

## 1. Misi Utama Kita: "Menjadi Detektif Sejarah"

Bayangkan kita kembali ke tahun 1912. Kapal Titanic baru saja menabrak gunung es dan tenggelam. Ada ribuan penumpang, tapi sekoci penyelamat (*lifeboat*) jumlahnya sedikit. Tidak semua orang bisa selamat.

**Tugas Kamu:** Kamu diberi sebuah buku daftar penumpang.

- Sebagian penumpang sudah diketahui nasibnya (Ada catatannya: Hidup/Mati).
- Sebagian penumpang lain **DIHAPUS** catatan nasibnya (Misterius).

**Goal Akhir:** Kamu harus membuat **Mesin Prediksi (Model)** yang bisa menebak nasib penumpang yang misterius tadi.

- Apakah si A selamat? (Jawab: **1**)
- Apakah si B meninggal? (Jawab: **0**)

Tebakan kamu nanti akan dinilai oleh Kaggle. Semakin banyak tebakan yang benar, semakin tinggi ranking kamu.

---

## 2. Bedah Data (Kamus Kolom)

Ini bagian paling penting. Kamu harus tahu arti setiap kolom supaya bisa mencari pola (misal: "*Oh, ternyata orang kaya lebih banyak yang selamat*").

Berikut terjemahan lengkap dari **Data Dictionary**:

### A. Target Utama (Yang Mau Ditebak)

- **Survival** : Ini kunci jawabannya.
  - **0** = **No** (Meninggal / Tidak Selamat).
  - **1** = **Yes** (Selamat / Hidup).

### B. Data Profil Penumpang

- **Pclass (Ticket Class)**: Kelas Tiket. Ini penanda **Status Sosial & Kekayaan**.
  - **1** = Kelas 1 (Eksekutif/Sultan). Posisi kamar biasanya di dek paling atas (dekat sekoci). Contoh di film: *Rose*.
  - **2** = Kelas 2 (Menengah/Bisnis).
  - **3** = Kelas 3 (Ekonomi/Rakyat Jelata). Posisi kamar biasanya di paling bawah kapal (susah keluar saat tenggelam). Contoh di film: *Jack*.
- **Sex** : Jenis Kelamin ( `male` / `female` ).
  - *Ingat aturan laut: "Women and Children First"* (Wanita dan anak-anak didahulukan masuk sekoci). Jadi kemungkinan wanita selamat lebih tinggi.

- **Age** : Umur dalam tahun.
    - Bayi/Anak kecil biasanya diprioritaskan selamat.
  - **SibSp (Siblings / Spouses)**: Jumlah **Saudara Kandung** atau **Suami/Istri** yang dibawa.
    - Ini menandakan dia bawa keluarga "kesamping" (kakak/adik/pasangan).
  - **Parch (Parents / Children)**: Jumlah **Orang Tua** atau **Anak** yang dibawa.
    - Ini menandakan dia bawa keluarga "keatas/bawah" (ayah/ibu/anak).
    - *Gabungan SibSp + Parch bisa dipakai buat hitung total rombongan keluarga.*
  - **Ticket** : Nomor Tiket.
    - Biasanya angkanya acak (misal: A/5 21171). Kadang susah dicari polanya, tapi kadang tiket yang kodenya sama berarti mereka sekamar.
  - **Fare** : Harga Tiket.
    - Berapa duit yang mereka bayar. Biasanya nyambung sama **Pclass**. Makin mahal, makin fasilitas VIP.
  - **Cabin** : Nomor Kamar Kabin.
    - Contoh: C85 , B42 . Huruf depannya (C, B, D) menunjukkan **Dek/Lantai** kapal.
    - *Masalah*: Data ini banyak yang kosong (NaN) karena pencatatannya buruk.
  - **Embarked** : Pelabuhan Keberangkatan (Dari mana dia naik kapal?).
    - **C = Cherbourg** (Prancis). Biasanya banyak orang kaya naik dari sini.
    - **Q = Queenstown** (Irlandia). Biasanya banyak orang kelas 3 (imigran) naik dari sini.
    - **S = Southampton** (Inggris). Pelabuhan awal berangkat.
- 

### 3. Bedah File (Apa Bedanya Train dan Test?)

Di folder `data/` kamu ada 3 file. Jangan sampai salah buka!

#### 1. `train.csv` (Buku Pelajaran)

- **Isinya**: Data penumpang LENGKAP dengan kunci jawaban (`Survived`).
- **Gunanya**: Ini makanan buat si Robot (Model). Kamu kasih data ini biar dia **BELAJAR**.
  - *Robot mikir*: "Oh, ternyata kalau Cewek (Sex=female) dan Kelas 1 (Pclass=1), rata-rata selamat (1). Oke aku ingat pola itu."

#### 2. `test.csv` (Soal Ujian)

- **Isinya**: Data penumpang TAPI **kolom Survived -nya DIHAPUS/HIDDEN**.
- **Gunanya**: Setelah robot pintar, kamu kasih data ini.
  - *Kamu*: "Nih, ada penumpang baru. Tebak dia selamat apa gak?"
  - *Robot*: "Oke, berdasarkan pola yang aku pelajari tadi, orang ini selamat!"
- Hasil tebakan robot inilah yang nanti kita simpan dan upload ke Kaggle.

#### 3. `gender_submission.csv` (Contoh Jawaban)

- Ini cuma file contoh dari Kaggle. Isinya tebakan bodoh: "Pokoknya kalau Cewek Selamat, kalau Cowok Mati."
- Kita gak pakai file ini buat coding, cuma buat referensi format upload aja.

## Deskripsi Data Titanic

### Dataset Description

#### Overview

The data has been split into two groups:

- training set (train.csv)
- test set (test.csv)

**The training set** should be used to build your machine learning models. For the training set, we provide the outcome (also known as the "ground truth") for each passenger. Your model will be based on "features" like passengers' gender and class. You can also use **feature engineering** to create new features.

**The test set** should be used to see how well your model performs on unseen data. For the test set, we do not provide the ground truth for each passenger. It is your job to predict these outcomes. For each passenger in the test set, use the model you trained to predict whether or not they survived the sinking of the Titanic.

We also include **gender\_submission.csv**, a set of predictions that assume all and only female passengers survive, as an example of what a submission file should look like.

#### Data Dictionary

Variable	Definition	Key
survival	Survival	0 = No, 1 = Yes
pclass	Ticket class	1 = 1st, 2 = 2nd, 3 = 3rd
sex	Sex	
Age	Age in years	
sibsp	# of siblings / spouses aboard the Titanic	
parch	# of parents / children aboard the Titanic	
ticket	Ticket number	
fare	Passenger fare	
cabin	Cabin number	
embarked	Port of Embarkation	C = Cherbourg, Q = Queenstown, S = Southampton

## Variable Notes

**pclass:** A proxy for socio-economic status (SES)

1st = Upper

2nd = Middle

3rd = Lower

**age:** Age is fractional if less than 1. If the age is estimated, is it in the form of xx.5

**sibsp:** The dataset defines family relations in this way...

Sibling = brother, sister, stepbrother, stepsister

Spouse = husband, wife (mistresses and fiancés were ignored)

**parch:** The dataset defines family relations in this way...

Parent = mother, father

Child = daughter, son, stepdaughter, stepson

Some children travelled only with a nanny, therefore `parch=0` for them.

*Gambar 1. Overview Dataset (Klik gambar untuk mengunjungi Kaggle)*

# Exploratory Data Analysis (EDA)

```
In [1]: import pandas as pd
import numpy as np

import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline

from sklearn.preprocessing import OrdinalEncoder, OneHotEncoder

from sklearn.impute import SimpleImputer

from sklearn.compose import make_column_transformer, ColumnTransformer
from sklearn.pipeline import Pipeline, make_pipeline

from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.naive_bayes import GaussianNB
from xgboost import XGBClassifier
from sklearn.ensemble import RandomForestClassifier, AdaBoostClassifier, GradientBo

from sklearn.model_selection import cross_val_score, StratifiedKFold, train_test_sp

# Setting untuk membuat angka mudah dibaca di display
pd.options.display.float_format = '{:20.2f}'.format
```

```
# Menampilkan semua kolom pada output
pd.set_option('display.max_columns', None)
```

In [2]: train\_df = pd.read\_csv('../data/train.csv')  
test\_df = pd.read\_csv('../data/test.csv')

In [3]: train\_df.head(100) # Saya murni penasaran, umumnya cukup 5 aja.

Out[3]:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare
0	1	0	3	Braund, Mr. Owen Harris	male	22.00	1	0	A/5 21171	7
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th... Heikkinen, Miss. Laina	female	38.00	1	0	PC 17599 STON/O2. 3101282	71 7
2	3	1	3	Futrelle, Mrs. Jacques Heath (Lily May Peel) Allen, Mr. William Henry	female	26.00	0	0	113803 373450	53 8
3	4	1	1	Shorney, Mr. Charles Joseph	male	35.00	1	0	374910	8
4	5	0	3	Goldschmidt, Mr. George B	male	71.00	0	0	PC 17754	34
5	...	...	...	Greenfield, Mr. William Bertram	male	23.00	0	1	PC 17759	63
6	96	0	3	Doling, Mrs. John T (Ada Julia Bone)	female	34.00	0	1	231919	23
7	97	1	1	Kantor, Mr. Sinai	male	34.00	1	0	244367	26
8	98	1	2							
9	99	0	2							
10	100	0	2							

100 rows × 12 columns



In [4]: test\_df.head(100)

Out[4]:

	PassengerId	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	E
0	892	3	Kelly, Mr. James	male	34.50	0	0	330911	7.83	NaN	
1	893	3	Wilkes, Mrs. James (Ellen Needs)	female	47.00	1	0	363272	7.00	NaN	
2	894	2	Myles, Mr. Thomas Francis	male	62.00	0	0	240276	9.69	NaN	
3	895	3	Wirz, Mr. Albert	male	27.00	0	0	315154	8.66	NaN	
4	896	3	Hirvonen, Mrs. Alexander (Helga E Lindqvist)	female	22.00	1	1	3101298	12.29	NaN	
...	...	...	...	...	...	...	...	...	...	...	...
95	987	3	Tenglin, Mr. Gunnar Isidor	male	25.00	0	0	350033	7.80	NaN	
96	988	1	Cavendish, Mrs. Tyrell William (Julia Florence...)	female	76.00	1	0	19877	78.85	C46	
97	989	3	Makinen, Mr. Kalle Edvard	male	29.00	0	0	STON/O 2. 3101268	7.92	NaN	
98	990	3	Braf, Miss. Elin Ester Maria	female	20.00	0	0	347471	7.85	NaN	
99	991	3	Nancarrow, Mr. William Henry	male	33.00	0	0	A/5. 3338	8.05	NaN	

100 rows × 11 columns

(Pure komentar saya :) Baik, disini bedanya antara file `test.csv` dan `train.csv` adalah tidak adanya kolom `Survived` di `test.csv`, sesuai dengan ekspektasi kita, karena kita ingin mencoba untuk memprediksi apakah mereka selamat atau mati untuk file `test.csv` dan disubmit ke kaggle, seperti contoh pada file `gender_submission`.

Mari kita coba pahami terlebih dahulu pelan-pelan untuk masing-masing kolom.

- **PassengerId** : Bisa dikatakan sebagai identitas unik untuk setiap satu penumpang
- **Survived** : Kolom yang menjadi target kita untuk prediksi. **1** berarti selamat dan **0** berarti mati/tidak selamat
- **Pclass** : Kelas dari tiket yang dibeli dan sebagai penanda Status Sosial & Kekayaan. Dimana kelas pertama yang paling bagus, kelas kedua menengah dan kelas ketiga buat ekonomi (yang paling jelek). Diliat dari overview tabel, tidak ada angka yang aneh.
- **Name** : Nama penumpang, yang saya yakin tidak akan dipakai, karena nama orang sama sekali tidak akan menentukan orang itu selamat atau tidak.
- **Sex** : Jenis Kelamin. Disini ada funfact dan mungkin akan menjadi pertimbangan yaitu : "Women and Children First". Dengan begitu bisa menjadi faktor yang perlu dicatat untuk pembuatan model. Saya masih belum tau apakah video tutorial ini akan membahas tentang hal tersebut. Lalu ini juga masih berupa kata-kata, belum diubah dalam angka. Komputer tidak paham apa itu male atau female.

"Women and children first" is a traditional, unofficial rule of conduct prioritizing women and children for rescue in emergencies like shipwrecks, originating from the 1852 sinking of the HMS Birkenhead, where soldiers sacrificed themselves for women and children. While popularized by the Birkenhead and echoed in the Titanic disaster, it's not maritime law, though the principle of protecting the vulnerable persists in various contexts, from disaster relief to healthcare resource allocation.

## Funfact

### Origin & History

- **HMS Birkenhead (1852)**: The first widely known instance, where soldiers stood their ground, allowing women and children to board the limited lifeboats, with all of them surviving, as detailed in [the YouTube video](#).
- **RMS Titanic (1912)**: The principle was again observed, though often attributed to captain's orders rather than purely spontaneous chivalry, leading to higher survival rates for women and children, notes [this YouTube video](#).
- **Maritime Law**: Despite these famous examples, the rule remains an informal code, not a legally binding maritime regulation, says the [Wikipedia article](#).

- **Age** : Umur penumpang. Disini ada funfact dan mungkin akan menjadi pertimbangan yaitu : "Women and Children First". Bayi/Anak kecil biasanya diprioritaskan selamat. Dengan begitu bisa menjadi faktor yang perlu dicatat untuk pembuatan model. Saya masih belum tau apakah video tutorial ini akan membahas tentang hal tersebut. Dilihat

pertama kali. Ada beberapa yang aneh yaitu umur dengan belakang koma .05, lalu ada juga `NAN` alias not a number. Berdasarkan dokumentasi data :

Age is fractional if less than 1. If the age is estimated, is it in the form of xx.5

- **SibSp** : Siblings or Spouses. Jumlah saudara kandung atau suami/istri yang dibawa. (Hubungan kesamping)

The dataset defines family relations in this way:

- **Sibling** = brother, sister, stepbrother, stepsister
- **Spouse** = husband, wife (mistresses and fiancés were ignored)  
Sebagian besar nilainya adalah `0` dan `1` jika dilihat sekilas

- **Parch** : (Parents / Children): Jumlah Orang Tua atau Anak yang dibawa. (hubungan ke atas/bawah)

The dataset defines family relations in this way...

