
intended_balcon_amount  payment_type  zip_count_4w  velocity_6h  velocity_24h
velocity_4w  bank_branch_count_8w  date_of_birth_distinct_emails_4w
employment_status  credit_risk_score  email_is_free  housing_status
phone_home_valid  phone_mobile_valid  bank_months_count  has_other_cards
proposed_credit_limit  foreign_request  source  session_length_in_minutes
device_os  keep_alive_session  device_distinct_emails_8w  month
0      0  0.90000      0.61579      -1
51      40      0.01034      -0.69955      AB
2659  11007.70883  7448.79742  5576.30356      2030
7      CA      253      1      BA
1      1      9      1      1500.00000
0  INTERNET      13.07981  linux      0
1      1
1      0  0.60000      0.07246      -1
125      20      0.00697      -0.95947      AC
5599  6304.26284  4373.09359  4721.10002      1
13      CA      112      1      BE
1      0      -1      0      200.00000
0  INTERNET      6.66655  windows      0
1      4
2      1  0.50000      0.07659      -1
37      50      0.00632      -1.21993      AB
1352  1357.41355  6667.02966  6318.87626      15
6      CF      199      1      BC
1      1      28      0      1500.00000
0  INTERNET      4.49267  windows      1
2      0
3      0  0.30000      0.25543      23
19      20      0.00927      3.24224      AA
3743  5234.26230  6139.48958  4201.72592      1
14      CA      63      1      BC
0      1      16      0      200.00000
0  INTERNET      2.32189  other      1
1      3
4      0  0.90000      0.19236      -1
34      30      0.01924      -1.13156      AB
546  3589.15345  4574.39557  4318.25891      16
```


5		CA		275		1		BC	
1		1		1		1		1000.00000	
0	INTERNET			4.84515	linux			1	
1	5								

Berdasarkan deskripsi data sebelumnya, kita mengetahui bahwa kita memiliki beberapa kolom kategorikal.

Mari kita ubah kolom-kolom tersebut menggunakan `pd.get_dummies()`.

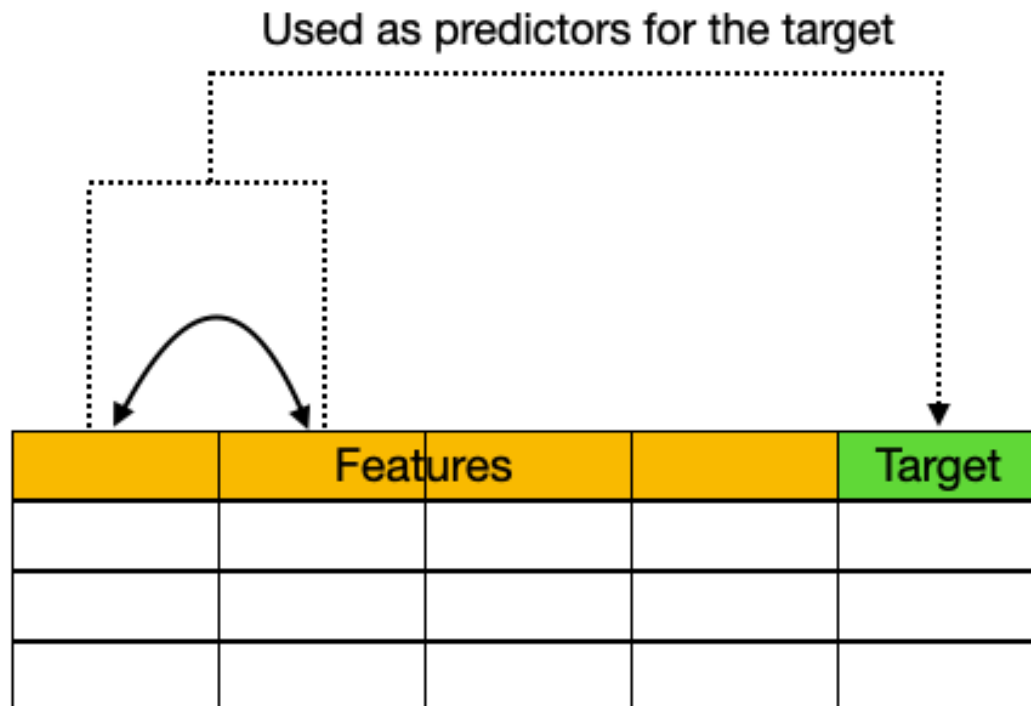
```
[14]: credit_enc = pd.get_dummies(credit,
                                columns = cat_cols,
                                drop_first = True,
                                dtype='int64')

credit_enc.head()
```

```
[14]: fraud_bool income name_email_similarity prev_address_months_count
current_address_months_count customer_age days_since_request
intended_balcon_amount zip_count_4w velocity_6h velocity_24h velocity_4w
bank_branch_count_8w date_of_birth_distinct_emails_4w credit_risk_score
email_is_free phone_home_valid phone_mobile_valid bank_months_count
has_other_cards proposed_credit_limit foreign_request
session_length_in_minutes keep_alive_session device_distinct_emails_8w month
payment_type_AB payment_type_AC payment_type_AD payment_type_AE
employment_status_CB employment_status_CC employment_status_CD
employment_status_CE employment_status_CF employment_status_CG
housing_status_BB housing_status_BC housing_status_BD housing_status_BE
housing_status_BF housing_status_BG source_TELEAPP device_os_macintosh
device_os_other device_os_windows device_os_x11
0 0 0.90000 0.61579 -1
51 40 0.01034 -0.69955 2659
11007.70883 7448.79742 5576.30356 2030
7 253 1 1 1
9 1 1500.00000 0
13.07981 0 1 1 1
0 0 0 0 0
0 0 0 0 0
0 0 0 0 0
0 0 0 0 0
0 0 0 0 0
1 0 0.60000 0.07246 -1
125 20 0.00697 -0.95947 5599
6304.26284 4373.09359 4721.10002 1
13 112 1 1 0
-1 0 200.00000 0
6.66655 0 1 4 0
1 0 0 0 0
```

0		0		0		0		0
0		0		0		1		0
0		0		0		0		1
0								
2	1	0.50000		0.07659				-1
37	50		0.00632		-1.21993		1352	
1357.41355	6667.02966		6318.87626		15			
6	199		1		1		1	
28	0		1500.00000		0			
4.49267		1			2	0		1
0	0		0		0			0
0		0		1			0	
0	1		0		0			0
0	0		0		0		1	
0								
3	0	0.30000		0.25543			23	
19	20		0.00927		3.24224		3743	
5234.26230	6139.48958		4201.72592		1			
14	63		1		0		1	
16	0		200.00000		0			
2.32189		1			1	3		0
0	0		0		0			0
0		0		0			0	
0	1		0		0			0
0	0		0		1		0	
0								
4	0	0.90000		0.19236			-1	
34	30		0.01924		-1.13156		546	
3589.15345	4574.39557		4318.25891		16			
5	275		1		1		1	
1	1		1000.00000		0			
4.84515		1			1	5		1
0	0		0		0			0
0		0		0			0	
0	1		0		0			0
0	0		0		0		0	
0								

4.2 Target & Predictors Splitting



Di dunia *machine learning*, variabel yang kita gunakan dalam proses pemodelan terbagi menjadi dua:

1. Variabel Prediktor/Variabel Independen (X)

Variabel prediktor merupakan variabel-variabel yang digunakan untuk membuat prediksi. Hasil prediksi *machine learning* sangat bergantung pada nilai dari variabel-variabel prediktor. Variabel prediktor dapat berupa variabel numerikal maupun variabel kategorikal. Umumnya, variabel prediktor dinotasikan dengan X . Seperti yang terlihat pada gambar, variabel prediktor ditunjukkan oleh kolom-kolom yang berwarna oranye.

2. Variabel Target/Variabel Dependen (y)

Variabel target merupakan variabel yang akan diprediksi menggunakan prediktor. Seperti namanya, hasil prediksi dependen terhadap variabel prediktor. Pada kasus klasifikasi, variabel target merupakan variabel kategorikal. Berbeda dengan *case* regresi yang variabel targetnya merupakan variabel numerikal. Variabel target dinotasikan sebagai y . Pada gambar, variabel target ditunjukkan oleh kolom yang berwarna hijau.

Untuk memisahkan variabel prediktor dan variabel target, kita bisa memanfaatkan fungsi `.drop()`. Berdasarkan *problem statement* sebelumnya, variabel target kita adalah kolom `fraud_bool` dan sisanya merupakan variabel prediktor.

```
[15]: # mendefinisikan variabel target
      y = credit_enc['fraud_bool']

      # mendefinisikan variabel prediktor
```

```
X = credit_enc.drop(columns = ['fraud_bool'])
```

```
[16]: # melihat 5 baris pertama X
X.head()
```

```
[16]: income name_email_similarity prev_address_months_count
current_address_months_count customer_age days_since_request
intended_balcon_amount zip_count_4w velocity_6h velocity_24h velocity_4w
bank_branch_count_8w date_of_birth_distinct_emails_4w credit_risk_score
email_is_free phone_home_valid phone_mobile_valid bank_months_count
has_other_cards proposed_credit_limit foreign_request
session_length_in_minutes keep_alive_session device_distinct_emails_8w month
payment_type_AB payment_type_AC payment_type_AD payment_type_AE
employment_status_CB employment_status_CC employment_status_CD
employment_status_CE employment_status_CF employment_status_CG
housing_status_BB housing_status_BC housing_status_BD housing_status_BE
housing_status_BF housing_status_BG source_TELEAPP device_os_macintosh
device_os_other device_os_windows device_os_x11
0 0.90000 0.61579 -1
51 40 0.01034 -0.69955 2659
11007.70883 7448.79742 5576.30356 2030
7 253 1 1 1
9 1 1500.00000 0
13.07981 0 1 1 1
0 0 0 0 0
0 0 0 0 0
0 0 0 0 0
0 0 0 0 0
1 0.60000 0.07246 -1
125 20 0.00697 -0.95947 5599
6304.26284 4373.09359 4721.10002 1
13 112 1 1 0
-1 0 200.00000 0
6.66655 0 1 4 0
1 0 0 0 0
0 0 0 0 0
0 0 1 1 0
0 0 0 1
0 0.50000 0.07659 -1
37 50 0.00632 -1.21993 1352
1357.41355 6667.02966 6318.87626 15
6 199 1 1 1
28 0 1500.00000 0
4.49267 1 2 0 1
0 0 0 0 0
```

```

0          0          1          0          0
0          1          0          0          0
0          0          0          0          1
0
3 0.30000          0.25543          23
19          20          0.00927          3.24224          3743
5234.26230        6139.48958        4201.72592          1
14          63          1          0          1
16          0          200.00000          0
2.32189          1          1          3          0
0          0          0          0          0
0          0          0          0          0
0          1          0          0          0
0          0          0          1          0
0
4 0.90000          0.19236          -1
34          30          0.01924          -1.13156          546
3589.15345        4574.39557        4318.25891          16
5          275          1          1          1
1          1          1000.00000          0
4.84515          1          1          5          1
0          0          0          0          0
0          0          0          0          0
0          1          0          0          0
0          0          0          0          0
0

```