- **Parent** = mother, father
- **Child** = daughter, son, stepdaughter, stepson Some children travelled only with a nanny (pengasuh), therefore `parch=0` for them.

Kebanyakan nilainya 1 dan 0 juga, penasaran maksudnya apakah ortu dianggap nya 1 kah kalau dari pov anak. Terus juga dari pov si orang tua, apakah 1 dianggap 1 anak? Gabungan **SibSp + Parch** bisa dipakai buat hitung total rombongan keluarga.

- **Ticket** : Nomor tiket. Dilihat sekilas, ada no tiker seperti : `A/5 21171 STON/02. 3101282 PC 17599` dan `231919`. Nah apa maksudnya? kenapa berbeda-beda? ada lumayan banyak yang hanya angka dan depannya `PC`. Kadang susah dicari polanya, tapi kadang tiket yang kodennya sama berarti mereka sekamar.
- **Fare** : Tarif penumpang. Berapa duit yang mereka bayar. Biasanya nyambung sama Pclass. Makin mahal, makin fasilitas VIP.
- **Cabin** : Nomor kabin. Banyak sekali jika dilihat datanya berupa `NAN`. Ini akan menjadi pr kita untuk memutuskan startegi apa yang kita ambil untuk mengatasi masalah ini. Contoh: C85, B42. Huruf depannya (C, B, D) menunjukkan **Dek/Lantai** kapal.
- **Embarked** : Pelabuhan Keberangkatan (Dari mana dia naik kapal?).
  - **C** = **Cherbourg** (Prancis). Biasanya banyak orang kaya naik dari sini.
  - **Q** = **Queenstown** (Irlanidia). Biasanya banyak orang kelas 3 (imigran) naik dari sini.
  - **S** = **Southampton** (Inggris). Pelabuhan awal berangkat. Namun itu masih asumsi, perlu dianalisis lebih lanjut

train\_df.describe()

	PassengerId	Survived	Pclass	Age	SibSp	Parch	Fare
count	891.000000	891.000000	891.000000	714.000000	891.000000	891.000000	891.000000
mean	446.000000	0.383838	2.308642	29.699118	0.523008	0.381594	32.204208
std	257.353842	0.486592	0.836071	14.526497	1.102743	0.806057	49.693429
min	1.000000	0.000000	1.000000	0.420000	0.000000	0.000000	0.000000
25%	223.500000	0.000000	2.000000	20.125000	0.000000	0.000000	7.910400
50%	446.000000	0.000000	3.000000	28.000000	0.000000	0.000000	14.454200
75%	668.500000	1.000000	3.000000	38.000000	1.000000	0.000000	31.000000
max	891.000000	1.000000	3.000000	80.000000	8.000000	6.000000	512.329200

+ Code + Markdown

give mean your standard deviation Min and

**Beginner Data Science Portfolio Project Walkthrough (Kaggle Titanic)**

Ryan & Matt Data Science 34.3K subscribers

Join Subscribe

1.4K

Welcome to my data science journey through the Kaggle Titanic - Machine Learning from Disaster Project!

In this video, we'll dive deep into the world of data analysis, feature engineering, and machine learning to predict passenger survival rates on the

Gambar 2. Overview Video Tutorial (Klik gambar untuk mengunjungi Source Video)

Mari kita *highlight* beberapa poin dari analisis saya dan sinkronkan dengan pendapat si Ryan (Youtuber):

## 1. Intuisi tentang Name (Ryan vs Kamu)

- **Kamu:** "Nama orang sama sekali tidak akan menentukan orang itu selamat atau tidak."  
→ Logis banget. Masa gara-gara nama "Budi" dia lebih gampang tenggelam?
- **Ryan:** "Name does give you some interesting information... specifically like what type of class someone is in... married not married."
  - **Insight Ryan:** Ryan melihat sesuatu yang tersembunyi. Gelar seperti "**Mr.**", "**Mrs.**", "**Miss.**", "**Master**", "**Dr.**", "**Countess**".
  - **Contoh:** "Master" itu biasanya anak laki-laki kecil (Anak-anak prioritas selamat). "Countess" itu bangsawan (Mungkin diprioritaskan). "Rev" (Pendeta) mungkin memilih mati demi orang lain.
  - **Kesimpulan:** Nama itu sendiri gak penting, tapi **GELAR (Title)** di dalam nama itu yang nanti bakal kita ambil (Feature Engineering).

## 2. Intuisi tentang Age & Cabin (Data Bolong)

- **Kamu:** "Banyak sekali jika dilihat datanya berupa NaN. Ini akan menjadi pr kita untuk memutuskan startegi apa."
- **Ryan:** Setuju. Dia bilang "Guess what we already have our null values in Cabin so that means we have some work to do."
  - Untuk `Age`, Ryan bilang bisa pakai **SimpleImputer** (mengisi dengan rata-rata). (Apa itu **SimpleImputer**? Coba nanti kita pelajari cara handle missing data)
  - Untuk `Cabin`, Ryan bilang "Decide if we're gonna drop cabin or take a look at it from another perspective." (Nanti kita lihat triknya Ryan, biasanya dia ambil huruf depannya aja: A, B, C...).

## 3. Intuisi tentang Ticket

- **Kamu:** "Kadang susah dicari polanya, tapi kadang tiket yang kodennya sama berarti mereka seamar."
- **Analisis:** Tepat! Tiket dengan kode sama = Satu grup/keluarga. Kalau satu keluarga selamat, biasanya yang lain ikut selamat (atau sebaliknya).

## 4. PassengerId

- **Ryan:** "I don't think we need that for our machine learning model."
  - Benar. ID itu cuma nomor urut absen. Gak ada hubungannya sama nasib hidup mati. Nanti kolom ini pasti dibuang pas training.

In [5]: `train_df.info()`

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

In [6]: `train_df.describe()`

Out[6]:

	PassengerId	Survived	Pclass	Age	SibSp	Parch	Fare
<b>count</b>	891.00	891.00	891.00	714.00	891.00	891.00	891.00
<b>mean</b>	446.00	0.38	2.31	29.70	0.52	0.38	32.20
<b>std</b>	257.35	0.49	0.84	14.53	1.10	0.81	49.69
<b>min</b>	1.00	0.00	1.00	0.42	0.00	0.00	0.00
<b>25%</b>	223.50	0.00	2.00	20.12	0.00	0.00	7.91
<b>50%</b>	446.00	0.00	3.00	28.00	0.00	0.00	14.45
<b>75%</b>	668.50	1.00	3.00	38.00	1.00	0.00	31.00
<b>max</b>	891.00	1.00	3.00	80.00	8.00	6.00	512.33

In [7]:

train\_df.describe(include=['O'])

Out[7]:

	Name	Sex	Ticket	Cabin	Embarked
<b>count</b>	891	891	891	204	889
<b>unique</b>	891	2	681	147	3
<b>top</b>	Braund, Mr. Owen Harris	male	347082	G6	S
<b>freq</b>	1	577	7	4	644

Kalau dari sini aku ngambil insight nya berarti :

`Survived` hanya 38 persen saja yang survived, `Pclass` kebanyakan di kelas 3? karena kalau lihat data posisi ke 50 persen - max itu di 3. `Age` rata rata di umur 29-30 tahun. Dimana menurut video tutorialnya nya itu dia termasuk muda, expect lebih tua katanya. `SibSp` kebanyakan cuman 0 yaa, dan juga 1. Namun ada ternyata ada yang punya 8. Lalu `Parch` kebanyakan 0, namun ada juga yang memiliki 6. dan `Fare` (harga tiket) yang rata rata di 32.20. Namun ada yang beli sampe harga 512.33 ( kalau dilihat sekilas outlier karena, data ke 75 persen itu adalah 31.00).

Beginu pula untuk tipe data object yaitu `Name` keliatan normal tidak ada nama yang dobel. `Sex` kebanyakan laki-laki dibandingkan perempuan. `Ticket` yang unik lebih sedikit dibandingkan dengan jumlah aslinya, menandakan ada yang sama tiketnya. Sangat masuk akal melihat ada yang membawa anak/saudara dengan jumlah terbanyaknya adalah 7 tiket yang sama. Beginu pula dengan `Cabin`. Lalu ada `Embarked` yang kebanyakan berangkat dari `S` yaitu **S = Southampton** (Inggris).

Mari kita bedah sedikit lagi dan sinkronkan dengan pandangan Ryan (dan logika Data Science).

## 1. Survived (0.38)

- **Kamu:** "Hanya 38% yang selamat."
- **Fakta:** Betul. Ini adalah *Baseline Accuracy*.
  - Kalau kamu tebak "SEMUA ORANG MATI", akurasi kamu otomatis **62%** ( $100\% - 38\%$ ).
  - Tujuan Model kita nanti harus bisa mengalahkan angka 62% ini. Kalau model canggih kita cuma dapet akurasi 60%, mending nggak usah bikin model.

## 2. Pclass (2.31)

- **Kamu:** "Kebanyakan di kelas 3."
- **Logika:** Betul.
  - Median (50%) = 3.
  - $75\% = 3$ .
  - Artinya **lebih dari separuh penumpang** adalah rakyat jelata (Kelas 3). Ini penting karena Kelas 3 posisinya paling bawah di kapal (paling susah lari).

## 3. Fare (Outlier Alert!) 🚨

- **Kamu:** "Rata-rata 32, tapi ada yang 512. Outlier karena 75% cuma 31."
- **Analisis:** Tajam!
  - Jarak antara Q3 (31) ke Max (512) itu jauh banget.
  - Ini mirip kasus "Sultan" di project Retail kemarin.
  - **Fun Fact:** Harga 512 itu dibayar oleh Cardeza bersaudara (orang super kaya) yang menempati Suite Mewah dengan 3 kamar tidur + dek pribadi.

## 4. Ticket (Duplikat)

- **Kamu:** "Unik lebih sedikit... menandakan ada yang sama tiketnya... terbanyak 7 tiket sama."
- **Analisis:**
  - Total Tiket: 891.
  - Unik: 681.
  - Artinya ada sekitar 200 orang yang berbagi tiket (Tiket grup/keluarga).
  - Di tabel `freq` terlihat 347082 muncul 7 kali. Artinya satu tiket ini dipakai 7 orang. Ini pasti keluarga besar.

## 5. Cabin (Data Bolong Parah)

- **Kamu:** "Cabin uniknya 147... count 204."
- **Analisis:**
  - Total Penumpang: 891.
  - Yang punya info Cabin: Cuma 204.
  - Berarti ada **687 penumpang (77%)** yang data Cabin-nya HILANG ( `Nan` ).

- **Strategi Nanti:** Ryan pasti akan mikir keras di sini. Apakah kolom Cabin dibuang aja? Atau kita ambil huruf depannya doang? (Biasanya kalau bolong > 50%, opsinya dibuang atau dibikin kategori baru "Unknown").
- 

## Kesimpulan Sementara

Kamu sudah paham betul kondisi datanya:

1. **Imbalance:** Lebih banyak yang mati daripada selamat.
2. **Missing Values:** Age bolong dikit, Cabin bolong parah.
3. **Outliers:** Harga tiket ada yang gila-gilaan.
4. **Karakteristik:** Kebanyakan cowok, kelas 3, berangkat dari Inggris (S).

```
In [8]: train_df.groupby(["Pclass"], as_index=False)[ "Survived" ].mean()
```

```
Out[8]: Pclass  Survived
```

Pclass	Survived	
0	1	0.63
1	2	0.47
2	3	0.24

Cara bacanya disini adalah, kebanyakan penumpang kelas 1 selamat dibandingkan dengan kelas 2 dan 3. Hipotesis kita mengenai bahwa semakin tinggi kelas, semakin besar kemungkinan survived adalah benar. Khususnya yang kelas 3 sangat amat kecil.

```
In [9]: train_df.groupby(["Sex"], as_index=False)[ "Survived" ].mean()
```

```
Out[9]: Sex  Survived
```

Sex	Survived	
female	0.74	
male	0.19	

Aturan tak tertulis yang terkenal tadi dimana, "Women and children first" ternyata benar. Bahwasannya sangat jauh lebih banyak female yang survived dibandingkan male

```
In [10]: train_df.groupby(["SibSp"], as_index=False)[ "Survived" ].mean()
```

Out[10]:

	SibSp	Survived
0	0	0.35
1	1	0.54
2	2	0.46
3	3	0.25
4	4	0.17
5	5	0.00
6	8	0.00

In [11]:

```
train_df.groupby(["Parch"], as_index=False)[ "Survived" ].mean()
```

Out[11]:

	Parch	Survived
0	0	0.34
1	1	0.55
2	2	0.50
3	3	0.60
4	4	0.00
5	5	0.20
6	6	0.00

In [12]:

```
train_df[ 'Family_Size' ] = train_df[ 'SibSp' ] + train_df[ 'Parch' ] + 1  
test_df[ 'Family_Size' ] = test_df[ 'SibSp' ] + test_df[ 'Parch' ] + 1
```

In [13]:

```
train_df.head(100)
```

Out[13]:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	F
0	1	0	3	Braund, Mr. Owen Harris	male	22.00	1	0	A/5 21171	7
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th... Heikkinen, Miss. Laina	female	38.00	1	0	PC 17599	71
2	3	1	3	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	26.00	0	0	STON/O2. 3101282	7
3	4	1	1	Allen, Mr. William Henry	male	35.00	0	0	113803	53
4	5	0	3	Shorley, Mr. Charles Joseph	male	NAN	0	0	373450	8
...	...	...	...	...	...	...	...	...	...	...
95	96	0	3	Goldschmidt, Mr. George B	male	71.00	0	0	PC 17754	34
97	98	1	1	Greenfield, Mr. William Bertram	male	23.00	0	1	PC 17759	63
98	99	1	2	Doling, Mrs. John T (Ada Julia Bone)	female	34.00	0	1	231919	23
99	100	0	2	Kantor, Mr. Sinai	male	34.00	1	0	244367	26

100 rows × 13 columns



In [14]: `train_df.groupby(["Family_Size"], as_index=False)[["Survived"]].mean()`

Out[14]:

	Family_Size	Survived
0	1	0.30
1	2	0.55
2	3	0.58
3	4	0.72
4	5	0.20
5	6	0.14
6	7	0.33
7	8	0.00
8	11	0.00

angka-angka 2,3,4 memiliki kesempatan untuk survived yang lumayan tinggi dan diatas rata-rata yang survived secara keseluruhan. Namun jujur saya masih gak nangkep kenapa kok hal ini bisa berguna? Mungkin kalau keluarganya banyak, apakah bikin kerepotan kah? atau gimana?? saya masih belum tau atau nangkep kenapa kok ini dijadikan faktor. Video tutorial ini menganggap data ini interesting

Oke, mari kita bedah penjelasan Ryan soal **Family Size** dengan bahasa yang lebih sederhana.