```
[17]: # melihat 5 baris pertama y
```

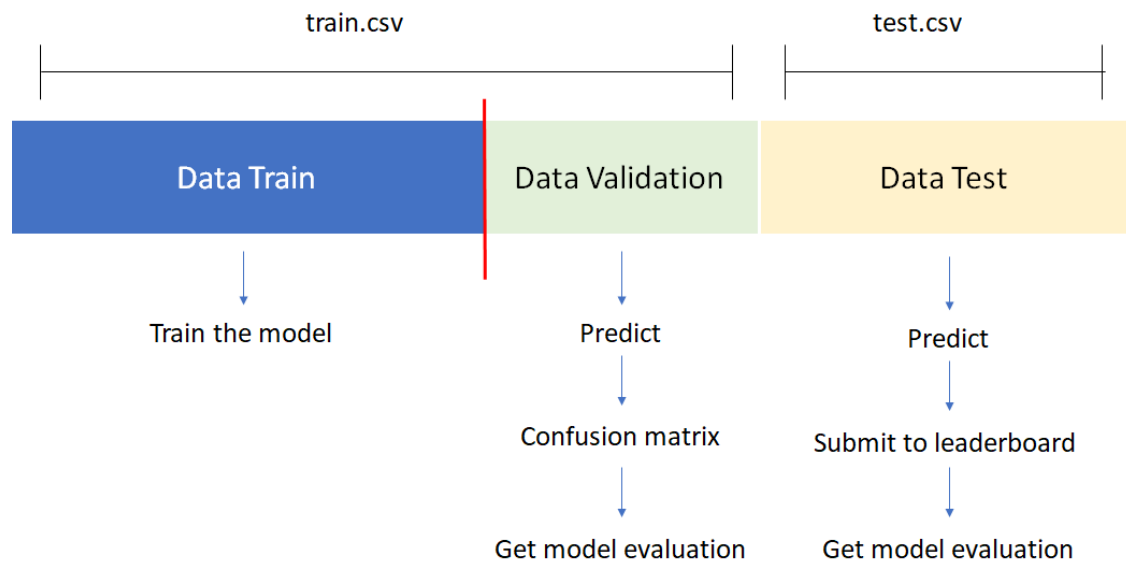
```
y.head()
```

```

[17]: 0    0
      1    0
      2    1
      3    0
      4    0
      Name: fraud_bool, dtype: int64

```

4.3 Cross Validation



Sebelum membuat model *machine learning* untuk klasifikasi, kita akan berkenalan dengan konsep *cross validation* di mana kita:

1. Memisahkan *dataset* kita menjadi *training set*, *validation set*, dan *testing set*.
2. Melatih model kita hanya menggunakan *training set*.
3. Mengevaluasi model pada *validation set* dan kembali ke langkah sebelumnya jika perlu (misalnya, memilih variabel prediktor yang berbeda, menggunakan parameter yang berbeda, atau menyetel aspek lain dari spesifikasi model).
4. Memilih model final berdasarkan kriteria evaluasi, misalnya berdasarkan akurasi.
5. Mendapatkan hasil prediksi yang tidak bias saat melakukan prediksi pada *testing set*.

Kita dapat mengulangi langkah 2 dan langkah 3 sebanyak yang diperlukan. Misalnya, kita ingin menguji berbagai metode klasifikasi, mengganti parameter model, atau kombinasi variabel prediktor yang berbeda. Dari hasil percobaan tersebut, kita memilih model terbaik berdasarkan suatu kriteria evaluasi. Model terbaik ini yang akan digunakan untuk memprediksi data pada *testing set*. Perlu diperhatikan bahwa *testing set* tidak boleh digunakan selama proses pelatihan model. Dalam hal ini, *testing set* merupakan data baru yang tidak ditemui model selama proses pelatihan.

4.3.1 Train-Test Split

Untuk mengimplementasikan *cross validation*, kita dapat menggunakan fungsi `train_test_split()` dari *package sklearn*.

Terdapat banyak praktik terkait berapa persentase dari data yang dialokasikan untuk *training set* dan *testing set*. Beberapa praktik yang cukup umum adalah 80% untuk *training set* dan 20% untuk *testing set*. Praktik lainnya menggunakan 75% dari data untuk *training set* dan 25% untuk *testing set*.

Note: perlu diingat bahwa proporsi untuk *training set* selalu lebih besar.

```
[18]: # implementasi train_test_split dengan test_size sebesar 20% dari keseluruhan data
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.20, random_state=42, stratify=y)
```

```
[19]: # shape data awal
```

```
X.shape, y.shape
```

```
[19]: ((100000, 46), (100000,))
```

```
[20]: # shape data untuk training set
```

```
X_train.shape, y_train.shape
```

```
[20]: ((80000, 46), (80000,))
```

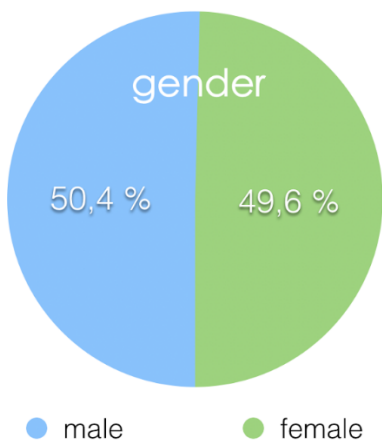
```
[21]: # shape data untuk testing set
```

```
X_test.shape, y_test.shape
```

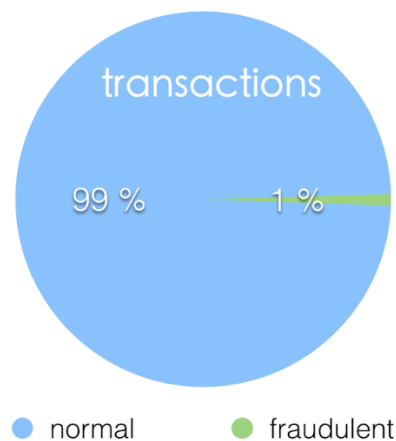
```
[21]: ((20000, 46), (20000,))
```

4.3.2 Class Balance & Imbalance

Balanced Dataset



Unbalanced Dataset



Imbalance merupakan suatu kondisi di mana terdapat satu kelas yang proporsinya lebih banyak (mayoritas) dibandingkan kelas lainnya. *Imbalance* sangat sering ditemui di dunia nyata, termasuk pada *dataset* yang kita gunakan saat ini.

Sekarang, mari kita lihat bagaimana proporsi antara label 0 dan 1 pada `y_train` menggunakan fungsi `.value_counts()`.

```
[22]: y_train.value_counts()
```

```
[22]: fraud_bool
0      71177
1       8823
Name: count, dtype: int64
```

Terlihat bahwa kelas 0, yakni pembeli yang tidak membeli produk lebih banyak dari pada kelas 1, yakni pembeli yang membeli produk. Kasus *imbalance* ini perlu ditangani dengan benar untuk mencegah model bias ke salah satu kelas. Dalam hal ini, model lebih banyak mempelajari contoh kasus dari kelas 0 dan tidak dapat mengidentifikasi karakteristik dari kelas 1 dengan benar.

Teknik yang bisa dimanfaatkan untuk mengatasi *imbalance dataset* adalah *oversampling* (*upsampling*) dan *undersampling* (*downsampling*). Referensi terkait metode *handling imbalance dataset* dapat dilihat [di sini](#)

5 Model Classification 1 - Logistic Regression

Sejauh ini, kita sudah melakukan persiapan untuk membuat model *machine learning* yang bisa melakukan klasifikasi. Mulai dari akses data yang akan digunakan sampai melakukan *sampling* untuk mengatasi data yang *imbalance*. Model pertama yang akan kita gunakan untuk melakukan klasifikasi adalah *logistic regression*.

Logistic regression merupakan metode klasifikasi yang menjadi pondasi untuk menentukan hasil prediksi y .

- Apabila y merupakan satu nilai di antara dua pilihan kategorikal (biner), *logistic regression* dapat juga disebut sebagai *binomial logistic regression*.
- Apabila y merupakan satu nilai di antara banyak pilihan kategorikal, *logistic regression* dapat juga disebut sebagai *multinomial logistic regression*.

Logistic regression menggunakan konsep *probability*/peluang untuk mengeluarkan prediksi. Oleh karena itu, kita akan coba untuk meninjau kembali konsep peluang.

5.1 Probability

Secara sederhana, *probability*/peluang dapat dikatakan sebagai besarnya kemungkinan sesuatu akan terjadi. Apabila didefinisikan secara matematis, peluang p suatu kejadian dapat dirumuskan:

$$p = \frac{\text{frekuensi kejadian yang diharapkan}}{\text{frekuensi seluruh percobaan}}$$

Contoh:

- Jika terdapat 20 dari 100 pelanggan perusahaan telekomunikasi yang *churn* (hilang), maka peluang $\text{churn} = \frac{20}{100} = 0.2$.
- Jika 3 dari 5 *email* terdeteksi sebagai email yang *spam*, maka peluang $\text{spam} = \frac{3}{5} = 0.6$.
- Jika 4 dari 5 penerbangan dilakukan tepat waktu, maka peluang penerbangan dilakukan tepat waktu $= \frac{4}{5} = 0.8$.

Berikut merupakan beberapa poin penting yang perlu diingat mengenai peluang:

- *Range* peluang adalah 0 sampai 1. 0 artinya kejadian tersebut mustahil terjadi dan 1 artinya kejadian tersebut pasti terjadi.
- Bila p merupakan peluang suatu peristiwa untuk terjadi, maka q adalah peluang di mana peristiwa tidak terjadi (peluang komplement). Secara matematis:

$$p + q = 1 \rightarrow q = 1 - p$$

5.2 Relating Probabilities to Odds

Selama ini, kita hanya mengenal p untuk menyatakan suatu kemungkinan/peluang. Sekarang, kita akan mendalami konsep lain yang juga digunakan untuk membahasakan peluang, yakni *odds*.

Odds secara sederhana merupakan rasio peluang suatu peristiwa terjadi (p) dan peluang suatu peristiwa tidak terjadi/peluang komplement peristiwa tersebut (q).

$$Odds = \frac{p}{q} = \frac{p}{1 - p}$$

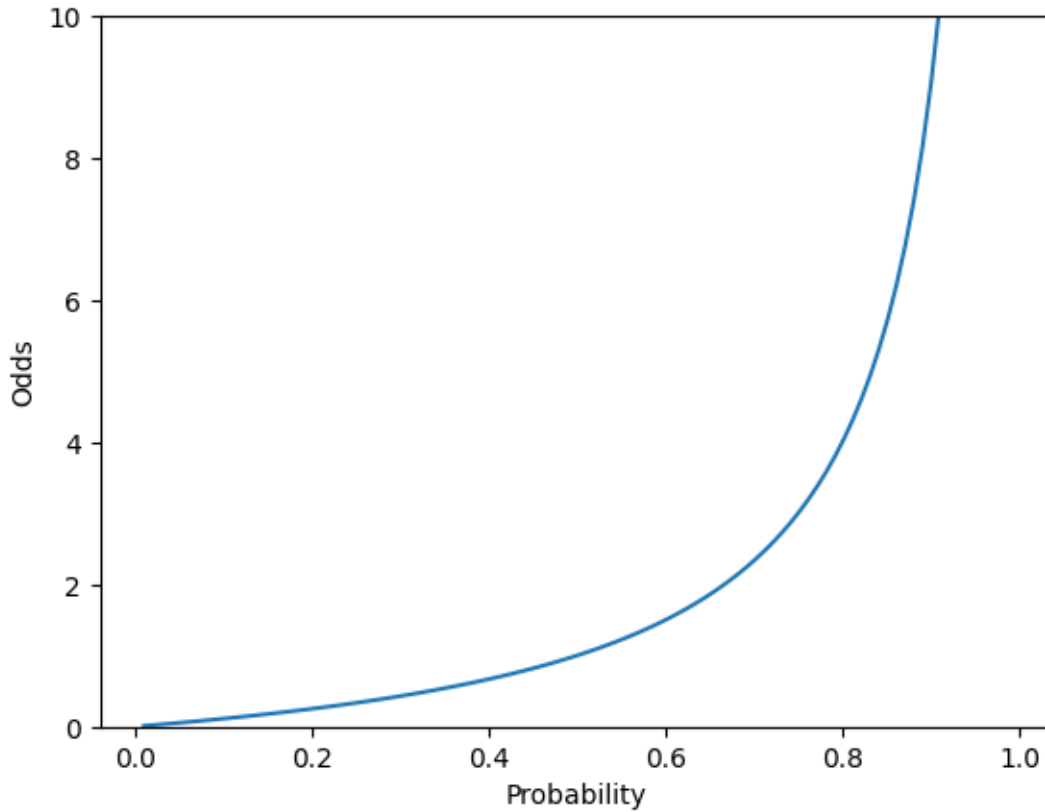
Apabila kita menggunakan kembali contoh pada bagian *probability*:

- *Odds* pelanggan yang *churn* dari perusahaan telekomunikasi = $\frac{0.2}{0.8} = 0.25$. Artinya, peluang pelanggan untuk kembali/berlangganan kembali 4 kali lebih besar dibandingkan dengan peluang pelanggan yang hilang.
- *Odds email* yang *spam* = $\frac{0.6}{0.4} = 1.5$. Artinya, peluang suatu email untuk *spam* 1.5 kali lebih tinggi daripada *email* tersebut tidak *spam*.
- *Odds* penerbangan tepat waktu = $\frac{0.8}{0.2} = 4$. Artinya, suatu penerbangan 4 kali lebih berkeungkinan untuk berangkat tepat waktu.

Apabila kita membuat suatu plot yang menunjukkan hubungan peluang p dan nilai *odds*-nya:

```
[23]: x = np.linspace(0.01, 0.99, 100)
      y = x/(1-x)

      plt.plot(x,y);
      plt.ylabel('Odds');
      plt.xlabel('Probability');
      plt.ylim(0,10);
```



Terlihat bahwa hubungan antara peluang p dan *odds*-nya adalah monotonik. Semakin besar nilai peluang p , semakin besar nilai *odds*.

Range nilai *odds* adalah 0 sampai $+\text{Inf}$

5.2.1 Log of Odds

Sebelumnya kita sudah mengenal *odds* sebagai perbandingan antara peluang p dan komplemen q . *Log of odds* tidak lain merupakan *odds* yang ditransformasi dengan fungsi `log()`. Untuk transformasi ini, kita bisa memanfaatkan fungsi `np.log()` dari *package* `numpy`.

$$\log(\text{odds}) = \log\left(\frac{p}{q}\right) = \log\left(\frac{p}{1-p}\right) = \text{logit}(p)$$

Misalnya, kita memiliki peluang $p = 0.2$. Untuk mendapatkan *log of odds*-nya:

[24]: `# menentukan log of odds untuk peluang p = 0.2`

```
p = 0.2
odds = p/(1-p)
log_odds = np.log(odds)
```

```
print(odds)
print(log_odds)
```

0.25
-1.3862943611198906

Untuk $p = 0.2$ didapatkan nilai *log of odds*-nya adalah -1.3862943611198906.

Range nilai *log of odds* adalah $-\text{Inf}$ sampai $+\text{Inf}$.

Apabila kita ingin mendapatkan kembali nilai *odds* dari nilai *log of odds* tersebut, kita bisa menggunakan fungsi `np.exp()` dari *package* `numpy`.

```
[25]: # mendapatkan nilai odds dari nilai log of odds = -1.3862943611198906

np.exp(log_odds)
```

[25]: 0.25

Untuk mendapatkan nilai peluang dari *log of odds*:

```
[26]: # mendapatkan nilai p dari log of odds

1/(1 + np.exp(-log_odds))
```

[26]: 0.2

[Key Points]

- Proses transformasi: peluang - odds - *log of odds*.
- Mengubah peluang menjadi odds:

$$\frac{p}{1-p}$$

- Mengubah odds menjadi *log of odds*:

$$\log(odds)$$

- Mengubah *log of odds* menjadi odds:

$$\exp(\log(odds))$$

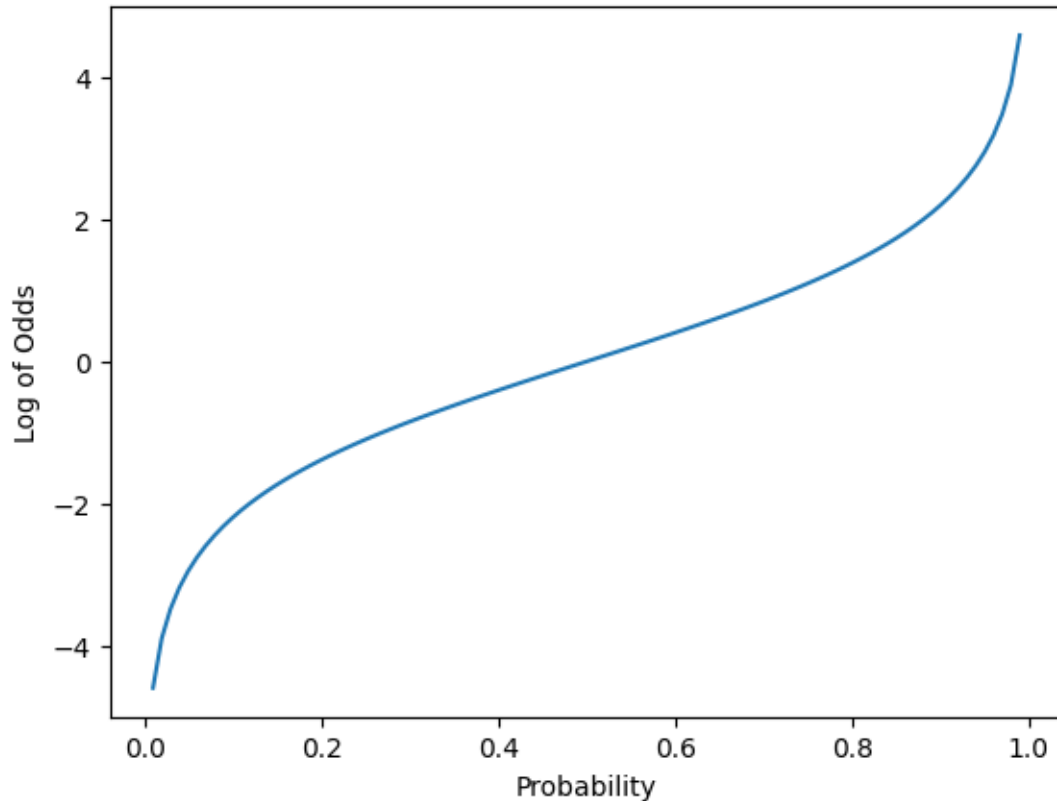
- Mengubah *log of odds* menjadi peluang :

$$\frac{1}{(1 + e^{(-\log of odds)})}$$

Berikut merupakan plot antara peluang p dengan nilai *log of odds*-nya.

```
[27]: x = np.linspace(0.01, 0.99, 100)
odds = x/(1-x)
y = np.log(odds)
```

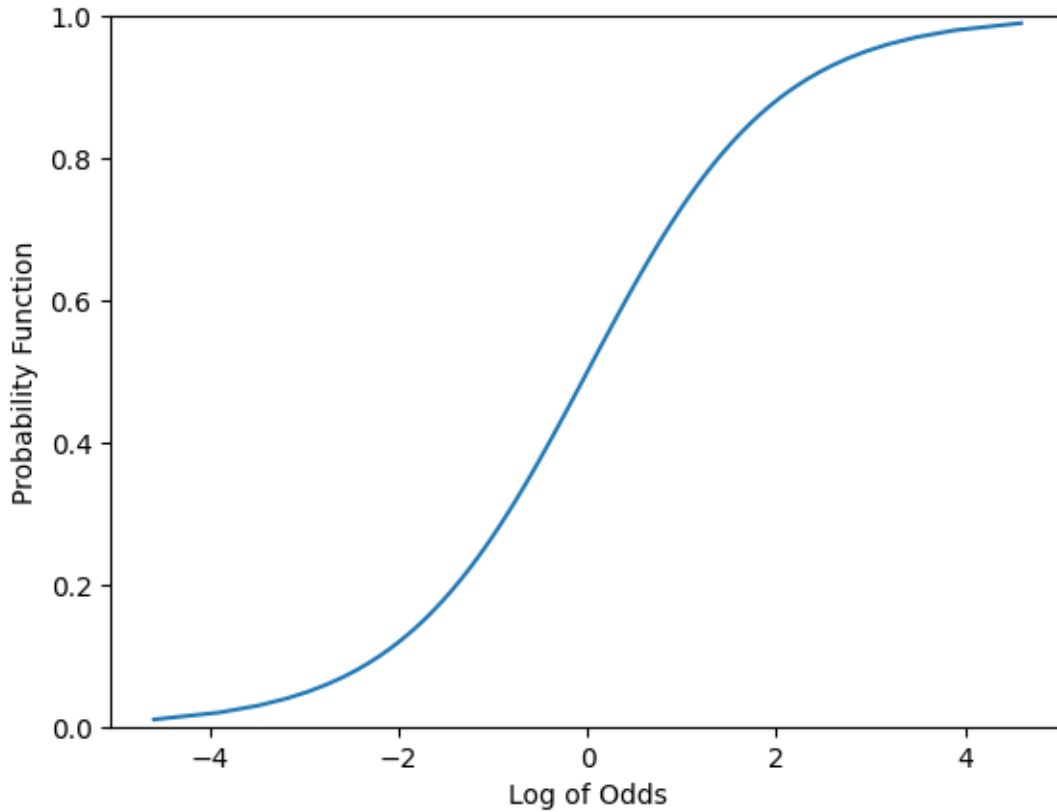
```
plt.plot(x,y);
plt.ylabel('Log of Odds');
plt.xlabel('Probability');
plt.ylim(-5,5);
```



Pada plot di atas, kita melihat peluang p pada sumbu x dan *log of odds* pada sumbu y . Sekarang, kita lebih tertarik dengan nilai peluang. Bagaimana jika kita tukar posisi keduanya sehingga *log of odds* terletak pada sumbu x dan peluang p pada sumbu y ?