Inti dari transkrip itu adalah dia mencoba membuat **Kolom Baru** bernama **Family\_Size** (Ukuran Keluarga) karena dia merasa **SibSp** (Saudara/Pasangan) dan **Parch** (Ortu/Anak) itu kalau dipisah kurang "bercerita".

## 1. Logika Rumus Family\_Size

Ryan bilang: "*adding both SibSp and also Parch and then plus yourself*"

Rumusnya:

$$\text{Family Size} = \text{SibSp} + \text{Parch} + 1$$

- **SibSp:** Jumlah saudara/istri yang ikut.
- **Parch:** Jumlah orang tua/anak yang ikut.
- **+ 1:** Diri kamu sendiri (Si Penumpang itu).

### Contoh:

- Budi naik Titanic sama Istrinya.
  - **SibSp** = 1 (Istri).
  - **Parch** = 0 (Gak bawa anak/ortu).
  - **Family\_Size** =  $1 + 0 + 1 = 2$ .
- Siti naik sendirian (Jomblo).

- `SibSp` = 0.
  - `Parch` = 0.
  - `Family_Size` =  $0 + 0 + 1 = 1$ .
- 

## 2. Temuan Ryan (Analisis Survival Rate)

Setelah dia bikin kolom baru itu, dia melakukan `groupby` (sama kayak yang kamu lakukan tadi).

### Apa yang dia temukan?

- **Family Size = 1 (Sendirian):** Survival rate **0.3 (30%)**.
  - *Kata Ryan:* "A little bit below the mean." (Rata-rata selamat kan 38%, jadi sendirian itu kurang aman).
- **Family Size = 2, 3, 4 (Keluarga Kecil):** Survival rate **Tinggi (50% - 70%)**.
  - *Kata Ryan:* "Good survival... you have a few different family members... a lot higher survival rate."
- **Family Size = 5, 6, 7... 11 (Keluarga Besar):** Survival rate **Jelek Banget (Mendekati 0)**.
  - *Kata Ryan:* "Once you get a little bit higher... your chances of survival are not that good."

**Kesimpulan Ryan:** Punya keluarga *sedikit* (2-4 orang) itu menguntungkan (bisa saling bantu). Tapi kalau keluarganya *kebanyakan* (>5 orang), malah jadi beban dan sulit selamat bareng-bareng.

---

## Kenapa Ini Penting?

Karena nanti saat kita bikin Model Machine Learning, kolom `Family_Size` ini akan menjadi **Petunjuk Kuat (Strong Feature)** buat si Robot.

Robot akan mikir: "*Oh, kalau dia sendirian (Size 1), kurangi nilai keselamatannya dikit. Kalau dia berempat (Size 4), tambah nilai keselamatannya. Kalau dia sekeluarga besar (Size 8), anggap dia bakal mati.*"

Jadi intinya Ryan cuma menggabungkan dua kolom (`SibSp` + `Parch`) menjadi satu kolom baru yang lebih "bermakna" (`Family_Size`).

Genuine question dari saya : oke aku tau, cumann masa sih hal itu masuk akal?

Nah, ini yang namanya **Critical Thinking!** Jangan telan mentah-mentah omongan tutorial.

Kamu ragu: "*Masa sih punya keluarga 3 orang bikin kita lebih selamat daripada sendirian? Bukannya sendirian lebih gampang lari?*"

Mari kita debat secara logis (dan historis).

### Argumen Pro-Sendirian (Logika Kamu):

- Lari lebih cepat, nggak perlu nungguin siapa-siapa.
- Bisa nyempil di sekoci mana aja tanpa mikirin "Istriku mana?".
- *Ini masuk akal.*

### TAPI... Kenapa Datanya Bilang Sebaliknya? (Logika Sejarah Titanic):

#### 1. Faktor Psikologis & Informasi:

- Titanic itu kapal raksasa dan membingungkan (labirin).
- Orang yang sendirian (terutama kelas 3) seringkali bingung, panik, dan tidak tahu harus ke mana. Mereka diam di kamar atau tersesat.
- Orang yang berkeluarga saling mengingatkan: "*Pak, bangun! Ada air masuk! Ayo kita ke atas!*". Ada *Support System*.

#### 2. Faktor "Ibu dan Anak" (The Golden Ticket):

- Ingat aturan **Women and Children First?**
- Kalau kamu Cowok Dewasa Sendirian → Petugas: "Minggir Mas, antri belakang!" (Peluang mati tinggi).
- Kalau kamu Suami yang bawa Istri & Anak → Petugas: "Istri dan Anak masuk sekoci!". Si Suami mungkin mati, tapi Istri & Anaknya selamat.
- Dalam statistik `Family_Size`, kalau 2 dari 3 orang selamat, rata-ratanya jadi tinggi (66%). Kalau sendirian mati, rata-ratanya 0%.

#### 3. Faktor "Rantai Manusia" (Keluarga Besar - Sisi Buruk):

- Kalau keluarganya 11 orang (Ayah, Ibu, 9 Anak).
- Mereka bergerak lambat kayak rombongan itik.
- Petugas sekoci gak mau masukin rombongan besar sekaligus.
- Keluarga besar biasanya menolak dipisah ("Kalau Ibu gak ikut, aku gak mau naik!"). Akhirnya mati semua.

**Kesimpulan:** Angka statistik (55-70% selamat) itu bukan kebetulan. Itu mencerminkan dinamika sosial di atas kapal saat bencana:

- **Sendirian:** Cepat tapi tidak diprioritaskan petugas.
- **Keluarga Kecil:** Diprioritaskan (terutama wanita/anak) dan gesit.
- **Keluarga Besar:** Lambat dan susah masuk sekoci barengan.

```
In [15]: family_map = {1: 'Alone', 2: 'Small', 3: 'Small', 4: 'Small', 5: 'Medium', 6: 'Medium', 7: 'Large', 8: 'Large', 9: 'Large'}
train_df['Family_Size_Grouped'] = train_df['Family_Size'].map(family_map)
test_df['Family_Size_Grouped'] = test_df['Family_Size'].map(family_map) #Kalau di groupby ini
```

```
In [16]: train_df.head(5)
```

Out[16]:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare
0	1	0	3	Braund, Mr. Owen Harris	male	22.00	1	0	A/5 21171	7.25
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th... Heikkinen, Miss. Laina	female	38.00	1	0	PC 17599	71.28
2	3	1	3	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	26.00	0	0	STON/O2. 3101282	7.92
3	4	1	1	Allen, Mr. William Henry	male	35.00	1	0	113803	53.10
4	5	0	3				0	0	373450	8.05

In [17]: `train_df.groupby(["Family_Size_Grouped"], as_index=False)[ "Survived" ].mean()`

Out[17]:

	Family_Size_Grouped	Survived
0	Alone	0.30
1	Large	0.16
2	Medium	0.16
3	Small	0.58

Catatan pribadi aja : Untuk kode diatas ini, video tutorial terinspirasi dari notebook lainnya. Benefit dari open competition, jika ada notebook orang lain, coba kamu baca dan cari apa sih perbedannya? dari situ akan muncul ide ide baru atau pemikiran baru yang mungkin kamu tidak lihat.

In [18]: `train_df.groupby(["Embarked"], as_index=False)[ "Survived" ].mean()`

Out[18]:

	Embarked	Survived
0	C	0.55
1	Q	0.39
2	S	0.34

Inget dengan hipotesis awal :

- **Embarked** : Pelabuhan Keberangkatan (Dari mana dia naik kapal?).
  - **C = Cherbourg** (Prancis). Biasanya banyak orang kaya naik dari sini.
  - **Q = Queenstown** (Irlandia). Biasanya banyak orang kelas 3 (imigran) naik dari sini.
  - **S = Southampton** (Inggris). Pelabuhan awal berangkat. Namun itu masih asumsi, perlu dianalisis lebih lanjut

Nah si video tutorial sendiri pun menyebutkan

and you can see C has a higher survival compared to Q which is about average and then S. So there is some things to think about on that side of things maybe C was like a more expensive city or more rich people ended up jumping on the ship from there. Q maybe middle class City. S maybe a lower class City don't know for sure but it is quite interesting.....

some things that i learned from doing this :

Mari kita bahas detailnya:

## 1. Family Size Grouping (Teknik Binning)

Ide untuk mengelompokkan **Family Size** menjadi **Alone, Small, Medium, Large** itu sangat brilian. Ini disebut teknik **Binning** atau **Discretization**.

- **Kenapa ini lebih baik daripada angka mentah (1, 2, 3... 11)?**
  - Bayangkan kamu jadi Mesin (Robot).
  - Kamu melihat angka **Family\_Size** : 1, 2, 3, 4, 5, 6, 7, 8, 11. (Ada 9 jenis angka).
  - Mesin harus belajar pola untuk setiap angka itu. "Kalau 5 gimana? Kalau 6 gimana?". Padahal sampelnya dikit banget.
  - Dengan **Grouping**, kita menyederhanakan tugas Mesin:
    - "Kalau Small (2,3,4) → Bagus!"
    - "Kalau Medium/Large/Alone → Kurang Bagus."
  - Ini membuat pola jadi lebih **TEGAS** dan mengurangi *noise*.
- **Hasil `groupby` kamu:**

- Small (2-4): **0.58 (58%)** → Tinggi!
- Alone (1): **0.30 (30%)** → Rendah.
- Medium/Large (>5): **0.16 (16%)** → Jelek Banget.
- **Kesimpulan:** Hipotesis kita valid. Punya keluarga kecil adalah kunci keselamatan.

## 2. Embarked (C, Q, S)

- **Data Kamu:**
  - C (Cherbourg): **0.55 (55%).**
  - Q (Queenstown): **0.39 (39%).**
  - S (Southampton): **0.34 (34%).**
- **Analisis Ryan vs Fakta Sejarah:**
  - Ryan: "Maybe C was like a more expensive city or more rich people..."
  - **Fakta:** Ryan benar! Cherbourg (Prancis) adalah perhentian pertama setelah Titanic berangkat dari Inggris. Di sini banyak penumpang **Kelas 1** (orang kaya Eropa/Amerika) yang naik.
  - Southampton (S) adalah tempat start awal, isinya campur aduk (banyak kru kapal dan kelas 3).
  - Queenstown (Q) adalah perhentian terakhir di Irlandia, mayoritas yang naik di sini adalah **Imigran Irlandia (Kelas 3)** yang mau mengadu nasib ke Amerika.
  - *Tapi kok Q (39%) lebih tinggi dari S (34%)?* Mungkin kebetulan, atau mungkin karakter orang Irlandia lebih tangguh/solidaritas tinggi? (Siapa tahu).

## 3. Koreksi Kodingan (Lagi)

Kamu benar, baris ini di tutorial mungkin typo atau kamu yang jeli melihat potensi bug:

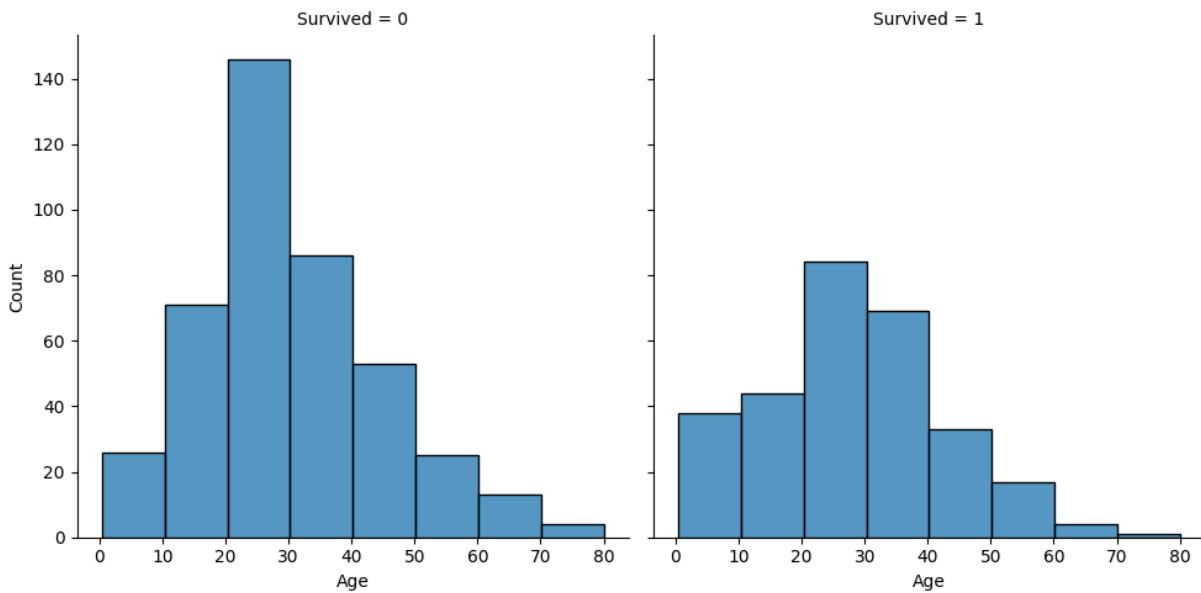
```
test_df['Family_Size_Grouped'] = test_df['Family_Size'].map(family_map)
# Pastikan pakai test_df di kiri dan kanan!
```

Kalau di tutorial dia nulis `train_df` di kanan padahal variabelnya `test_df`, itu **FATAL**.

Nanti data tes isinya malah data latihan. *Good job spotting that!*

```
In [19]: sns.displot(train_df, x='Age', col='Survived', binwidth=10, height=5) # Melihat Leb
```

```
Out[19]: <seaborn.axisgrid.FacetGrid at 0x1fdc04e6f60>
```



Kesimpulan Visual: Anak-anak (0-10 tahun) punya peluang selamat yang bagus. Pemuda (20-30 tahun) adalah korban terbanyak.

```
In [20]: train_df['Age_Cut'] = pd.qcut(train_df['Age'], 8)
test_df['Age_Cut'] = pd.qcut(test_df['Age'], 8)
```

## Teknik `pd.qcut` (Age Grouping)

Kode: `train_df['Age_Cut'] = pd.qcut(train_df['Age'], 8)`

### Apa itu `qcut` ?

- **Quantile Cut.**
- Kalau `bins` (histogram) membagi berdasarkan *Jarak Angka* (0-10, 10-20), `qcut` membagi berdasarkan **Jumlah Orang**.
- `qcut(..., 8)` artinya: "Bagi penumpang jadi 8 kelompok, di mana **SETIAP KELOMPOK ISINYA SAMA BANYAK.**"

**Kenapa pakai ini?** Supaya adil. Kita nggak mau bikin kelompok "Manula 80-90 tahun" kalau isinya cuma 1 orang. Dengan `qcut`, kelompok "Tua" mungkin rentangnya lebar (47-80 tahun) supaya jumlah orangnya setara dengan kelompok "Muda" (0-16 tahun).

```
In [21]: train_df.groupby(["Age_Cut"], as_index=False, observed = False)[["Survived"]].mean()
```

Out[21]:

	Age_Cut	Survived
0	(0.419, 16.0]	0.55
1	(16.0, 20.125]	0.34
2	(20.125, 24.0]	0.37
3	(24.0, 28.0]	0.35
4	(28.0, 32.312]	0.42
5	(32.312, 38.0]	0.45
6	(38.0, 47.0]	0.33
7	(47.0, 80.0]	0.42

## Analisis Tabel `groupby(['Age_Cut'])`

Mari kita lihat angka Survival Rate per kelompok umur:

Rentang Umur	Survival Rate	Analisis
0 - 16 thn	<b>0.55 (55%)</b>	TINGGI. Anak-anak diprioritaskan.
16 - 20 thn	0.34 (34%)	RENDAH. Remaja (mungkin dianggap dewasa muda).
20 - 24 thn	0.37 (37%)	RENDAH.
24 - 28 thn	0.35 (35%)	RENDAH. Ryan bilang: " <i>I would not have survived.</i> "
28 - 32 thn	<b>0.42 (42%)</b>	MENINGKAT. Usia matang?
32 - 38 thn	<b>0.45 (45%)</b>	MENINGKAT. Orang tua muda (mungkin karena bawa anak?).
38 - 47 thn	0.33 (33%)	TURUN LAGI.
47 - 80 thn	<b>0.42 (42%)</b>	NAIK LAGI. Orang tua dihormati? Atau orang tua kaya?

**Kesimpulan Strategis:** Umur itu **Bukan Garis Lurus**.

- Sangat Muda = Selamat.
- Muda (20-an) = Mati.
- Tua = Lumayan Selamat.

Pola naik-turun-naik ini penting buat Model Machine Learning. Kalau kita cuma kasih angka umur mentah (22, 23, 24), model mungkin bingung. Tapi kalau kita kasih label kelompok (0, 1, 2... 7), model lebih gampang nangkep polanya.

## Next step : iloc

Oke, mari kita *zoom out* sebentar biar kelihatan peta besarnya.

## 1. Masalah Sekarang

Saat ini, kolom `Age_Cut` isinya jelek banget buat komputer. Isinya teks panjang kayak gini: `(0.419, 16.0]`, `(16.0, 20.125]`, dst.

Komputer (Machine Learning) **BENCI** teks kayak gitu. Dia nggak bisa ngitung matematika pakai kurung siku dan koma.

## 2. Tujuan Ryan (dan Kita)

Kita ingin mengubah teks yang ribet itu menjadi **ANGKA SEDERHANA (KATEGORI)**.

- `(0.419, 16.0]` → Ubah jadi angka **0** (Anak Kecil).
- `(16.0, 20.125]` → Ubah jadi angka **1** (Remaja).
- ...
- `(47.0, 80.0]` → Ubah jadi angka **7** (Tua).

## 3. Kenapa Harus Diubah?

Supaya nanti kolom `Age` di data kita isinya bukan lagi umur acak (22, 38, 26, 35...), tapi menjadi **Kelompok Umur** (0, 1, 2, 3...).

**Manfaatnya:** Ini membantu model Machine Learning untuk **Menggeneralisasi**.

- Model gak perlu pusing mikirin bedanya umur 21 sama 22.
- Model cukup tahu: "Oh, mereka berdua sama-sama Kelompok 1 (Remaja), nasibnya mirip (Rentan Mati)."

## 4. Cara Melakukannya (Next Step)

Ryan akan menggunakan `loc` (atau `iloc`) untuk melakukan operasi "Find and Replace" massal.

"*Eh Pandas, cari semua baris yang umurnya di bawah 16 tahun, terus ganti nilai kolom 'Age'-nya jadi 0.*" "*Cari yang umurnya 16-20, ganti jadi 1.*" Dst...

Jadi, langkah selanjutnya adalah **TRANSFORMASI DATA**: Mengubah Umur (Angka Kontinu) menjadi Kelompok Umur (Angka Kategori 0-7).

```
In [22]: train_df.loc[train_df['Age'] <= 16, 'Age'] = 0
train_df.loc[(train_df['Age'] > 16) & (train_df['Age'] <= 20.125), 'Age'] = 1
train_df.loc[(train_df['Age'] > 20.125) & (train_df['Age'] <= 24), 'Age'] = 2
train_df.loc[(train_df['Age'] > 24) & (train_df['Age'] <= 28), 'Age'] = 3
train_df.loc[(train_df['Age'] > 28) & (train_df['Age'] <= 32.312), 'Age'] = 4
train_df.loc[(train_df['Age'] > 32.312) & (train_df['Age'] <= 38), 'Age'] = 5
train_df.loc[(train_df['Age'] > 38) & (train_df['Age'] <= 47), 'Age'] = 6
train_df.loc[(train_df['Age'] > 47) & (train_df['Age'] <= 80), 'Age'] = 7
train_df.loc[train_df['Age'] > 80, 'Age'] = 8
```

```
test_df.loc[test_df['Age'] <= 16, 'Age'] = 0
test_df.loc[(test_df['Age'] > 16) & (test_df['Age'] <= 20.125), 'Age'] = 1
test_df.loc[(test_df['Age'] > 20.125) & (test_df['Age'] <= 24), 'Age'] = 2
test_df.loc[(test_df['Age'] > 24) & (test_df['Age'] <= 28), 'Age'] = 3
test_df.loc[(test_df['Age'] > 28) & (test_df['Age'] <= 32.312), 'Age'] = 4
test_df.loc[(test_df['Age'] > 32.312) & (test_df['Age'] <= 38), 'Age'] = 5
test_df.loc[(test_df['Age'] > 38) & (test_df['Age'] <= 47), 'Age'] = 6
test_df.loc[(test_df['Age'] > 47) & (test_df['Age'] <= 80), 'Age'] = 7
test_df.loc[test_df['Age'] > 80, 'Age'] = 8
```

## 1. Kenapa `loc` bukan `iloc` ?

Ini konsep dasar Pandas yang sering ketukar:

- **`iloc` (Index Location):** Cari berdasarkan **Urutan Baris/Kolom (Angka)**.
  - Contoh: `iloc[0, 1]` → Ambil baris pertama, kolom kedua.
  - **Kelemahan:** Dia nggak ngerti nama kolom "Age" atau kondisi logika (`Age > 16`).  
Dia cuma ngerti urutan angka.
- **`loc` (Location):** Cari berdasarkan **Label Nama Kolom** atau **Kondisi Logika**.
  - Contoh: `loc[df['Age'] > 16, 'Age']` → Cari baris yang umurnya > 16, lalu fokus ke kolom 'Age'.
  - **Ini yang kita butuhkan!** Kita mau cari baris berdasarkan *kondisi*, bukan urutan.

Makanya Ryan error pas pakai `iloc` dan akhirnya ganti ke `loc`. Kode kamu di atas sudah pakai `loc`, jadi sudah **BENAR dan AMAN**.

---

## 2. Soal "Bisa Pakai Integer"

Ryan bilang: "*I know I could technically just put full numbers on here but whatever... because the ages are just going to be integers*".

Maksudnya gini: Kamu lihat kode kamu: `train_df.loc[(train_df['Age'] > 20.125)`  
`...` Angka **20.125** itu pecahan (desimal).

Ryan bilang, sebenarnya bisa aja dibuletin jadi **20** atau **21**. Kenapa? Karena umur manusia di data Titanic itu mayoritas bilangan bulat (20 tahun, 21 tahun). Jarang ada yang nulis "Umur saya 20.125 tahun". (Kecuali bayi, biasanya ditulis 0.5 tahun).

Tapi Ryan tetap pakai angka desimal (20.125) karena dia **copy-paste persis** dari hasil output `pd.qcut` tadi biar akurat 100% sama pembagian kelompoknya.

**Kesimpulan:** Pakai angka desimal (20.125) itu lebih presisi dan konsisten dengan `qcut`. Jadi langkah kamu (dan Ryan) sudah benar.

---

## 3. Masalah "Copy-Paste Code" yang Panjang

Kamu lihat kan kodennya panjang banget sampai 16 baris (8 baris buat train, 8 baris buat test)? Ryan sendiri sadar itu *painful*: "*I could have technically combined train and then also test early on... but visually for myself I just want to see how both of these are working*".

- **Cara Ryan (Manual):** Nulis satu-satu buat `train`, terus nulis ulang buat `test`.  
Rawan salah ketik (typo).
- **Cara Pro (Best Practice):** Harusnya digabung dulu jadi satu dataset `all_data`, diproses bareng, baru dipisah lagi. ATAU bikin fungsi `def categorize_age(df): ...` lalu panggil fungsinya dua kali.

Tapi karena ini tutorial pemula, Ryan memilih cara manual biar penonton **paham logika perubahannya step-by-step**.

---

## 4. Hasil Akhir (Apa yang berubah?)

Coba lihat output `train_df.head()` kamu:

- **Dulu:** Kolom `Age` isinya 22.0, 38.0, 26.0 (Angka umur asli).
- **Sekarang:** Kolom `Age` isinya **2.0, 5.0, 3.0** (Angka kategori/kelompok).

**Artinya:**

- Si Braund (22 tahun) → Masuk Kelompok 2.
- Si Cumings (38 tahun) → Masuk Kelompok 5.

In [23]: `train_df.head()`

Out[23]:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare
0	1	0	3	Braund, Mr. Owen Harris	male	2.00	1	0	A/5 21171	7.25
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th... Heikkinen, Miss. Laina	female	5.00	1	0	PC 17599	71.28
2	3	1	3	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	3.00	0	0	STON/O2. 3101282	7.92
3	4	1	1	Allen, Mr. William Henry	male	5.00	1	0	113803	53.10
4	5	0	3				0	0	373450	8.05

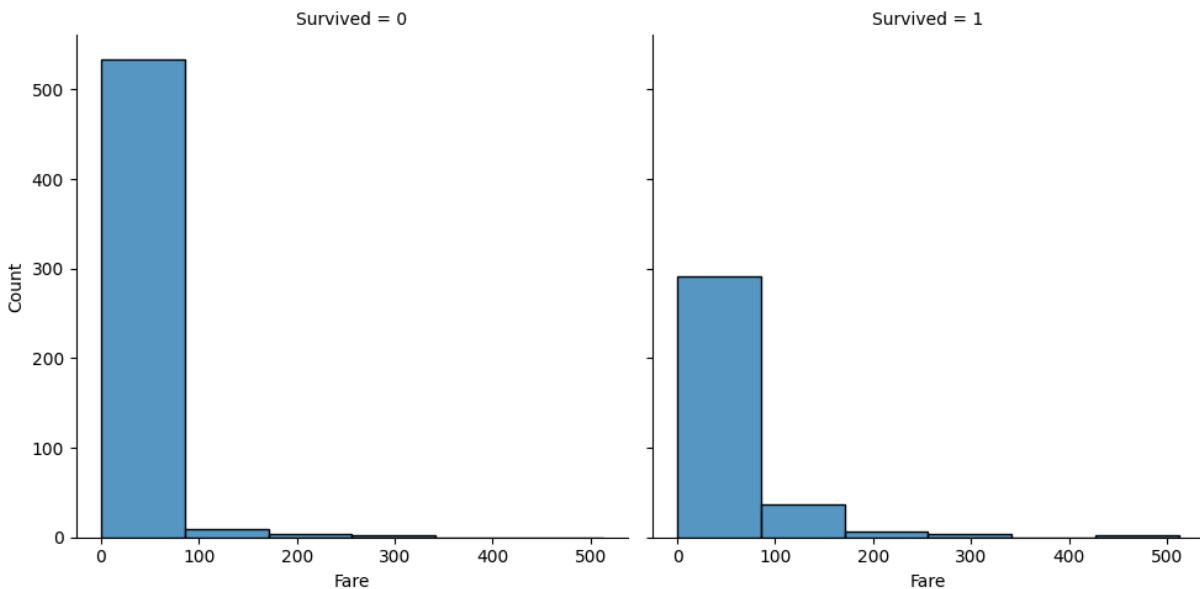
In [24]: `train_df.groupby(["Age"], as_index=False)[["Survived"]].mean()`

Out[24]:

	Age	Survived
0	0.00	0.55
1	1.00	0.34
2	2.00	0.37
3	3.00	0.35
4	4.00	0.42
5	5.00	0.45
6	6.00	0.33
7	7.00	0.42

In [25]: `sns.displot(train_df, x='Fare', col='Survived', binwidth=80, height=5)`