```
[28]: y = np.linspace(0.01, 0.99, 100)
      odds = y/(1-y)
      x = np.log(odds)

      plt.plot(x,y);
      plt.ylabel('Probability Function');
      plt.xlabel('Log of Odds');
      plt.ylim(0,1);
```



Ternyata kita mendapatkan grafik berbentuk huruf S. Plot di atas tidak lain merupakan plot dari fungsi sigmoid yang digunakan oleh model *logistic regression* untuk memprediksi peluang.

$$\text{Sigmoid}(x) = \frac{1}{1 + e^{-x}}$$

Peluang inilah yang nantinya akan menjadi patokan apakah data kita termasuk dalam kelas 0 atau 1. Misalnya, dengan menetapkan *threshold* 0.5, apabila kita memiliki prediksi peluang sebesar 0.8, maka kita mengategorikan bahwa data termasuk ke dalam kelas 1.

5.3 Logistic Regression from First Principles

Meskipun namanya mengandung kata “*regression*”, *logistic regression* tidak ditujukan untuk *case* regresi. *Nature* dari *logistic regression* yang melibatkan konsep peluang membuatnya lebih cocok untuk *case* klasifikasi. Apabila ditinjau lebih lanjut, *logistic regression* dapat dikatakan sebagai versi spesial dari regresi linear.

Recall kembali bahwa pada regresi linear, kita memprediksi variabel target yang kontinu. Hubungan antara prediktor x dan target y dapat dituliskan sebagai berikut.

$$y = \beta_0 + \beta_1 \cdot x_1 + \beta_2 \cdot x_2 + \dots + \beta_n \cdot x_n$$

Keterangan:

- n = banyak prediktor
- β_0 = *intercept*. *Intercept* menunjukkan nilai y saat semua prediktornya 0.
- $\beta_1, \beta_2, \dots, \beta_n$ = koefisien prediktor. Koefisien prediktor menunjukkan besarnya kenaikan variabel target saat variabel prediktor naik sebesar 1 satuan dengan catatan nilai semua prediktor lainnya tetap.

Dengan formulasi di atas, maka variabel target y untuk *case* regresi dapat memiliki nilai pada *range* $-\text{Inf}$ sampai $+\text{Inf}$.

Apabila kita menggunakan formulasi yang sama untuk memprediksi peluang pada *case* klasifikasi, maka tidak akan sesuai. Ingat kembali bahwa peluang hanya memiliki *range* dari 0 sampai 1. Untuk menyiasati hal tersebut, kita perlu menggunakan “representasi lain/transformatasi” dari peluang di mana “representasi lain/transformatasi” tersebut memiliki nilai pada *range* $-\text{Inf}$ sampai $+\text{Inf}$.

Masih ingat dengan *log of odds* yang kita bahas pada bagian sebelumnya? Kita akan menggunakan *log of odds* sebagai “representasi lain/transformatasi” dari peluang. Dengan demikian, untuk *case* klasifikasi menggunakan *logistic regression*, formulasi sebelumnya dapat ditulis ulang sebagai berikut:

$$\log\left(\frac{p}{1-p}\right) = \text{logit}(p) = \beta_0 + \beta_1 \cdot x_1 + \beta_2 \cdot x_2 + \dots + \beta_n \cdot x_n$$

Untuk mengubah *log of odds* menjadi peluang yang diinginkan, kita menggunakan fungsi sigmoid yang plot grafiknya sempat kita gambarkan pada bagian sebelumnya.

$$p(x) = \text{Sigmoid}(\beta_0 + \beta_1 \cdot x_1 + \dots + \beta_n \cdot x_n) = \frac{1}{1 + e^{-(\beta_0 + \beta_1 \cdot x_1 + \dots + \beta_n \cdot x_n)}}$$

[Key Points]

- Pada *logistic regression*, kita memprediksi *log of odds* sebagai bentuk lain dari peluang.
- Fungsi sigmoid digunakan untuk *mapping log of odds* menjadi peluang.

5.4 Logistic Regression in Action

Pada bagian sebelumnya, kita telah mengetahui bahwa:

- *Logistic regression* mengeluarkan prediksi dalam bentuk *log of odds*.
- Untuk mengubah *log of odds* menjadi peluang, digunakan fungsi sigmoid.

Pada bagian ini, kita akan melihat bagaimana implementasi *logistic regression* pada Python menggunakan *package stats.model*. Untuk keperluan ini, kita coba untuk menggabungkan kembali X_{train} dan y_{train} yang sudah kita pisahkan sebelumnya.

```
[29]: # menggabungkan variabel prediktor dan variabel target dalam satu tabel/  
      ↪ dataframe  
  
train_data = X_train.copy()  
train_data['fraud_bool'] = y_train
```

```
train_data = train_data.reset_index().drop(columns = "index")
```

```
[30]: train_data.head()
```

```
[30]:  income  name_email_similarity  prev_address_months_count
current_address_months_count  customer_age  days_since_request
intended_balcon_amount  zip_count_4w  velocity_6h  velocity_24h  velocity_4w
bank_branch_count_8w  date_of_birth_distinct_emails_4w  credit_risk_score
email_is_free  phone_home_valid  phone_mobile_valid  bank_months_count
has_other_cards  proposed_credit_limit  foreign_request
session_length_in_minutes  keep_alive_session  device_distinct_emails_8w  month
payment_type_AB  payment_type_AC  payment_type_AD  payment_type_AE
employment_status_CB  employment_status_CC  employment_status_CD
employment_status_CE  employment_status_CF  employment_status_CG
housing_status_BB  housing_status_BC  housing_status_BD  housing_status_BE
housing_status_BF  housing_status_BG  source_TELEAPP  device_os_macintosh
device_os_other  device_os_windows  device_os_x11  fraud_bool
0 0.50000          0.58414          -1
354          60          0.02091          22.64003          1342
1097.47353  4135.31093  5993.67415          547
4          142          1          1          1
28          1          200.00000          0
6.25370          0          1          0          0
0          0          0          0          0          1
0          0          0          0          0
0          0          0          0          0          0
0          0          0          0          0          0
0          0          0          0          0          0
0          0          0          0          0          0
1 0.10000          0.88506          -1
156          20          0.00458          14.54435          597
4891.02002  3214.06166  3144.47120          1
6          223          1          0          1
28          0          990.00000          0
8.95264          1          1          7          0
0          0          0          0          0          0
0          0          0          1          0
1          0          0          0          0          0
0          0          0          1          0
0          0          0          1          0
2 0.10000          0.69251          -1
37          20          0.01531          -0.76549          2120
2480.21932  6969.44880  3710.56637          0
7          128          1          1          1
-1          0          200.00000          1
9.67542          1          1          6          0
1          0          0          0          0          0
0          0          0          0          0          0
```

```

0          1          0          0          0
0          0          0          1          0
0          0          0          0          0
3 0.90000          0.58853          61
12          40          0.00384          15.55668          268
4593.44153          3050.17974          4256.22243          46
11          108          1          1          1
21          1          500.00000          0
8.50664          0          0          1          6          0
0          0          0          0          0          0
0          0          0          1          0          0
1          0          0          0          0          0
0          0          0          0          0          0
0          0          0          0          0          0
4 0.30000          0.81677          -1
209          20          0.00454          -0.74999          1629
1620.07894          2234.76245          3111.30845          12
6          130          1          0          1
1          0          200.00000          0          0
6.34498          0          0          1          7          1
0          0          0          0          0          0
0          0          0          0          0          0
0          0          0          1          0
0          0          0          0          1
0          1

```

5.4.1 Logistic Regression with One Categorical (Discrete) Predictor

Untuk memahami bagaimana *logistic regression* bekerja, kita akan coba untuk menggunakan satu prediktor kategorikal untuk memprediksi variabel target kita. Untuk pembelajaran pada kasus ini, kita akan menggunakan:

- Prediktor: kolom `has_other_cards`.
- Target: kolom `fraud_bool`.

Dengan satu prediktor, yaitu kolom `has_other_cards`, maka formula untuk prediksi kita dapat ditulis:

$$\text{logit}(p) = \beta_0 + \beta_1 \cdot \text{has_other_cards}$$

Kita akan menggunakan fungsi `sm.Logit()` dari *package statsmodel* untuk melakukan *logistic regression*.

```
[31]: train_data['intercept'] = 1

logit_model_tr = sm.Logit(train_data['fraud_bool'],
    ↪train_data[['intercept', 'has_other_cards']]).fit()
logit_model_tr.summary()
```


Optimization terminated successfully.
 Current function value: 0.340175
 Iterations 7

[31]:

Dep. Variable:	fraud_bool	No. Observations:	80000
Model:	Logit	Df Residuals:	79998
Method:	MLE	Df Model:	1
Date:	Sun, 24 Dec 2023	Pseudo R-squ.:	0.02000
Time:	14:49:36	Log-Likelihood:	-27214.
converged:	True	LL-Null:	-27769.
Covariance Type:	nonrobust	LLR p-value:	1.672e-243

	coef	std err	z	P> z	[0.025	0.975]
intercept	-1.9227	0.012	-161.481	0.000	-1.946	-1.899
has_other_cards	-1.1462	0.039	-29.021	0.000	-1.224	-1.069

Dengan hasil di atas, formula prediksi kita dapat ditulis ulang:

$$\text{logit}(p) = -1.9227 - 1.1462 \cdot \text{has_other_cards}$$

Mari kita bahas arti dari setiap koefisien yang telah kita dapatkan.

Intercept (β_0)

Intercept menunjukkan besarnya *log of odds* saat `has_other_cards = 0`.

Pada kasus kita maka *log of odds* yang didapatkan adalah -1.9227.

Untuk mendapatkan *odds* dan peluang dari *log of odds*:

[32]: `y_train.value_counts()`

```
[32]: fraud_bool
0      71177
1       8823
Name: count, dtype: int64
```

```
[33]: # mendapatkan odds dan peluang dari log of odds = -1.7615

log_odds = -1.9227
odds = np.exp(log_odds)
p = 1 / (1 + np.exp(-log_odds)) # fungsi sigmoid

print(f'Log of odds akun bank fraud = {round(log_odds, 3)}.')
print(f'Odds akun bank fraud ketika tidak memiliki kartu lain = {round(odds, 3)}.')
print(f'Peluang akun bank fraud ketika tidak memiliki kartu lain = {round(p, 3)}.')
```

Log of odds akun bank fraud = -1.923.

Odds akun bank fraud ketika tidak memiliki kartu lain = 0.146.

Peluang akun bank fraud ketika tidak memiliki kartu lain = 0.128.

```
[34]: # crosscheck
pd.crosstab(train_data['fraud_bool'], train_data['has_other_cards'])
```

```
[34]: has_other_cards      0      1
fraud_bool
0          55297  15880
1          8085   738
```

```
[35]: # Peluang akun bank fraud ketika tidak memiliki kartu lain

round(8085/(55297 + 8085), 3)
```

```
[35]: 0.128
```

Saat memiliki kartu lain (`has_other_cards = 0`), peluang akun tersebut Response = 1 adalah sebesar 12.8%.

Koefisien `has_other_cards` (β_1)

Koefisien `has_other_cards` menunjukkan *log* dari perbandingan *odds* akun merupakan fraud ketika tidak memiliki kartu lain dan ketika memiliki kartu lain.

$$Koe\text{fisien}_{has_other_cards} = \log\left(\frac{odd(has_other_cards = 1, fraud_bool = 1)}{odd(has_other_cards = 0, fraud_bool = 1)}\right)$$

Pada kasus kita, *log* perbandingan *odds* saat memiliki kartu lain dan saat tidak tidak memiliki kartu lain adalah -1.1462.

Dengan *log* perbandingan sebesar -1.1462, maka rasio *odds* yang sebenarnya adalah:

```
[36]: log_ratio = -1.1462
ratio = np.exp(log_ratio)

print(f'Perbandingan odds akun bank fraud jika akun memiliki kartu lain yaitu =_
↳ {round(ratio, 3)}.')
```

Perbandingan odds akun bank fraud jika akun memiliki kartu lain yaitu = 0.318.

Berdasarkan perbandingan *odds*-nya, *odds* untuk sebuah akun jika memiliki kartu lain adalah (`has_other_cards = 1`) 31% lebih rendah daripada odds ketika akun tidak memiliki kartu lain (`has_other_cards = 0`).

Jika kita melihat kembali proporsi antara kolom `fraud_bool` dan `has_other_cards`:

```
[37]: # crosscheck: manual calculation

pd.crosstab(train_data['fraud_bool'], train_data['has_other_cards'])
```

```
[37]: has_other_cards      0      1
      fraud_bool
0                55297  15880
1                8085   738
```

- Untuk `has_other_cards = 0`
 - Peluang `fraud_bool = 0` = $\frac{55297}{(55297+8085)}$
 - Peluang `fraud_bool = 1` = $\frac{8085}{(55297+8085)}$
- Untuk `has_other_cards = 1`
 - Peluang `fraud_bool = 0` = $\frac{15880}{(15880+738)}$
 - Peluang `fraud_bool = 1` = $\frac{738}{(15880+738)}$
- Perhitungan *odds*
 - *Odds* `has_other_cards = 0` mendapatkan `fraud_bool = 1`

$$\frac{\frac{8085}{(55297+8085)}}{\frac{55297}{(55297+8085)}} = \frac{8085}{55297}$$

- *Odds* `has_other_cards = 1` mendapatkan `fraud_bool = 1`

$$\frac{\frac{738}{(15880+738)}}{\frac{15880}{(15880+738)}} = \frac{738}{15880}$$

- Perbandingan *odds*

$$\frac{\frac{738}{15880}}{\frac{8085}{55297}} = -1.146$$

```
[38]: # koefisien dari Treatment

odd_fraud1_card0 = (8085/55297)
odd_fraud1_card1 = (738/15880)

ratio_card0and1 = odd_fraud1_card0/odd_fraud1_card1

print(f'Rasio odds: {round(ratio_card0and1, 3)}')
print(f'Log rasio odds: {round(np.log(ratio_card0and1),3)}')
```

Rasio odds: 3.146

Log rasio odds: 1.146.

Kita berhasil memverifikasi bahwa koefisien dari `has_other_cards` merupakan *log* dari perbandingan *odds* apakah akun bank fraud berdasarkan status kepemilikan kartu lain akun tersebut.

5.4.2 Logistic Regression with One Numerical (Continuous) Predictor

Sekarang, kita akan mencoba untuk menggunakan satu prediktor numerikal, yaitu `credit_risk_score`. Variabel targetnya masih menggunakan kolom `fraud_bool`. Dengan demikian, fungsi untuk melakukan prediksi dapat ditulis:

$$\text{logit}(p) = \beta_0 + \beta_1 \cdot \text{credit_risk_score}$$

```
[39]: logit_model_crs = sm.Logit(train_data['fraud_bool'],
    ↪train_data[['intercept', 'credit_risk_score']]).fit()
logit_model_crs.summary()
```

Optimization terminated successfully.

Current function value: 0.326851

Iterations 7

[39]:

Dep. Variable:	fraud_bool	No. Observations:	80000
Model:	Logit	Df Residuals:	79998
Method:	MLE	Df Model:	1
Date:	Sun, 24 Dec 2023	Pseudo R-squ.:	0.05838
Time:	14:49:37	Log-Likelihood:	-26148.
converged:	True	LL-Null:	-27769.
Covariance Type:	nonrobust	LLR p-value:	0.000

	coef	std err	z	P> z	[0.025	0.975]
intercept	-3.4334	0.029	-118.024	0.000	-3.490	-3.376
credit_risk_score	0.0088	0.000	55.977	0.000	0.008	0.009

Berdasarkan hasil di atas, kita dapat menulis ulang fungsi prediksi kita menjadi:

$$\text{logit}(p) = -3.4334 + 0.0088 \cdot \text{credit_risk_score}$$

Intercept (β_0)

Intercept menunjukkan besarnya *log of odds* saat `credit_risk_score = 0`.

Pada kasus kita, saat `credit_risk_score = 0`, maka besar *log of odds* akun bank tersebut fraud adalah -3.4334.

Untuk mendapatkan *odds* dan peluang dari *log of odds*:

```
[40]: # mendapatkan odds dan peluang dari log of odds = -1.6828

log_odds = -3.4334
odds = np.exp(log_odds)
p = 1 / (1 + np.exp(-log_odds))

print(f'Log of odds akun bank fraud saat `credit_risk_score = 0` =
    ↪{round(log_odds, 3)}.')
print(f'Odds akun bank fraud saat `credit_risk_score = 0` = {round(odds, 3)}.')
print(f'Peluang akun bank fraud saat `credit_risk_score = 0` = {round(p, 3)}.')
```

Log of odds akun bank fraud saat ``credit_risk_score = 0`` = -3.433.

Odds akun bank fraud saat ``credit_risk_score = 0`` = 0.032.

Peluang akun bank fraud saat ``credit_risk_score = 0`` = 0.031.

Saat `credit_risk_score = 0`, peluang akun bank mendapatkan `fraud_bool = 1` adalah sebesar 3.1%.

Koefisien `credit_risk_score` (β_1)

Koefisien `credit_risk_score` menunjukkan kenaikan *log of odds* apabila nilai `credit_risk_score` naik sebesar 1 satuan.

Pada kasus kita, jika nilai `credit_risk_score` naik, maka nilai *log of odds* akan naik sebesar 0.0088.

Misalnya, nilai awal `credit_risk_score` = 50. Selanjutnya, nilai `credit_risk_score` berubah menjadi 51. Dengan demikian:

```
[41]: # crosscheck

# log of odds saat credit_risk_score = 50
l1 = -3.4334 + 0.0088 * 50

# log of odds saat credit_risk_score = 51
l2 = -3.4334 + 0.0088 * 51

# selisih log of odds
print(f'Selisih log of odds = {round(l2-l1, 4)}.')
```

Selisih log of odds = 0.0088.

5.4.3 Logistic Regression with Multiple Predictor

Sejauh ini kita sudah mempelajari bagaimana cara mengartikan suatu koefisien jika prediktornya hanya satu kolom kategorikal saja atau satu kolom numerikal saja. Bagaimana jika kita ingin menggunakan banyak prediktor?

```
[42]: train_data.phone_home_valid.unique()
```

```
[42]: array([1, 0], dtype=int64)
```

```
[43]: train_data.head(2)
```

```
[43]: income name_email_similarity prev_address_months_count
current_address_months_count customer_age days_since_request
intended_balcon_amount zip_count_4w velocity_6h velocity_24h velocity_4w
bank_branch_count_8w date_of_birth_distinct_emails_4w credit_risk_score
email_is_free phone_home_valid phone_mobile_valid bank_months_count
has_other_cards proposed_credit_limit foreign_request
session_length_in_minutes keep_alive_session device_distinct_emails_8w month
payment_type_AB payment_type_AC payment_type_AD payment_type_AE
employment_status_CB employment_status_CC employment_status_CD
employment_status_CE employment_status_CF employment_status_CG
housing_status_BB housing_status_BC housing_status_BD housing_status_BE
housing_status_BF housing_status_BG source_TELEAPP device_os_macintosh
device_os_other device_os_windows device_os_x11 fraud_bool intercept
0 0.50000 0.58414 -1
```

```

354          60          0.02091          22.64003          1342
1097.47353    4135.31093    5993.67415          547
4            142            1            1            1
28            1            200.00000            0
6.25370          0            0            1            0            0
0            0            0            0            0            1
0            0            0            0            0
0            0            0            0            0
0            0            0            0            0
0            0            0            0            0
0            0            1            0            0
1 0.10000          0.88506          -1
156          20          0.00458          14.54435          597
4891.02002    3214.06166    3144.47120            1
6            223            1            0            1
28            0            990.00000            0
8.95264          1            1            7            0
0            0            0            0            0
0            0            0            1            0
1            0            0            0            0
0            0            0            1            0
0            0            1

```

```

[44]: model_logit_multi = sm.Logit(train_data['fraud_bool'],
    ↪train_data[['intercept', 'credit_risk_score', 'intended_balcon_amount',
    ↪'has_other_cards', 'phone_home_valid']]).fit()
model_logit_multi.summary()

```

Optimization terminated successfully.
 Current function value: 0.310968
 Iterations 7

```

[44]:

```

Dep. Variable:	fraud_bool	No. Observations:	80000
Model:	Logit	Df Residuals:	79995
Method:	MLE	Df Model:	4
Date:	Sun, 24 Dec 2023	Pseudo R-squ.:	0.1041
Time:	14:49:37	Log-Likelihood:	-24877.
converged:	True	LL-Null:	-27769.
Covariance Type:	nonrobust	LLR p-value:	0.000

	coef	std err	z	P> z	[0.025	0.975]
intercept	-3.0453	0.030	-100.579	0.000	-3.105	-2.986
credit_risk_score	0.0094	0.000	58.248	0.000	0.009	0.010
intended_balcon_amount	-0.0128	0.001	-16.504	0.000	-0.014	-0.011
has_other_cards	-1.2307	0.041	-30.332	0.000	-1.310	-1.151
phone_home_valid	-0.6579	0.026	-24.846	0.000	-0.710	-0.606

Dengan *multiple predictor*, fungsi logit untuk prediksi kita dapat ditulis:

$\$ \text{logit}(p) = -3.0453 + 0.0094 \text{ credit risk score} - 0.0128 \text{ intended balcon amount} \setminus - 1.2307 \text{ has other cards} - 0.6579 \text{ phone home valid} \$$

Cara kita menginterpretasi setiap koefisien masih sama. Hanya saja terdapat sedikit tambahan saat menginterpretasi koefisien prediktor.

- *Intercept* menunjukkan *log of odds* pelanggan membeli produk saat semua prediktor = 0.
- Koefisien prediktor menunjukkan besarnya kenaikan *log of odds* saat semua prediktor lain nilainya tetap.

Sebagai contoh pada `intended_balcon_amount`:

```
[45]: train_data.intended_balcon_amount.describe()
```

```
[45]: count    80000.00000
      mean       8.05265
      std       19.76626
      min      -14.06498
      25%       -1.18521
      50%       -0.84004
      75%       -0.19351
      max       111.69735
      Name: intended_balcon_amount, dtype: float64
```

```
[46]: crs = 50
      balcon0 = 0.48
      balcon1 = 1.48
      cards = 0
      phone = 1