Out[25]: &lt;seaborn.axisgrid.FacetGrid at 0x1fdc0cb6c60&gt;



```
In [26]: # Cara Modern (Tanpa inplace=True)
test_df['Fare'] = test_df['Fare'].fillna(test_df['Fare'].mean())
```

Kode diatas ditambahkan karena di tutorial Ryan, untuk saat ini belum menanggapi kasus ini. Kasus ini pun ditemukan pada saat saya mencoba untuk running lagi mengenai df test dan train info(). Dan menemukan jebakan di df\_test bahwasannya ada yang bolong satu data.

```
In [27]: train_df['Fare_Cut'] = pd.qcut(train_df['Fare'], 6)
test_df['Fare_Cut'] = pd.qcut(test_df['Fare'], 6)
```

Catatan : Discretization / Binning pada data umur dan fare bisa diotak atik lagi untuk menyempurnakan model.

```
In [28]: train_df.groupby(['Fare_Cut'], as_index=False, observed = False)[["Survived"]].mean()
```

Out[28]:

	Fare_Cut	Survived
<b>0</b>	(-0.001, 7.775]	0.21
<b>1</b>	(7.775, 8.662]	0.19
<b>2</b>	(8.662, 14.454]	0.37
<b>3</b>	(14.454, 26.0]	0.44
<b>4</b>	(26.0, 52.369]	0.42
<b>5</b>	(52.369, 512.329]	0.70

```
In [29]: train_df.loc[train_df['Fare'] <= 7.775, 'Fare'] = 0
train_df.loc[(train_df['Fare'] > 7.775) & (train_df['Fare'] <= 8.662), 'Fare'] = 1
```

```

train_df.loc[(train_df['Fare'] > 8.662) & (train_df['Fare'] <= 14.454), 'Fare'] = 2
train_df.loc[(train_df['Fare'] > 14.454) & (train_df['Fare'] <= 26.0), 'Fare'] = 3
train_df.loc[(train_df['Fare'] > 26.0) & (train_df['Fare'] <= 52.369), 'Fare'] = 4
train_df.loc[(train_df['Fare'] > 52.369) & (train_df['Fare'] <= 512.329), 'Fare'] = 5
train_df.loc[train_df['Fare'] > 512.329, 'Fare']

test_df.loc[test_df['Fare'] <= 7.775, 'Fare'] = 0
test_df.loc[(test_df['Fare'] > 7.775) & (test_df['Fare'] <= 8.662), 'Fare'] = 1
test_df.loc[(test_df['Fare'] > 8.662) & (test_df['Fare'] <= 14.454), 'Fare'] = 2
test_df.loc[(test_df['Fare'] > 14.454) & (test_df['Fare'] <= 26.0), 'Fare'] = 3
test_df.loc[(test_df['Fare'] > 26.0) & (test_df['Fare'] <= 52.369), 'Fare'] = 4
test_df.loc[(test_df['Fare'] > 52.369) & (test_df['Fare'] <= 512.329), 'Fare'] = 5
test_df.loc[test_df['Fare'] > 512.329, 'Fare']

```

Out[29]: 343 512.33  
Name: Fare, dtype: float64

In [30]: train\_df.head(5)

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare
0	1	0	3	Braund, Mr. Owen Harris	male	2.00	1	0	A/5 21171	0.00
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...)	female	5.00	1	0	PC 17599	5.00
2	3	1	3	Heikkinen, Miss. Laina	female	3.00	0	0	STON/O2. 3101282	1.00
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	5.00	1	0	113803	5.00
4	5	0	3	Allen, Mr. William Henry	male	5.00	0	0	373450	1.00



In [31]: train\_df.info()

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 891 entries, 0 to 890
Data columns (total 16 columns):
 #   Column           Non-Null Count  Dtype  
--- 
 0   PassengerId      891 non-null    int64  
 1   Survived         891 non-null    int64  
 2   Pclass            891 non-null    int64  
 3   Name              891 non-null    object  
 4   Sex               891 non-null    object  
 5   Age               714 non-null    float64 
 6   SibSp            891 non-null    int64  
 7   Parch             891 non-null    int64  
 8   Ticket            891 non-null    object  
 9   Fare              891 non-null    float64 
 10  Cabin             204 non-null    object  
 11  Embarked          889 non-null    object  
 12  Family_Size       891 non-null    int64  
 13  Family_Size_Grouped 891 non-null    object  
 14  Age_Cut           714 non-null    category 
 15  Fare_Cut          891 non-null    category 
dtypes: category(2), float64(2), int64(6), object(6)
memory usage: 100.0+ KB
```

In [32]: `test_df.head(5)`

	PassengerId	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Emba
0	892	3	Kelly, Mr. James	male	5.00	0	0	330911	1.00	NaN	
1	893	3	Wilkes, Mrs. James (Ellen Needs)	female	6.00	1	0	363272	0.00	NaN	
2	894	2	Myles, Mr. Thomas Francis	male	7.00	0	0	240276	2.00	NaN	
3	895	3	Wirz, Mr. Albert	male	3.00	0	0	315154	2.00	NaN	
4	896	3	Hirvonen, Mrs. Alexander (Helga E Lindqvist)	female	2.00	1	1	3101298	2.00	NaN	

In [33]: `test_df.info()`

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 418 entries, 0 to 417
Data columns (total 15 columns):
 #   Column           Non-Null Count  Dtype  
--- 
 0   PassengerId      418 non-null    int64  
 1   Pclass            418 non-null    int64  
 2   Name              418 non-null    object  
 3   Sex               418 non-null    object  
 4   Age               332 non-null    float64 
 5   SibSp            418 non-null    int64  
 6   Parch            418 non-null    int64  
 7   Ticket            418 non-null    object  
 8   Fare              418 non-null    float64 
 9   Cabin             91 non-null     object  
 10  Embarked          418 non-null    object  
 11  Family_Size       418 non-null    int64  
 12  Family_Size_Grouped 418 non-null    object  
 13  Age_Cut           332 non-null    category 
 14  Fare_Cut          418 non-null    category 
dtypes: category(2), float64(2), int64(5), object(6)
memory usage: 44.1+ KB
```

In [34]: `train_df['Name']`

```
Out[34]: 0                  Braund, Mr. Owen Harris
1      Cumings, Mrs. John Bradley (Florence Briggs Th...
2                      Heikkinen, Miss. Laina
3      Futrelle, Mrs. Jacques Heath (Lily May Peel)
4                  Allen, Mr. William Henry
...
886                  Montvila, Rev. Juozas
887                  Graham, Miss. Margaret Edith
888      Johnston, Miss. Catherine Helen "Carrie"
889                  Behr, Mr. Karl Howell
890                  Dooley, Mr. Patrick
Name: Name, Length: 891, dtype: object
```

In [35]: `train_df['Name'].str.split(pat= ",", expand=True) # Seperti yang dapat dilihat, bah`

Out[35]:

	0	1
0	Braund	Mr. Owen Harris
1	Cumings	Mrs. John Bradley (Florence Briggs Thayer)
2	Heikkinen	Miss. Laina
3	Futrelle	Mrs. Jacques Heath (Lily May Peel)
4	Allen	Mr. William Henry
...	...	...
886	Montvila	Rev. Juozas
887	Graham	Miss. Margaret Edith
888	Johnston	Miss. Catherine Helen "Carrie"
889	Behr	Mr. Karl Howell
890	Dooley	Mr. Patrick

891 rows × 2 columns

In [36]: `train_df['Name'].str.split(pat= ",", expand=True)[1] #Enggak peduli dengan last name`

Out[36]:

0	Mr. Owen Harris
1	Mrs. John Bradley (Florence Briggs Thayer)
2	Miss. Laina
3	Mrs. Jacques Heath (Lily May Peel)
4	Mr. William Henry
...	...
886	Rev. Juozas
887	Miss. Margaret Edith
888	Miss. Catherine Helen "Carrie"
889	Mr. Karl Howell
890	Mr. Patrick

Name: 1, Length: 891, dtype: object

In [37]: `train_df['Name'].str.split(pat= ",", expand=True)[1].str.split(pat= ".", expand=True)[1]`

Out[37]:

	0	1	2
0	Mr	Owen Harris	None
1	Mrs	John Bradley (Florence Briggs Thayer)	None
2	Miss	Laina	None
3	Mrs	Jacques Heath (Lily May Peel)	None
4	Mr	William Henry	None
...	...	...	...
886	Rev	Juozas	None
887	Miss	Margaret Edith	None
888	Miss	Catherine Helen "Carrie"	None
889	Mr	Karl Howell	None
890	Mr	Patrick	None

891 rows × 3 columns

In [38]: `train_df['Name'].str.split(pat= ",", expand=True)[1].str.split(pat= ".", expand=True)`

Out[38]:

0	Mr
1	Mrs
2	Miss
3	Mrs
4	Mr
...	...
886	Rev
887	Miss
888	Miss
889	Mr
890	Mr

Name: 0, Length: 891, dtype: object

In [39]: `train_df['Title'] = train_df['Name'].str.split(pat= ",", expand=True)[1].str.split(test_df['Title'] = test_df['Name'].str.split(pat= ",", expand=True)[1].str.split(pat= ".", expand=True))`

In [40]: `train_df.head()`

Out[40]:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin
0	1	0	3	Braund, Mr. Owen Harris	male	2.00	1	0	A/5 21171	0.00	
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...)	female	5.00	1	0	PC 17599	5.00	
2	3	1	3	Heikkinen, Miss. Laina	female	3.00	0	0	STON/O2. 3101282	1.00	
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	5.00	1	0	113803	5.00	
4	5	0	3	Allen, Mr. William Henry	male	5.00	0	0	373450	1.00	



String split berhasil

In [41]: `train_df.groupby(["Title"], as_index=False)[["Survived"]].mean()`

Out[41]:

	Title	Survived
0	Capt	0.00
1	Col	0.50
2	Don	0.00
3	Dr	0.43
4	Jonkheer	0.00
5	Lady	1.00
6	Major	0.50
7	Master	0.57
8	Miss	0.70
9	Mlle	1.00
10	Mme	1.00
11	Mr	0.16
12	Mrs	0.79
13	Ms	1.00
14	Rev	0.00
15	Sir	1.00
16	the Countess	1.00

In [42]: `train_df['Title'].value_counts() # Kode ini baru ditambahkan setelah penjelasan vid`

Out[42]: Title

Mr	517
Miss	182
Mrs	125
Master	40
Dr	7
Rev	6
Col	2
Mlle	2
Major	2
Ms	1
Mme	1
Don	1
Lady	1
Sir	1
Capt	1
the Countess	1
Jonkheer	1
Name: count, dtype: int64	

Mari kita bahas poin-poin penting dari kode **Title Extraction** ini, karena ini adalah teknik *text processing* yang sangat berguna.

## 1. Logika Pemecahan Nama (The Anatomy of Name)

Nama di Titanic formatnya baku: `Nama Belakang, Gelar. Nama Depan` Contoh:  
`Braund, Mr. Owen Harris`

### Strategi Potong-Memotong:

#### 1. Split Pertama (Koma ,):

- `Braund` (Index 0 - Nama Keluarga)
- `Mr. Owen Harris` (Index 1 - Sisa) → **Kita ambil yang ini.**

#### 2. Split Kedua (Titik .):

- `Mr` (Index 0 - Gelar) → **Kita ambil yang ini.**
- `Owen Harris` (Index 1 - Nama Depan).

#### 3. `strip()` (Pembersihan Spasi):

- Hasil split seringkali ada spasi nyangkut, misal " Mr".
- `strip()` membuang spasi di awal/akhir jadi bersih "Mr".

## 2. Analisis Hasil `groupby(['Title'])` (Ini Menarik!)

Coba lihat tabel Survival Rate berdasarkan Title:

- **Wanita Bangsawan (Lady, Mlle, Mme, Ms, Countess): 1.00 (100% Selamat!).**
  - Ini bukti nyata bahwa wanita kelas atas sangat diprioritaskan.
- **Mrs (Ibu-ibu): 0.79 (79%).** Tinggi banget.
- **Miss (Nona): 0.70 (70%).** Tinggi juga.
- **Master (Anak Laki-laki): 0.57 (57%).**
  - Ini menarik! Padahal rata-rata "Male" cuma 19%. Tapi kalau gelarnya "Master" (artinya dia masih bocil), peluang selamatnya naik drastis jadi 57%. Bukti "Children First".
- **Mr (Laki-laki Dewasa): 0.16 (16%).**
  - Kasihan sekali para Bapak-bapak/Pemuda. Peluang hidupnya paling kecil.
- **Rev (Pendeta): 0.00 (0%).**
  - Mereka memilih tenggelam sambil berdoa bersama jemaatnya. (Fakta sejarah yang sedih tapi heroik).

## 3. Masalah Baru: "Terlalu Banyak Gelar" 😱

Kamu lihat kan ada banyak gelar aneh-aneh yang jumlahnya cuma 1 atau 2 orang?

- `Capt, Col, Don, Jonkheer, Major, Sir ...`

Kalau kita biarkan begini, nanti model Machine Learning bakal bingung karena sampelnya terlalu sedikit (*Sparse Data*).

**Solusi Selanjutnya (Prediksi Saya):** Ryan pasti akan melakukan **Grouping Title**. Dia akan menyatukan gelar-gelar langka itu menjadi satu kelompok besar, misalnya "**Military**" atau "**Noble**" atau sekadar "**Misc**".

Dan dia juga mungkin akan menyatukan gelar Prancis ( `Mlle` , `Mme` ) ke gelar Inggris ( `Miss` , `Mrs` ).