      # log of odds awal
      l1 = -3.0453 + 0.0094*crs - 0.0128*balcon0 - 1.2307*cards - 0.6579*phone

      # log of odds setelah var_2 bertambah 1 unit
      l2 = -3.0453 + 0.0094*crs - 0.0128*balcon1 - 1.2307*cards - 0.6579*phone

      # selisih
      diff = l2-l1

      print(f"Selisih log of odds setelah `intended_balcon_amount` naik sebesar 1_
      ↪unit = {round(diff, 3)}.")
```

Selisih log of odds setelah `intended_balcon_amount` naik sebesar 1 unit = -0.013.

Jika nilai variabel lain tidak berubah, maka dengan kenaikan `intended_balcon_amount` sebesar 1 unit, maka nilai *log of odds* akan berkurang sebesar 0.013.

5.5 Prediksi

Ketika kita sudah berhasil membuat model, normalnya kita akan mencoba melakukan prediksi terhadap data *test* yang sudah kita persiapkan pada tahap *train-test-splitting*. Dalam melakukan prediksi, kita bisa memanfaatkan fungsi `predict()`. Dengan syntax sebagai berikut:

```
<nama_model>.predict(<var_prediktor>)
```

```
[47]: X_test["intercept"] = 1

pred_credit = model_logit_multi.
        ↪predict(X_test[['intercept', 'credit_risk_score', 'intended_balcon_amount',
        ↪                                'has_other_cards',
        ↪                                'phone_home_valid']])

# ubah hasil prediksi dari peluang ke label
pred_label = pred_credit.apply(lambda x: 1 if x > 0.5 else 0)
pred_label.head()
```

```
[47]: 64955    0
      53224    0
      31080    0
      91843    0
      92873    0
      dtype: int64
```

5.6 Evaluasi Model

Untuk mengetahui performa suatu model, kita perlu meninjau hasil prediksi model kita pada data test. Pada kasus ini kita menggunakan satu buah konsep terkait evaluasi kasus klasifikasi yaitu *confusion matrix*.

5.6.1 Confusion Matrix

Pada *confusion matrix* langkah awal yang perlu dilakukan adalah menentukan kelas positif. Ketika kita memiliki kasus klasifikasi, kelas positif adalah kelas yang lebih kita fokuskan atau secara sederhana yang ingin kita tinjau lebih dalam.

Pada kode berikut, kita akan meninjau hasil *confusion matrix* pada model kita:

```
[48]: # buat confusion matrix
      pd.crosstab(pred_label, y_test,
                  rownames=["predicted label"],
                  colnames=["true label"])
```

```
[48]: true label      0      1
      predicted label
      0      17766  2174
      1       28     32
```


Pada data ini, kita akan berfokus kepada kelas 1 yang merupakan akun bank fraud. Perhatikan pada matrix di atas kita memprediksi data lalu membandingkannya dengan data aslinya yang berkelas 0 dan 1. Setiap pengukuran dari matrix ini sebenarnya memiliki nama, yaitu:

- *True Positive* (TP): banyaknya label positif yang diprediksi dengan benar (aktual +, prediksi +)
- *True Negative* (TN): banyaknya label negatif yang diprediksi dengan benar (aktual -, prediksi -)
- *False Positive* (FP): banyaknya label negatif yang diprediksi dengan salah (aktual -, prediksi +)
- *False Negative* (FN): banyaknya label negatif yang diprediksi dengan salah (aktual +, prediksi -)

		AKTUAL	
		0	1
PREDIKSI	0	TN	FN
	1	FP	TP

Dari keempat pengukuran tersebut, kita dapat menilai evaluasi model berdasarkan 4 metrik berikut:

- **Accuracy**: seberapa tepat model kita memprediksi kelas target (secara global).

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN}$$

- **Sensitivity/ Recall**: ukuran kebaikan model terhadap kelas **positif**.

$$Recall = \frac{TP}{TP + FN}$$

- Pos Pred Value/**Precision**: seberapa presisi model memprediksi kelas positif.

$$Precision = \frac{TP}{TP + FP}$$

- **Specificity**: ukuran kebaikan model terhadap kelas **negatif** (jarang digunakan karena merupakan kebalikan dari *recall*).

5.6.2 Accuracy

Kita dapat menghitung nilai akurasi dengan fungsi `accuracy_score()` dari `sklearn`.

[49]: `accuracy_score(y_test, pred_label)`

```
[49]: 0.8899
```

5.6.3 Precision

Kita dapat menghitung nilai *precision* dengan fungsi `precision_score()` dari `sklearn`.

```
[50]: precision_score(y_test, pred_label)
```

```
[50]: 0.5333333333333333
```

5.6.4 Recall/Sensitivity

Kita dapat menghitung nilai *recall* dengan fungsi `recall_score()` dari `sklearn`.

```
[51]: recall_score(y_test, pred_label)
```

```
[51]: 0.014505893019038985
```

5.6.5 Specificity/True Negative Rate (TNR)

Kita dapat menghitung nilai *specificity* dengan fungsi `recall_score()` dan memasukkan parameter `pos_label = 0`.

```
[52]: recall_score(y_test, pred_label, pos_label=0)
```

```
[52]: 0.998426435877262
```

Secara umum sebenarnya kita dapat menggunakan metrik *accuracy*, tetapi terdapat beberapa kondisi ketika kita harus memilih untuk meninjau *recall/precision*:

- *Data imbalance* (proporsi target variabel tidak seimbang)
- Ada kelas yang lebih penting
- Ada resiko yang perlu dipertimbangkan

Tanpa ketiga syarat di atas, kita bisa dengan mudah meninjau nilai akurasi saja. Tetapi contohnya pada kasus kita sekarang, terdapat kelas yang lebih penting yaitu kelas 1 yang merupakan kondisi akun bank fraud. Ketika kita telah mengetahui kelas positifnya maka kita bisa meninjau resiko apa yang terjadi jika kita melakukan salah prediksi:

- FP: diprediksi fraud ternyata bukan fraud
 - risikonya adalah kehilangan nasabah yang mungkin menggunakan berbagai program bank.
- FN: diprediksi bukan fraud ternyata fraud
 - risikonya adalah kehilangan dana untuk menanggulangi kerugian akibat akun fraud ini.

Berdasarkan penjabaran di atas, kita sendiri yang akan menentukan resiko mana yang sebaiknya kita minimalisir. Apabila kita ingin meminimalisir FP maka kita akan menggunakan metrik *precision*, sementara ketika kita ingin meminimalisir FN maka kita akan menggunakan metrik *recall*.

5.7 Logistic Regression Key Assumptions

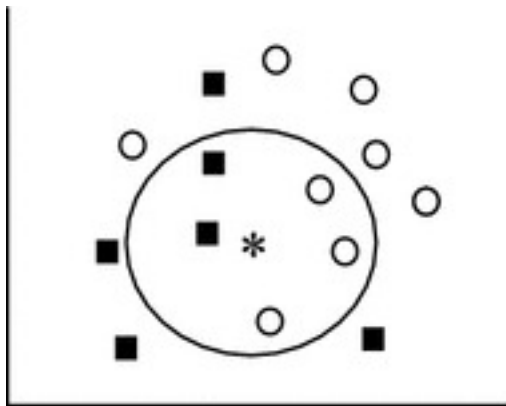
Seperti yang sudah kita pelajari di awal pembahasan mengenai *logistic regression*, *logistic regression* dapat dipandang sebagai *special case* dari regresi linear. Jika regresi linear memiliki asumsi yang harus dipenuhi, *logistic regression* juga memiliki beberapa asumsi:

- **Multikolinearitas**
 - Artinya, setiap prediktor tidak saling berkorelasi tinggi.
- **Observasi yang Independen**
 - Artinya, setiap observasi tidak berasal dari pengukuran yang berulang dan harus independen satu sama lain.
- **Linearitas antara Prediktor dan *Log of Odds***
 - Artinya, terdapat hubungan linear antara prediktor dan *log of odds*.

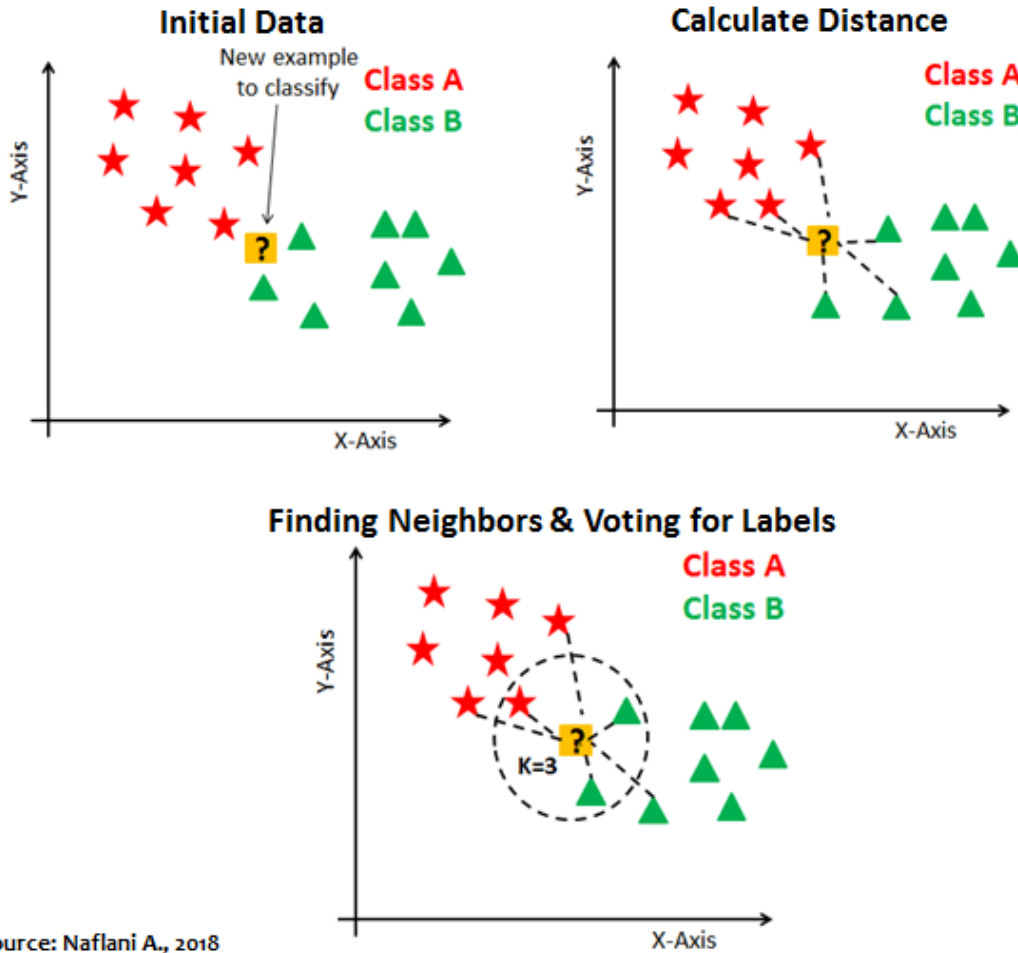
6 Model Classification 2 - K-Nearest Neighbour

Algoritma K-Nearest Neighbor merupakan metode yang menggunakan informasi k tetangga terdekat untuk mengklasifikasi data tanpa label. Ketika dilatih, model ini akan mempelajari informasi lokasi suatu data dan label data tersebut. Informasi lokasi dan label ini yang akan menjadi acuan ketika data tanpa label diidentifikasi. Dari jumlah tetangga terdekat suatu data tanpa label, kelas dominan yang akan dipilih sebagai kelas data tersebut.

Kita akan menggunakan contoh gambar di bawah sebagai ilustrasi. Anggaplah kita memilih nilai $k = 1$ maka tanda $*$ akan menghitung lokasi ke setiap titik di sekitarnya dan mengambil 1 tetangga paling dekat sebagai kelas dirinya. Pada kasus ini artinya dia akan diklasifikasikan sebagai kelas kotak.



Tetapi apabila kita memilih $k = 5$, maka kelas dominan dari lima poin di sekitar tanda $*$ ini adalah tanda bulat. Secara lebih jelasnya, berikut adalah proses pengambilan keputusan oleh k-NN.



6.1 Euclidean Distance

Metode k-NN ini menggunakan perhitungan jarak yang disebut *Euclidean distance* yaitu menghitung jarak paling dekat (seperti menggunakan penggaris untuk menghubungkan dua titik). Apabila kita memiliki dua buah data bernama A dan B dengan fitur seperti berikut:

$$A = (x_1, x_2, \dots, x_m)$$

$$B = (y_1, y_2, \dots, y_m)$$

Pada keterangan di atas, m merupakan dimensi fitur data kita. Untuk melakukan perhitungan jarak antara A dan B, formula *Euclidean distance* yang digunakan yaitu:

$$\text{dist}(A, B) = \sqrt{\sum_{i=1}^m (x_i - y_i)^2}$$

Jika kita aplikasikan formula di atas pada contoh *blind-tasting* suatu makanan, kita dapat menghitung jarak antara kedua makanan di bawah dengan fiturnya masing-masing sebagai berikut:

- tomato (sweet: 6, crunchy: 4)

- green bean (sweet: 3, crunchy: 7)

$\text{dist}(\text{tomato}, \text{green bean}) = \sqrt{((6-3)^2 + (4-7)^2)}$ yang menghasilkan angka 4.24.

6.2 Data Preparation

Pada kasus ini kita akan kembali menggunakan data credit kita sebelumnya. Pada metode k-NN karena kita memperhitungkan nilai berdasarkan jarak maka kita tidak akan menggunakan data kategorikal sama sekali. Mari kita lihat kembali data kita lalu kita ambil data numeriknya saja.

```
[53]: credit_knn = credit.select_dtypes("number")
      credit_knn.head(3)
```

```
[53]:  fraud_bool  income  name_email_similarity  prev_address_months_count
      current_address_months_count  customer_age  days_since_request
      intended_balcon_amount  zip_count_4w  velocity_6h  velocity_24h  velocity_4w
      bank_branch_count_8w  date_of_birth_distinct_emails_4w  credit_risk_score
      email_is_free  phone_home_valid  phone_mobile_valid  bank_months_count
      has_other_cards  proposed_credit_limit  foreign_request
      session_length_in_minutes  keep_alive_session  device_distinct_emails_8w  month
0          0 0.90000          0.61579          -1
51          40          0.01034          -0.69955          2659
11007.70883  7448.79742  5576.30356          2030
7           253          1          1          1
9           1          1500.00000          0
13.07981          0          0          1          1
1          0 0.60000          0.07246          -1
125          20          0.00697          -0.95947          5599
6304.26284  4373.09359  4721.10002          1
13           112          1          1          0
-1           0          200.00000          0
6.66655          0          1          4
2          1 0.50000          0.07659          -1
37          50          0.00632          -1.21993          1352
1357.41355  6667.02966  6318.87626          15
6           199          1          1          1
28           0          1500.00000          0
4.49267          1          2          0
```

Mari kita lanjutkan proses pembuatan model ini dengan membagi data kita menjadi data train dan data test.

```
[54]: y = credit_knn["fraud_bool"]
      X = credit_knn.drop(columns=["fraud_bool"])

      X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.20,
      ↪random_state=42, stratify=y)
```

6.2.1 Features rescaling

Jika kita lihat di atas, data kita tersusun dari berbagai data dengan *range* nilai yang berbeda-beda. Hal ini akan menjadi masalah ketika kita menggunakan metode k-NN yang menghitung nilai berdasarkan jarak. Variabel dengan nilai di antara 1 sampai 500 akan jauh lebih signifikan dibanding variabel dengan nilai 0 sampai 1.

Kondisi ini dapat diselesaikan dengan melakukan *rescaling* keseluruhan variabel kita. Dengan *rescaling* kita dapat memampatkan atau melebarkan nilai data menjadi *range* yang kurang lebih sama. Beberapa metode scaling yang biasa digunakan yaitu:

- Mix-Max normalization
- z-score standardization

Min-max normalization Metode ini bekerja dengan melakukan transformasi nilai data agar berada di jarak 0 sampai 1. Formula yang digunakan:

$$x_{new} = \frac{(x - \min(x))}{(\max(x) - \min(x))}$$

z-score standardization Metode ini bekerja dengan melakukan transformasi berdasarkan informasi mean dan standar deviasi dari data. Formula yang digunakan:

$$x_{new} = \frac{(x - \bar{x})}{std(x)}$$

Metode z-score ini akan menghasilkan nilai dalam skala z-score yang tidak memiliki batas maksimum atau minimum dan dapat bernilai negatif maupun positif.

Pada kasus ini, kita akan menggunakan metode z-score pada data kita. Metode ini dapat diakses menggunakan fungsi `StandardScaler()` dari `sklearn.preprocessing`.

Kita akan mempelajari informasi dari data train dan scaling pada data test akan dilakukan berdasarkan informasi data train untuk mencegah *data leaking*.

```
[55]: scaler = StandardScaler()

scaler.fit(X_train)

X_train_scale = scaler.transform(X_train)
X_test_scale = scaler.transform(X_test)
```

6.3 Modeling

6.3.1 Choosing an appropriate *k*

Penentuan berapa tetangga yang diperhitungkan oleh k-NN akan berdampak kepada seberapa mampu model men-generalisasi data. Keseimbangan antara overfitting dan underfitting data training umumnya dikenal sebagai **bias-variance tradeoff**. Penggunaan nilai k yang besar mengurangi

pengaruh variasi pada *noisy data*, tetapi dapat membuat pembelajaran model menjadi bias yang membuat model menghindari pola penting yang ditangkap dari informasi kecil/detail.

Jika kita menggunakan nilai k yang sangat besar, seperti sejumlah observasi pada data training kita. Hal ini akan menyebabkan model selalu memprediksi kelas dominan. Sebaliknya, jika kita menggunakan nilai $k = 1$ menyebabkan data outliers berpengaruh tidak semestinya terhadap klasifikasi. Jika salah satu data training secara tidak sengaja salah dilabeli dan terdapat satu data yang tepat di sebelahnya, pemilihan $k = 1$ akan menyebabkan misklasifikasi walaupun 10 data tetangga lain memberi vote kelas lain.

Pada prakteknya, salah satu strategi untuk menentukan nilai k yaitu dengan **menghitung akar kuadrat dari jumlah data training**. Strategi lain adalah menggunakan nilai k yang lebih besar tetapi memperhitungkan bobot berdasarkan jarak antar data. Jarak data yang lebih dekat berbobot lebih tinggi dibanding jarak data yang lebih jauh.

Selain itu pembulatan dari perhitungan akar kuadrat ini disesuaikan dengan beberapa syarat lain yaitu:

- k harus ganjil bila jumlah kelas target genap
- k harus genap bila jumlah kelas target ganjil
- k tidak boleh angka kelipatan jumlah kelas target

Hal ini dilakukan untuk menghindari seri ketika proses majority voting dilakukan. Apabila hasil majority voting ini seri maka kelas akan dipilih secara random. Mari kita hitung nilai k pada kasus kita sebelum membuat model.

```
[56]: np.sqrt(X_train.shape[0])
```

```
[56]: 282.842712474619
```

Berdasarkan perhitungan di atas maka kita akan menggunakan nilai $k = 283$.

```
[57]: model_knn = KNeighborsClassifier(n_neighbors = 283)
      model_knn.fit(X_train_scale, y_train)
```

```
[57]: KNeighborsClassifier(n_neighbors=283)
```

6.3.2 Characteristics of k-NN

Metode klasifikasi menggunakan k-NN biasa disebut sebagai '*lazy learners*' karena kita tidak benar-benar membuat model. Pada proses k-NN tidak terjadi proses generalisasi seperti pada model logistic regression. Secara teknikal dapat dikatakan bahwa k-NN hanya menghafal lokasi data tidak benar-benar mempelajari datanya.

Mari kita simpulkan proses yang perlu dilakukan k-NN hingga mendapatkan prediksi:

- Scaling (transformasi variabel agar berada pada *range* nilai yang kurang lebih sama agar tidak ada yang lebih dominan)
- Memilih nilai k
- Membuat model (memilih k tetangga terdekat untuk setiap sampel uji)
- Melakukan klasifikasi berdasarkan kelas dominannya

Karena berbagai alasan di atas, maka secara komputasi proses ini cukup berat apabila kita memiliki data dengan dimensi yang besar (membutuhkan memori yang besar karena melakukan perhitungan jarak secara berulang). Jika kita memilih k yang kecil, metode ini akan sensitif terhadap *noise* di data kita.

Meskipun terdapat berbagai limitasi, k -NN sangatlah *powerful* dan *versatile*. Beberapa kelemahan seperti kondisi outlier dapat diselesaikan dengan scaling yang sudah dipelajari. k -NN juga akan menjadi tidak sensitif terhadap outlier dan noise apabila kita memilih nilai k dengan tepat. Selain itu k -NN mampu bekerja baik pada data non-linear tidak seperti metode logistic regression.

6.4 Prediction

Mari kita kembali ke model kita dan coba melakukan prediksi pada data test kita.

```
[58]: knn_pred = model_knn.predict(X_test_scale)
```

6.5 Evaluation

Seperti model sebelumnya, kita tetap harus meninjau performa model pada data test agar kita dapat menentukan apakah model ini dapat digunakan atau tidak. Mari kita lihat confusion matrix model kita pada data test.

```
[59]: pd.crosstab(knn_pred, y_test,
                rownames=["predicted label"],
                colnames=["true label"])
```

```
[59]: true label      0      1
predicted label
0      17750  2065
1         44   141
```

Mari kita juga tinjau ketiga metrik evaluasi yaitu akurasi, recall, dan precision.