Yang dilakukan oleh Ryan untuk pengelompokan :

- military - Capt, Col, Major
- noble - Jonkheer, the Countess, Don, Lady, Sir
- unmarried Female - Mlle, Ms, Mme

```
In [43]: train_df['Title'] = train_df['Title'].replace({
    'Capt': 'Military',
    'Col': 'Military',
    'Major': 'Military',
    'Jonkheer': 'Noble',
    'the Countess': 'Noble',
    'Don': 'Noble',
    'Lady': 'Noble',
    'Sir': 'Noble',
    'Mlle': 'Noble',
    'Ms': 'Noble',
    'Mme': 'Noble'
})

test_df['Title'] = test_df['Title'].replace({
    'Capt': 'Military',
    'Col': 'Military',
    'Major': 'Military',
    'Jonkheer': 'Noble',
    'the Countess': 'Noble',
    'Don': 'Noble',
    'Lady': 'Noble',
    'Sir': 'Noble',
    'Mlle': 'Noble',
    'Ms': 'Noble',
    'Mme': 'Noble'
})
```

```
In [44]: train_df.groupby(["Title"], as_index=False)[["Survived"]].agg([ 'count', 'mean'])
```

Out[44]:

	Title	count	mean
0	Dr	7	0.43
1	Master	40	0.57
2	Military	5	0.40
3	Miss	182	0.70
4	Mr	517	0.16
5	Mrs	125	0.79
6	Noble	9	0.78
7	Rev	6	0.00

Dari sini bisa diambil beberapa asumsi mendasar :

- **Dr** Disini diatas rata-rata, mungkin ada yang berusaha menyelempatkan penumpang, ada juga yang dibutuhkan untuk di koci penyelamat
- **Master** Master adalah gelar kehormatan dalam bahasa Inggris untuk anak laki-laki dan pemuda. Dapat dilihat nilainya lumayan tinggi
- **Military** orang yang bekerja di militer. Agak gak bisa dimabil kesimpulan karena disatu sisi juga jumlahnya hanya 5 orang (2 orang selamat)
- **Miss , Mrs** Mrs. (dibaca "misses") untuk wanita yang sudah menikah, Miss untuk wanita yang belum menikah (biasanya muda). Mereka berduaa adalah sebutan wanita. Sesuai asumsi awal benar bahwa perempuan dan anak-anak didahulukan.
- **Mr** seorang pria, yang banyak tidak survived
- **Rev** "Rev" dalam bahasa Inggris memiliki beberapa arti utama yaitu singkatan dari Reverend (Pendeta) untuk rohaniwan Kristen. Kemungkinan para pendeta memutuskan untuk doa bersama kepada penumpang yang tidak mendapatkan koci (tidak ada yang sama sekali selamat)
- **Noble** sebutan pada orang yang memiliki gelar "Mulia".

Koreksi Kecil Pengelompokan Ryan: Ryan memasukkan Ms, Mlle, Mme ke Noble? Sebenarnya Mlle (Mademoiselle) = Miss (Perancis). Mme (Madame) = Mrs (Perancis). Ms = Miss. Tapi nggak apa-apa, karena jumlahnya sedikit (cuma 1-2), dimasukkan ke Noble pun nggak terlalu merusak data. Yang penting mereka "Wanita Terhormat".

## Next Step: Name Length (Panjang Nama)

Ini ide yang unik dari Ryan:

"*Someone has a longer name they're a little bit more important.*"

**Hipotesis:** Orang kaya/bangsawan jaman dulu namanya panjang-panjang.

- *Contoh:* "Mrs. John Bradley (Florence Briggs Thayer)" (Panjang) → Kaya.
- *Contoh:* "Mr. John Smith" (Pendek) → Biasa.

**Cara Mengeceknya:** Kita harus menghitung **jumlah karakter** dalam kolom `Name`.

Kode yang akan kamu tulis nanti (kira-kira seperti ini):

```
train_df['Name_Length'] = train_df['Name'].apply(lambda x: len(x))
test_df['Name_Length'] = test_df['Name'].apply(lambda x: len(x))
```

Setelah itu, kamu bisa pakai `qcut` lagi (seperti `Age` dan `Fare`) untuk mengelompokkan:

- Nama Pendek
- Nama Sedang
- Nama Panjang

Coba lihat, apakah orang yang namanya panjang beneran lebih banyak yang selamat? 😊

```
In [45]: train_df['Name_Length'] = train_df['Name'].apply(lambda x: len(x))
test_df['Name_Length'] = test_df['Name'].apply(lambda x: len(x))
```

Belajar kodingan syntax Kode target:

---

```
train_df['Name_Length'] = train_df['Name'].apply(lambda x: len(x))
```

---

## Analogi Koki di Dapur

Bayangkan `train_df['Name']` adalah sebuah **Keranjang** berisi ribuan kentang (Nama Penumpang). Kamu ingin tahu **Berat (Panjang Huruf)** dari setiap kentang itu.

1. `train_df['Name']` : Ini keranjang kentangnya.
  2. `.apply(...)` : Ini perintah: "*Woy, tolong TERAPKAN alat ukur ini ke SETIAP kentang satu per satu!*"
  3. `lambda x: len(x)` : Ini adalah **Alat Ukurnya** (Meteran).
- 

## Bedah `lambda x: len(x)` (Si Alat Ukur)

Ini adalah cara cepat (shortcut) nulis fungsi di Python.

- `lambda` : Kata kunci buat bilang "Eh, aku mau bikin fungsi sekali pakai nih."
- `x` : Ini adalah "Barang yang lagi dipegang".

- Pas giliran pertama, `x` = "Braund, Mr. Owen Harris".
- Pas giliran kedua, `x` = "Heikkinen, Miss. Laina".
- `len(x)` : Ini **Aksinya**. Hitung panjang karakter (Length) dari si `x`.

**Kalau ditulis versi panjang (Tanpa Lambda), jadinya gini:**

```
def hitung_panjang_nama(nama_penumpang):
    return len(nama_penumpang)

# Cara pakai:
train_df['Name'].apply(hitung_panjang_nama)
(Sama aja kan? Cuma Lambda lebih ringkas).
```

---

## Simulasi Proses (Step-by-Step)

### Data Awal:

1. "Ali"
2. "Budi Santoso"

Proses `.apply(lambda x: len(x))`:

1. **Ambil Data 1:** `x` = "Ali".
  - Hitung `len("Ali")` → 3.
  - Simpan angka 3.
2. **Ambil Data 2:** `x` = "Budi Santoso".
  - Hitung `len("Budi Santoso")` → 12 (Spasi dihitung).
  - Simpan angka 12.

**Hasil Akhir (`train_df['Name_Length']`):**

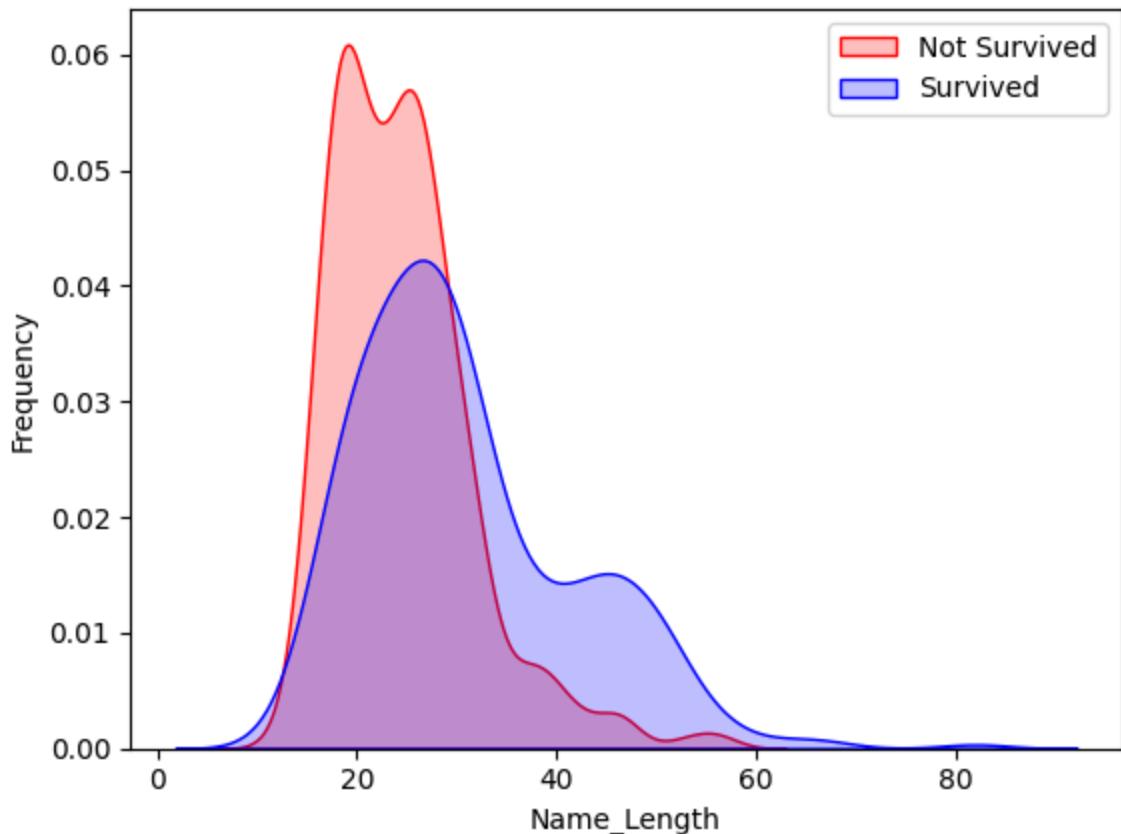
Name	Name_Length
Ali	3
Budi Santoso	12

Jadi `apply` itu kayak mandor yang nyuruh robot (`lambda`) buat ngukur satu-satu. 🤖

In [46]: `train_df.info()`

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 891 entries, 0 to 890
Data columns (total 18 columns):
 #   Column           Non-Null Count  Dtype  
--- 
 0   PassengerId      891 non-null    int64  
 1   Survived         891 non-null    int64  
 2   Pclass            891 non-null    int64  
 3   Name              891 non-null    object  
 4   Sex               891 non-null    object  
 5   Age               714 non-null    float64 
 6   SibSp            891 non-null    int64  
 7   Parch            891 non-null    int64  
 8   Ticket            891 non-null    object  
 9   Fare              891 non-null    float64 
 10  Cabin             204 non-null    object  
 11  Embarked          889 non-null    object  
 12  Family_Size       891 non-null    int64  
 13  Family_Size_Grouped 891 non-null    object  
 14  Age_Cut           714 non-null    category 
 15  Fare_Cut          891 non-null    category 
 16  Title              891 non-null    object  
 17  Name_Length        891 non-null    int64  
dtypes: category(2), float64(2), int64(7), object(7)
memory usage: 113.9+ KB
```

```
In [47]: g = sns.kdeplot(train_df['Name_Length'][train_df['Survived']==0] & (train_df['Name_Length']<=10))
g = sns.kdeplot(train_df['Name_Length'][train_df['Survived']==1] & (train_df['Name_Length']>10))
g.set_xlabel('Name_Length')
g.set_ylabel('Frequency')
g = g.legend(['Not Survived', 'Survived'])
```



```
In [48]: train_df.groupby(["Name_Length"], as_index=False)[["Survived"]].mean()
```

Out[48]:

	Name_Length	Survived
<b>0</b>	12	0.50
<b>1</b>	13	0.50
<b>2</b>	14	0.33
<b>3</b>	15	0.13
<b>4</b>	16	0.23
<b>5</b>	17	0.21
<b>6</b>	18	0.20
<b>7</b>	19	0.23
<b>8</b>	20	0.28
<b>9</b>	21	0.33
<b>10</b>	22	0.32
<b>11</b>	23	0.28
<b>12</b>	24	0.37
<b>13</b>	25	0.33
<b>14</b>	26	0.22
<b>15</b>	27	0.36
<b>16</b>	28	0.37
<b>17</b>	29	0.50
<b>18</b>	30	0.43
<b>19</b>	31	0.40
<b>20</b>	32	0.57
<b>21</b>	33	0.55
<b>22</b>	34	0.43
<b>23</b>	35	1.00
<b>24</b>	36	0.33
<b>25</b>	37	0.70
<b>26</b>	38	0.44
<b>27</b>	39	0.44
<b>28</b>	40	0.43
<b>29</b>	41	1.00

	Name_Length	Survived
30	42	0.20
31	43	0.80
32	44	1.00
33	45	0.78
34	46	0.57
35	47	0.73
36	48	1.00
37	49	1.00
38	50	1.00
39	51	1.00
40	52	0.75
41	53	1.00
42	54	0.00
43	55	0.50
44	56	0.67
45	57	0.50
46	61	1.00
47	65	1.00
48	67	1.00
49	82	1.00

Pendapatku jujur sebelum aku coba bertanya ke ai adalah, saya sama sekali gak melihat pola tertentu dan alasan kenapa kita melakukan ini. Setelah saya pelajari sepertinya memang saya perlu lebih telaah dan teliti lagi dalam hal hal ginian seperti menafsirkan grafik, mengambil tindakan dan kesmpulan serta hal lainnya.

Oke, mari kita bedah kebingunganmu. Kamu merasa: "*Kok kayaknya panjang pendek sama aja sih?*"

Sebenarnya, kalau kamu jeli melihat Grafik `kdeplot` (Gunung Merah vs Gunung Biru) dan Tabel `groupby`, ada pola yang sangat kuat lho!

## 1. Kenapa Panjang Nama Itu Penting? (Logika Sejarah)

Ryan bilang: "Name length a little bit more importance especially in the early 1900s."

### Fakta Zaman Dulu:

- **Orang Biasa (Miskin/Kelas 3):** Namanya pendek. "Mr. John Smith". (Cuma Nama Depan + Belakang).
- **Orang Kaya (Bangsawan/Kelas 1):** Namanya puanjaang banget. Punya nama tengah banyak, nama keluarga istri, gelar bangsawan.
  - Contoh di data kamu: Cumings, Mrs. John Bradley (Florence Briggs Thayer) . (Ini panjang banget kan?).

**Hipotesis:** Nama Panjang = Orang Kaya = Prioritas Selamat.

## 2. Analisis Grafik Gunung (KDE Plot)

Lihat gambar grafik yang kamu hasilkan:

- **Gunung Merah (Not Survived):** Puncaknya ada di **Kiri** (Angka kecil, sekitar 20-30 karakter). Artinya yang mati rata-rata namanya pendek.
- **Gunung Biru (Survived):** Puncaknya lebih landai dan **Bergeser ke Kanan** (Angka besar, ada tonjolan di angka 40-50). Artinya yang selamat cenderung punya nama lebih panjang.

## 3. Analisis Tabel Angka (groupby )

Coba lihat tabel angka yang kamu hasilkan tadi.

- **Nama Pendek (15-25 huruf):** Survival Rate-nya rata-rata **0.20 - 0.30** (Rendah).
- **Nama Panjang (45-55 huruf):**
  - 48 huruf: **1.00** (Selamat semua!)
  - 49 huruf: **1.00**
  - 50 huruf: **1.00**
  - 51 huruf: **1.00**
- **Nama Super Panjang (82 huruf):** **1.00** (Selamat!).

**Kesimpulan:** Ternyata benar! Orang yang namanya panjang banget (biasanya ibu-ibu pejabat yang nulis nama suami + nama gadis + gelar) hampir semuanya selamat. Sedangkan yang namanya pendek (bapak-bapak buruh migran) banyak yang tewas.

Jadi, `Name_Length` ini **BUKAN** fitur sampah. Ini fitur rahasia yang kuat!

## 4. Kenapa Kode Grafiknya Ribet (notnull )?

Kode Ryan:

```
train_df['Name_Length'][(train_df['Survived']==0) &
                        (train_df['Name_Length'].notnull())]
```

**Kenapa harus `notnull()`?** Sebenarnya di data kita `Name` itu tidak ada yang kosong (891 non-null). Jadi `Name_Length` juga pasti ada isinya semua. Ryan pakai `notnull()` cuma buat **jaga-jaga (defensive)** lagi. Di library grafik (Seaborn/Matplotlib), kalau ada satu aja data `NaN` nyelip di kolom yang mau digambar, kadang grafiknya bisa error atau kosong.

Jadi itu cuma kebiasaan *safety* aja, bukan karena datanya bolong.

---

**Next Step:** Sama kayak `Age` dan `Fare`, angka panjang nama ini (12, 13, ... 82) terlalu banyak variasinya. Langkah selanjutnya Ryan pasti akan melakukan **Binning (Qcut)** lagi untuk mengelompokkan jadi:

- Nama Pendek (Level 0)
- Nama Sedang (Level 1)
- Nama Panjang (Level 2)

```
In [49]: train_df['Name_LengthGB'] = pd.qcut(train_df['Name_Length'], 3)
test_df['Name_LengthGB'] = pd.qcut(test_df['Name_Length'], 3)
```

```
In [50]: train_df.groupby(['Name_LengthGB'], as_index=False, observed=False)[['Survived']].mean()
```

	Name_LengthGB	Survived
0	(11.999, 22.0]	0.25
1	(22.0, 28.0]	0.32
2	(28.0, 82.0]	0.59

```
In [51]: train_df.loc[train_df['Name_Length'] <= 22, 'Name_Size'] = 0
train_df.loc[(train_df['Name_Length'] > 22) & (train_df['Name_Length'] <= 28), 'Name_Size'] = 1
train_df.loc[(train_df['Name_Length'] > 28) & (train_df['Name_Length'] <= 82), 'Name_Size'] = 2
train_df.loc[train_df['Name_Length'] > 82, 'Name_Size'] = 3

test_df.loc[test_df['Name_Length'] <= 22, 'Name_Size'] = 0
test_df.loc[(test_df['Name_Length'] > 22) & (test_df['Name_Length'] <= 28), 'Name_Size'] = 1
test_df.loc[(test_df['Name_Length'] > 28) & (test_df['Name_Length'] <= 82), 'Name_Size'] = 2
test_df.loc[test_df['Name_Length'] > 82, 'Name_Size'] = 3
```

```
Out[51]: Series([], Name: Name_Size, dtype: float64)
```

```
In [52]: train_df.head()
```

Out[52]:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin
0	1	0	3	Braund, Mr. Owen Harris	male	2.00	1	0	A/5 21171	0.00	
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...)	female	5.00	1	0	PC 17599	5.00	
2	3	1	3	Heikkinen, Miss. Laina	female	3.00	0	0	STON/O2. 3101282	1.00	
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	5.00	1	0	113803	5.00	
4	5	0	3	Allen, Mr. William Henry	male	5.00	0	0	373450	1.00	

Oh, maksudnya **rekap hasil EDA (Exploratory Data Analysis)** ya! Siap, saya paham.

Berikut adalah rekap "**Kunci Jawaban Sementara**" yang sudah kita temukan dari hasil `groupby` :

## ✓ SUDAH DIBEDAH (Ada Pola Jelas)

### 1. Sex (Jenis Kelamin):

- **Pola:** Wanita (74%) jauh lebih selamat daripada Pria (19%).
- **Insight:** "Women First" itu nyata.

### 2. Pclass (Kelas Tiket):

- **Pola:** Kelas 1 (63%) > Kelas 2 (47%) > Kelas 3 (24%).
- **Insight:** Uang/Status Sosial menyelamatkan nyawa.

### 3. Age (Umur - setelah dikelompokkan):

- **Pola:** Anak-anak (0-16 thn) selamat tinggi (55%). Remaja/Dewasa Muda anjlok. Tua naik lagi dikit.
- **Insight:** "Children First" + Anak Muda berkorban/kurang pengalaman.

### 4. Fare (Harga Tiket):

- **Pola:** Tiket Mahal (>52) peluang selamatnya 70%. Tiket Murah (<7) cuma 21%.
- **Insight:** Sejalan dengan Pclass. Fasilitas VIP mungkin lebih dekat ke sekoci.

#### 5. Family\_Size (Ukuran Keluarga):

- **Pola:**
  - Sendirian (Alone) = Bahaya (30%).
  - Keluarga Kecil (2-4) = Aman (50-70%).
  - Keluarga Besar (>5) = Bencana (16%).
- **Insight:** Punya *support system* itu penting, tapi kalau kebanyakan malah ribet evakuasi.

#### 6. Embarked (Pelabuhan Asal):

- **Pola:** C (Cherbourg) = 55%. Q (Queenstown) = 39%. S (Southampton) = 34%.
- **Insight:** Cherbourg isinya orang kaya (Kelas 1). Southampton campur aduk.

#### 7. Title (Gelar Nama):

- **Pola:** Wanita Bangsawan (Noble/Mrs/Miss) → Hampir pasti selamat.
- **Pola:** Master (Bocil Cowok) → Selamat.
- **Pola:** Mr (Bapak-bapak) → Wassalam.
- **Insight:** Status sosial dan gender di nama sangat menentukan.

#### 8. Name\_Length (Panjang Nama):

- **Pola:** Nama Panjang > Nama Pendek.
- **Insight:** Nama panjang biasanya indikasi orang kaya/bangsawan.

## BELUM DIBEDAH (Misteri)

Ini kolom yang belum kita `groupby` atau belum kita lihat korelasinya dengan `Survived` :

#### 1. Ticket (Nomor Tiket):

- *Pertanyaan:* Apakah tiket yang nomornya kembar (rombongan) lebih selamat? Atau tiket yang depannya huruf 'PC' (mungkin kode VIP) lebih selamat? Belum dicek.

#### 2. Cabin (Kamar):

- *Pertanyaan:* Apakah orang di Dek A (Atas) lebih selamat daripada Dek E (Bawah)? Belum dicek karena datanya bolong parah.

#### 3. SibSp & Parch (Terpisah):

- *Status:* Tadi sudah dicek sekilas, tapi kita langsung gabung jadi `Family_Size`. Jadi secara teknis sudah terwakili.

**Kesimpulan Detektif Zaenal:** Sejauh ini, profil orang yang **PASTI SELAMAT** adalah:

"Wanita Bangsawan (Mrs/Miss/Noble), naik dari Cherbourg (C), beli tiket Mahal (Kelas 1), bawa keluarga kecil (Suami/Anak), dan namanya panjang banget."

Profil yang **PASTI WASALAM**:

"Laki-laki Dewasa (Mr), naik sendirian, beli tiket Murah (Kelas 3), umur 20-an."

```
In [53]: train_df['Ticket']
```

```
Out[53]: 0          A/5 21171
1          PC 17599
2      STON/O2. 3101282
3          113803
4          373450
...
886        211536
887        112053
888      W./C. 6607
889        111369
890        370376
Name: Ticket, Length: 891, dtype: object
```

```
In [54]: train_df['TicketNumber'] = train_df['Ticket'].apply(lambda x: pd.Series({'Ticket': test_df['TicketNumber'] = test_df['Ticket'].apply(lambda x: pd.Series({'Ticket': x.
```

```
In [55]: train_df['TicketNumber']
```

```
Out[55]: 0        21171
1        17599
2      3101282
3        113803
4        373450
...
886      211536
887      112053
888        6607
889      111369
890      370376
Name: TicketNumber, Length: 891, dtype: object
```

```
In [56]: train_df.groupby(['TicketNumber'], as_index=False, observed=False)['Survived'].agg
```

Out[56]:

	TicketNumber	count	mean
<b>196</b>	2343	7	0.00
<b>464</b>	347082	7	0.00
<b>94</b>	1601	7	0.71
<b>168</b>	2144	6	0.00
<b>468</b>	347088	6	0.00
...	...	...	...
<b>674</b>	8475	1	0.00
<b>675</b>	851	1	0.00
<b>676</b>	9234	1	1.00
<b>63</b>	11769	1	1.00
<b>647</b>	5727	1	0.00

679 rows × 3 columns

In [57]: `train_df.groupby('TicketNumber')['TicketNumber'].transform('count')`

Out[57]:

0	1
1	1
2	1
3	2
4	1
..	
886	1
887	1
888	2
889	1
890	1

Name: TicketNumber, Length: 891, dtype: int64

Siap! Mari kita bedah trik Ryan di bagian **Ticket** ini. Ini agak *tricky* karena format tiketnya berantakan.

## 1. Masalah pada Kolom **Ticket**

Coba lihat data aslinya:

- A/5 21171 (Ada kode huruf di depan + angka di belakang).
- PC 17599 (Kode huruf + angka).
- 113803 (Cuma angka).

**Tujuannya:** Ryan ingin mengambil **Angka Tiket**-nya saja (misal 21171, 17599, 113803). Dia nggak peduli sama kode huruf di depannya (A/5, PC, STON).

**Caranya (Logic):** Dia pakai `.split()[-1]`.

- "A/5 21171".`split()` → jadi `['A/5', '21171']`.
- Ambil yang paling belakang (`-1`) → dapet `21171`.
- "113803".`split()` → jadi `['113803']`.
- Ambil yang paling belakang (`-1`) → dapet `113803`.

Jadi kode: `x.split()[-1]` itu artinya "**Ambil kata paling terakhir dari string tiket**".

---

## 2. Kenapa Dia Hitung `count` Tiket?

Ryan melakukan `groupby(['TicketNumber'])` dan diurutkan. Hasilnya:

- Tiket `347082` : Muncul **7 kali**. (Artinya ada 7 orang pakai tiket yang sama).
- Tiket `1601` : Muncul **7 kali**.
- Tiket `2343` : Muncul **7 kali**.

**Hipotesis Ryan:** "Kalau satu tiket dipakai rame-rame, berarti mereka rombongan (keluarga/teman). Mungkin nasib mereka mirip?"

---

## 3. Trik `transform('count')` (Bagian Paling Penting)

Ini yang bikin kamu bingung kan? Kode: `train_df.groupby('TicketNumber')[['TicketNumber']].transform('count')`

**Apa bedanya sama `groupby` biasa?**

- **groupby biasa:** Hasilnya tabel ringkas (cuma 679 baris). Data aslinya (891 baris) jadi hilang/menyusut.
- **transform :** Hasilnya tetap 891 baris! Dia menempelkan hasil hitungan ke **setiap baris data asli**.

**Ilustrasi:**

Nama	Tiket
Budi	12345
Siti	12345
Joko	99999

**Kalau pakai `transform('count')` :** Komputer ngitung: "Tiket 12345 ada 2 orang. Tiket 99999 ada 1 orang." Lalu dia tempel angkanya:

Nama	Tiket	TicketCount (Hasil Transform)
Budi	12345	2
Siti	12345	2
Joko	99999	1

**Jadi Intinya:** Ryan ingin membuat kolom baru bernama `TicketCount` yang isinya: "**Berapa orang sih punya tiket sama kayak saya?**"

Kalau `TicketCount` = 7, berarti dia rombongan besar. Kalau `TicketCount` = 1, berarti dia sendirian.

Ini mirip sama `Family_Size`, tapi kadang ada rombongan yang bukan keluarga (teman/pembantu) yang pakai tiket sama. Jadi ini fitur tambahan yang bagus.