```
[60]: print("Akurasi: ", accuracy_score(y_test, knn_pred))
      print("Recall: ", recall_score(y_test, knn_pred, pos_label=1))
      print("Precision: ", precision_score(y_test, knn_pred, pos_label=1))
```

```
Akurasi:  0.89455
Recall:   0.06391659111514053
Precision: 0.7621621621621621
```

7 Modeling Summary

Secara kesimpulan, sebenarnya kedua model yang telah kita buat belum baik karena metrik utama kita (recall) masih menghasilkan nilai yang sangat kecil. Artinya kita masih perlu melakukan eksplorasi terkait penanganan data maupun tuning model. Hal ini utamanya karena pada kasus ini kita berhadapan dengan data yang imbalance.

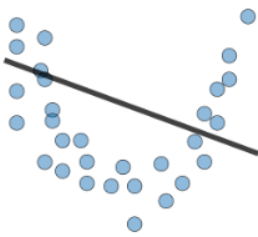
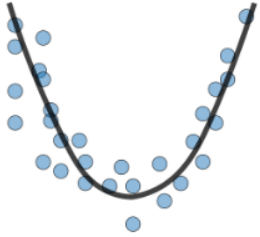
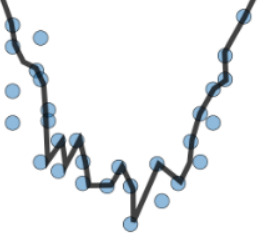
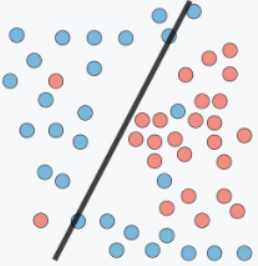
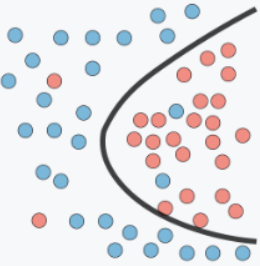
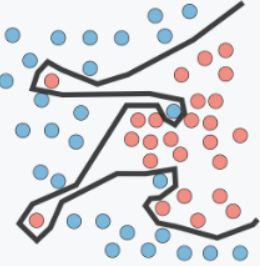
Seperti yang telah disebutkan sebelumnya, terdapat metode seperti oversampling dan undersampling yang mampu mengubah komposisi data kita. Selain itu kita juga dapat melakukan pemilihan

fitur yang digunakan pada model, melakukan *hyperparameter tuning*, dsb.

Berbagai metode di atas tidak akan dibahas di kelas ini dan dapat dieksplorasi mandiri di luar kelas.

7.1 [Additional] Classification Model Selection: Overfit, Underfit, Just Right

Pada bagian ini kita akan membahas istilah *overfitting* yang mengacu pada kondisi di mana model terlalu mempelajari pola pada data sehingga gagal memprediksi data baru. Perhatikan gambar di bawah ini!

	Underfitting	Just right	Overfitting
Symptoms	<ul style="list-style-type: none">• High training error• Training error close to test error• High bias	<ul style="list-style-type: none">• Training error slightly lower than test error	<ul style="list-style-type: none">• Very low training error• Training error much lower than test error• High variance
Regression illustration			
Classification illustration			

Source: stanford.edu - Machine Learning tips and tricks cheatsheet (A. Amidi & S. Amidi)

7.1.1 Overfitting (High Variance)

Seperti yang sudah kita ketahui, bahwa *overfitting* merupakan kondisi di mana model sangat menghafal pola pada data *training*. Bayangkan bahwa data training memiliki *signal*/pola dan *noise*.

Idealnya, kita hanya menginginkan model hanya mempelajari *signal*/pola.

Yang terjadi pada *overfitting*, model juga mempelajari *noise* pada data. Akibatnya, jika diberikan data baru, model gagal melakukan prediksi (gagal melakukan generalisasi).

Gejala *overfitting* dapat diketahui dari hal-hal berikut.

1. Model menunjukkan performa yang baik pada data *training* (tingkat *error* kecil).
2. Model menunjukkan performa yang rendah pada data *testing* (tingkat *error* besar). Perbedaan performa antara *training* dan *testing* sangat besar.

Penyebab *overfitting*:

1. Ukuran untuk data *training* sangat kecil dan tidak representatif.
2. Data yang digunakan untuk *training* banyak mengandung *noise*.
3. Prediktor yang digunakan pada data *training* tidak sesuai.
4. Model yang terlalu kompleks.

Cara mengatasi *overfitting* yang dapat dilakukan:

1. *Adjustment* dari segi model:
 - Menggunakan model yang lebih sederhana.
 - Melakukan *fine-tuning* model:
 - Melakukan *pruning/regularization*.
 - Melakukan *hyperparameter search*.
2. *Adjustment* dari segi data:
 - Menambah data yang digunakan untuk proses *training*.
 - Melakukan *feature selection*.
 - Melakukan *handling* untuk *imbalance dataset*.
3. *Adjustment* dari cara *training*:
 - Melakukan *cross validation*.

7.1.2 Underfitting/High Bias

Berkebalikan dengan *overfitting*, pada *underfitting*, model tidak dapat mendeteksi pola pada data. Akibatnya, model gagal untuk memprediksi data baru.

Gejala *underfitting* dapat diketahui dari hal berikut.

1. Model menunjukkan performa yang rendah pada data *training* (tingkat *error* besar).
2. Model menunjukkan performa yang rendah pada data *testing* (tingkat *error* besar). Perbedaan performa antara *training* dan *testing* tidak terlalu jauh.

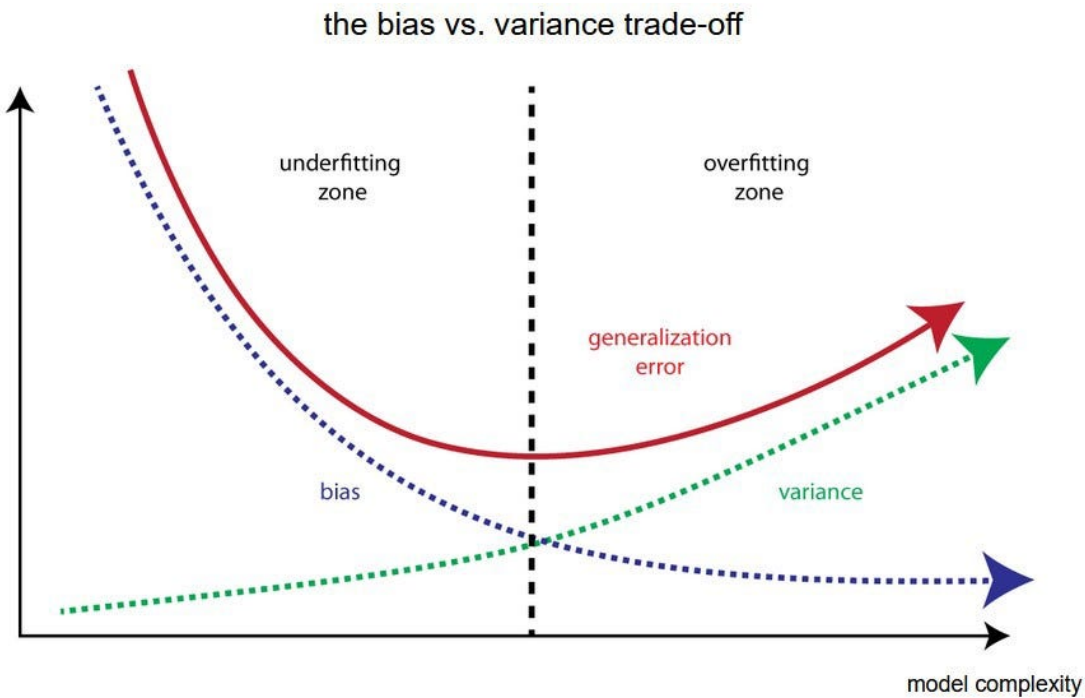
Penyebab *underfitting*:

1. Ukuran untuk data *training* sangat kecil dan tidak representatif.
2. Prediktor yang digunakan pada data *training* tidak sesuai.
3. Model yang terlalu sederhana.

Cara mengatasi *underfitting* yang dapat dilakukan:

1. *Adjustment* dari segi model:
 - Menggunakan model yang lebih kompleks.
 - Melakukan *fine-tuning* model.
2. *Adjustment* dari segi data:
 - Menambah data yang digunakan untuk proses *training*.
 - Mencoba mengganti prediktor yang dimasukkan ke data *training* (*feature selection*).
3. *Adjustment* dari cara *training*:
 - Melakukan *cross validation*.

7.1.3 Trade-Off Underfitting and Overfitting (Bias and Variance Trade-Off)

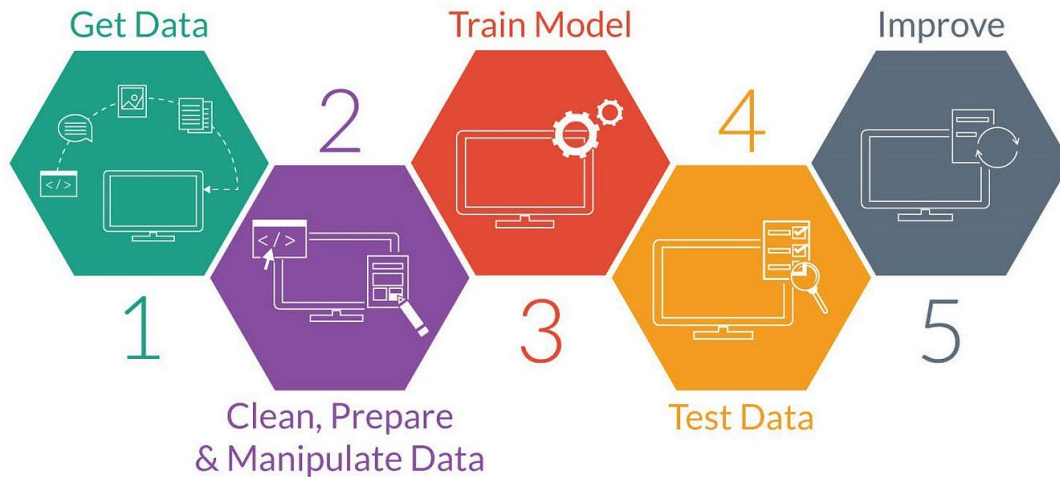


Challenge utama saat membangun *machine learning* adalah menghasilkan model yang merupakan perpotongan dari *underfitting* dan *overfitting*. Artinya, kita ingin membangun model yang tidak terlalu kompleks (tidak *overfit*), tetapi bisa menghasilkan prediksi dengan benar (tidak *underfit*).

Model ideal saat melakukan *machine learning*: memiliki *variance* dan bias yang rendah (tingkat *overfitting* dan *underfitting* rendah).

8 Wrap Up

Pada *course* ini, kita sudah mempelajari banyak hal untuk membuat *machine learning*. Apabila dirunut kembali, apa yang sudah kita lakukan sejauh ini sebenarnya mengikuti *workflow machine learning* secara umum:



1. Mendapatkan data
2. Persiapan data: cek tipe data, cek *missing values*, *feature selection*, *target and predictors splitting*, *one-hot encoding*, *cross validation*, *sampling*, dll.
3. *Modeling*: membuat dan melatih model *machine learning*.
4. Evaluasi performa model: menggunakan *metrics evaluation*.

Proses ini biasanya tidak berjalan satu arah. Terkadang kita sering harus kembali ke *step* sebelumnya. Dalam hal ini, dari *step* 4, kadang kita bisa mengulang lagi dari *step* 2 atau bahkan *step* 1.

Proses *machine learning* sering kali melibatkan serangkaian percobaan dan kesalahan. Pendekatan yang digunakan dalam menyelesaikan suatu masalah tertentu tidak selalu dapat diterapkan secara langsung untuk menyelesaikan masalah lain yang berbeda.