```
In [58]: train_df['TicketNumberCounts'] = train_df.groupby('TicketNumber')['TicketNumber'].t
test_df['TicketNumberCounts'] = test_df.groupby('TicketNumber')['TicketNumber'].tra
```

```
In [59]: train_df.groupby(['TicketNumberCounts'], as_index=False, observed=False)['Survived']
```

```
Out[59]:
```

TicketNumberCounts	count	mean
0	1	0.30
1	2	0.57
2	3	0.71
3	4	0.50
6	7	0.24
5	6	0.00
4	5	0.00

okeee aku nyoba nangkep sekali lagi yaa pakai bahasa ku sendiri yang aku paham itu seperti apa:

Oke awalnya kita tau hasil dari si groupby biasa, yang dimanaa kalau lihat ticketnumber nya itu kan ada countnya terus ada mean kemungkinan hidup. Lebih jelasnya misalnyaa seperti data ini :

TicketNumber	Count	Mean
2343	7	0.00
347082	7	0.00
1601	7	0.71

kalau kek gini doank si model gak tau apa apa nih, kan itu masih kode tiket number, yaa kalau misalnyaa model nemu kode ticket number yang lain, dia tau kalau kemungkinan selamat dari apaa?? gak ada kan? karena pola ticketnumber gak bisa dipelajari. Oleh karena itu kita ambil lagi suatu hal yang bisa dipelajari. Dengan cara pakai transform. Nah dari situ setiap orang bakalann dapett nomor unik yang nunjukkin kalau dia ituu sebenarnya orang yang punya berapa banyak tiket?? dan dari banyak tiket ituu, rata-rata keseuluruhan orang yang hidup itu berapaa??

jadi misalnya si Andi Budi Siti Ryan Joko dan Andre, misalnyaa adalah bagian dari yang punya 7 tiket sama. Awalnya kalau mislanya kode number ticket doank maka gak bisa dipelajari. Tapi kalau sekarang lihat jumlah tiketnya sama yang mereka punya bisa dipelajari kalau, orang yang punya 7 tiket, kemungkinan selamat itu segini.

## Catatan Kritis: Hipotesis Feature Engineering & Eksperimen

Dalam proses *Feature Engineering* (seperti pengelompokan `Family_Size` menjadi `Alone/Small/Large` atau `TicketCount` ), kita sebenarnya sedang melakukan **Trade-off Informasi:**

### 1. Simplifikasi (Grouping/Binning):

- **Pro:** Mengurangi *noise*, membantu model menangkap pola umum lebih cepat, dan mencegah *overfitting* pada data spesifik.
- **Con:** Menghilangkan detail nuansa (*granularity*). Contoh: Perbedaan nasib antara keluarga beranggota 2 dan 3 mungkin hilang saat digabung menjadi satu kategori 'Small'.

### 2. Data Mentah (Raw Features):

- **Pro:** Mempertahankan seluruh informasi asli. Model yang kompleks (seperti *Random Forest* atau *XGBoost*) seringkali mampu menemukan pola non-linear yang rumit dari angka mentah ini.
- **Con:** Risiko *overfitting* lebih tinggi dan komputasi bisa lebih berat.

### Prinsip Eksperimentasi:

"*There is no Free Lunch in Data Science.*"

Tidak ada jaminan bahwa fitur hasil rekayasa (seperti `Family_Size_Grouped` atau `Age_Cut` ) pasti akan meningkatkan akurasi dibandingkan fitur aslinya. Strategi yang kita terapkan saat ini adalah **Starting Hypothesis** berdasarkan *Best Practice* dan eksplorasi visual (EDA).

**Langkah Validasi Masa Depan:** Untuk mengetahui strategi mana yang paling optimal, metode yang valid adalah melakukan **A/B Testing Model**:

1. Melatih model dengan fitur hasil *grouping*.

2. Melatih model dengan fitur angka mentah (*continuous*).
3. Membandingkan skor validasi (*Cross-Validation Score*) untuk menentukan pendekatan terbaik.

In [60]: `train_df['Ticket']`

```
Out[60]: 0          A/5 21171
         1          PC 17599
         2      STON/O2. 3101282
         3          113803
         4          373450
         ...
        886          211536
        887          112053
        888      W./C. 6607
        889          111369
        890          370376
Name: Ticket, Length: 891, dtype: object
```

In [61]: `train_df['Ticket'].str.split(pat=" ", expand=True)`

	0	1	2
<b>0</b>	A/5	21171	None
<b>1</b>	PC	17599	None
<b>2</b>	STON/O2.	3101282	None
<b>3</b>	113803	None	None
<b>4</b>	373450	None	None
...	...	...	...
<b>886</b>	211536	None	None
<b>887</b>	112053	None	None
<b>888</b>	W./C.	6607	None
<b>889</b>	111369	None	None
<b>890</b>	370376	None	None

891 rows × 3 columns

## 1. Bedah `expand=True` (Ini Penting!)

Kode: `train_df['Ticket'].str.split(pat=" ", expand=True)`

**Apa bedanya pakai `expand=True` dan `expand=False` (default)?**

- `expand=False` (Default):

- Dia memotong string, tapi hasilnya tetap dalam **SATU KOLOM**. Isinya berupa *List*.
- Hasil: `[A/5, 21171]` (Masih numpuk di satu kotak).
- *Susah diambil satu-satu.*
- `expand=True` :
  - Dia memotong string, lalu **MENYEBARNYA** menjadi **Banyak Kolom Baru** (Kolom 0, Kolom 1, Kolom 2...).
  - Hasil:
    - Kolom 0: `A/5`
    - Kolom 1: `21171`
  - *Gampang diambil!* Tinggal panggil `[0]`.

### Analogi:

- `expand=False` : Kamu beli pizza dipotong-potong tapi masih di dalam satu kotak kardus.
- `expand=True` : Kamu beli pizza, potongannya langsung ditaruh di piring masing-masing (Piring 0, Piring 1).

## 2. Tantangan: "Tiket yang Cuma Angka"

Lihat baris ke-3 (`113803`):

- Kolom 0: `113803`
- Kolom 1: `None` (Kosong)

Masalahnya, kalau kita cuma ambil Kolom 0 mentah-mentah:

- Si `A/5` akan terambil sebagai kode huruf (Benar).
- Si `113803` akan terambil sebagai kode juga (Salah! Ini nomor).

**Strategi Ryan (Nanti):** Dia pasti akan bikin logika:

*"Kalau tiketnya ada spasi (pecah jadi 2), ambil yang depan. Kalau tiketnya cuma satu kata (angka doang), kasih label khusus (misal 'X' atau 'Blank')."*

Siap lihat gimana Ryan menyelesaikan masalah ini?

```
In [62]: train_df['TicketLocation'] = np.where(train_df['Ticket'].str.split(pat=" ", expand=True)[0] == ' ', 'X', train_df['Ticket'].str.split(pat=" ", expand=True)[0])
test_df['TicketLocation'] = np.where(test_df['Ticket'].str.split(pat=" ", expand=True)[0] == ' ', 'X', test_df['Ticket'].str.split(pat=" ", expand=True)[0])
```

Siap! Mari kita adu *head-to-head* antara **Bedah Nama** vs **Bedah Tiket** biar kelihatan bedanya di mana.

## KASUS 1: Bedah Nama (Gelar)

**Bentuk Data:** Konsisten 100%. Semua nama formatnya: Keluarga, Gelar. Depan.

- Braund, Mr. Owen
- Heikkinen, Miss. Laina

**Kodingan:**

```
train_df['Title'] = train_df['Name'].str.split(',')[1].str.split('.')[0]
```

**Logika:**

1. **Potong di Koma:** Braund | Mr. Owen
2. **Ambil Kanan (Index 1):** Mr. Owen
3. **Potong di Titik:** Mr | Owen
4. **Ambil Kiri (Index 0):** Mr

**Kenapa Simpel?** Karena kita **YAKIN** semua orang punya koma dan titik. Kita nggak perlu ngecek "Eh, dia punya koma gak ya?".

---

Siap! Ini dia bedah anatomi kodingannya per-potongan kecil.

Kode Raksasa:

```
np.where(train_df['Ticket'].str.split(pat=" ", expand=True)[1].notna(),
         train_df['Ticket'].str.split(pat=" ", expand=True)[0].apply(lambda x:
x.strip()), 'Blank')
```

Mari kita cincang!

**Potongan 1:** `train_df['Ticket'].str.split(pat=" ", expand=True)`

- **Fungsi:** "Belah tiket jadi dua kolom berdasarkan spasi."
- **Hasil:** Tabel bayangan dengan Kolom 0 dan Kolom 1.

**Potongan 2:** `[1].notna()`

- **Fungsi:** "Cek Kolom 1 (Buntut). Apakah ada isinya (bukan NaN)?"
- **Hasil:** Daftar True/False.
  - A/5 21171 → **True** (Punya buntut).
  - 113803 → **False** (Gak punya buntut).

**Potongan 3:** `np.where(KONDISI, JIKA_YA, JIKA_TIDAK)`

- **Fungsi:** Ini adalah IF-ELSE versi cepat (vektor).
- **Logika:** "Kalau Kondisi (Potongan 2) itu **True**, jalankan Perintah A. Kalau **False**, jalankan Perintah B."

Potongan 4 (JIKA\_YA): `...[0].apply(lambda x: x.strip())`

- **Fungsi:** "Ambil Kolom 0 (Kepala), lalu bersihkan spasinya."
- **Kenapa:** Karena kadang ada spasi nyangkut.
- **Hasil:** A/5 (Bersih).

Potongan 5 (JIKA\_TIDAK): `'Blank'`

- **Fungsi:** "Tulis teks 'Blank' aja."
- **Kenapa:** Karena kalau dia False (Gak punya buntut), berarti dia gak punya kode lokasi.

### Gabungan Semuanya:

"Coba belah tiketnya. Kalau belahannya lengkap (punya buntut), ambil kepalanya sebagai kode. Kalau belahannya gak lengkap (cuma satu potong), tulis aja 'Blank'."

In [63]: `train_df['TicketLocation'].value_counts()`

```
Out[63]: TicketLocation
Blank           665
PC              60
C.A.             27
STON/O            12
A/5              10
W./C.             9
CA.              8
SOTON/O.Q.        8
A/5.              7
SOTON/OQ          7
STON/02.          6
CA                6
C                 5
S.O.C.             5
SC/PARIS           5
F.C.C.             5
SC/Paris            4
A/4.              3
PP                3
A/4               3
S.O./P.P.           3
SC/AH              3
A./5.              2
P/PP              2
A.5.              2
WE/P              2
SOTON/02           2
S.C./PARIS          2
S.C./A.4.           1
Fa                1
S.O.P.             1
SO/C              1
S.P.              1
A4.              1
W.E.P.             1
A/S              1
SC                1
SW/PP              1
SCO/W              1
W/C              1
S.W./PP             1
F.C.              1
C.A./SOTON           1
Name: count, dtype: int64
```

Ada beberapa hal yang keliatannya mirip dan bisa dijadikan satu misalnya `C.A.` dengan `CA.` yang bisa dikatakan sebagai `CA` dan sebagainya.

```
In [64]: train_df['TicketLocation'] = train_df['TicketLocation'].replace({
    'SOTON/O.Q.':'SOTON/OQ',
    'C.A.':'CA',
    'CA.':'CA',
    'SC/PARIS':'SC/Paris',
```

```
'S.C./PARIS':'SC/Paris',
'A/4. ':'A/4',
'A/5. ':'A/5',
'A.5. ':'A/5',
'A./5. ':'A/5',
'W./C. ':'W/C',
})

test_df['TicketLocation'] = test_df['TicketLocation'].replace({
    'SOTON/O.Q.':'SOTON/OQ',
    'C.A.':'CA',
    'CA.':'CA',
    'SC/PARIS':'SC/Paris',
    'S.C./PARIS':'SC/Paris',
    'A/4. ':'A/4',
    'A/5. ':'A/5',
    'A.5. ':'A/5',
    'A./5. ':'A/5',
    'W./C. ':'W/C',
})
```

```
In [65]: train_df.groupby(['TicketLocation'], as_index=False)[ 'Survived'].agg([ 'count', 'mea
```

Out[65]:

	TicketLocation	count	mean
<b>0</b>	A/4	6	0.00
<b>1</b>	A/5	21	0.10
<b>2</b>	A/S	1	0.00
<b>3</b>	A4.	1	0.00
<b>4</b>	Blank	665	0.38
<b>5</b>	C	5	0.40
<b>6</b>	C.A./SOTON	1	0.00
<b>7</b>	CA	41	0.34
<b>8</b>	F.C.	1	0.00
<b>9</b>	F.C.C.	5	0.80
<b>10</b>	Fa	1	0.00
<b>11</b>	P/PP	2	0.50
<b>12</b>	PC	60	0.65
<b>13</b>	PP	3	0.67
<b>14</b>	S.C./A.4.	1	0.00
<b>15</b>	S.O./P.P.	3	0.00
<b>16</b>	S.O.C.	5	0.00
<b>17</b>	S.O.P.	1	0.00
<b>18</b>	S.P.	1	0.00
<b>19</b>	S.W./PP	1	1.00
<b>20</b>	SC	1	1.00
<b>21</b>	SC/AH	3	0.67
<b>22</b>	SC/Paris	11	0.45
<b>23</b>	SCO/W	1	0.00
<b>24</b>	SO/C	1	1.00
<b>25</b>	SOTON/O2	2	0.00
<b>26</b>	SOTON/OQ	15	0.13
<b>27</b>	STON/O	12	0.42
<b>28</b>	STON/O2.	6	0.50
<b>29</b>	SW/PP	1	1.00

	TicketLocation	count	mean
30	W.E.P.	1	0.00
31	W/C	10	0.10
32	WE/P	2	0.50

Berhasil melakukan Normalization (Penyamaan Format) pada kode tiket. Video tutorial mengambil asumsi bahwa hal hal mirip yang seperti tadi dikatakan, mending dijadikan satu. Meskipun bisa saja asumsi ini salah. Lalu ini katanya tidak bakalan dimasukkan ke dalam video ini. "Fail Fast, Learn Fast."

Ryan baru saja ngajarin filosofi penting dalam Data Science:

1. **Eksperimen:** Dia coba gali `TicketLocation`.
2. **Evaluasi:** Dia lihat hasilnya. Ternyata datanya terlalu nyebar (32 kategori) dan banyak kategori yang isinya cuma 1-2 orang.
3. **Keputusan:** "*I'm not going to use this in my final model.*" (Gak dipake).

## Kenapa Gak Dipake?

Karena datanya **TERLALU SEDIKIT** (`Sparse Data`).

- Kalau ada kategori yang cuma punya 1 sampel, model susah belajar. Bisa jadi itu cuma kebetulan.
- Kecuali dia mau *effort* lebih buat ngegabungin `SOTON` sama `STON` dll (research lagi berjam-jam), mendingan fitur ini di-skip dulu biar gak nambahin "noise" ke model.

**Pelajaran buat Saya pribadi:** Tidak semua fitur yang kamu oprek susah payah **HARUS** masuk ke model akhir. Kadang hasil analisisnya adalah: "*Oke, fitur ini ampas. Buang aja.*" Dan itu keputusan yang valid.

Jadi, kolom `TicketLocation` yang udah kamu bikin capek-capek itu... kemungkinan besar bakal **DI-DROP** nanti.

Lanjut ke **Cabin!**

```
In [66]: train_df['Cabin']
```

```
Out[66]: 0      NaN
         1      C85
         2      NaN
         3      C123
         4      NaN
         ...
        886     NaN
        887     B42
        888     NaN
        889     C148
        890     NaN
Name: Cabin, Length: 891, dtype: object
```

```
In [67]: train_df['Cabin'] = train_df['Cabin'].fillna('U')
train_df['Cabin'] = pd.Series([i[0] if not pd.isnull(i) else 'x' for i in train_df['Cabin']])
test_df['Cabin'] = test_df['Cabin'].fillna('U')
test_df['Cabin'] = pd.Series([i[0] if not pd.isnull(i) else 'x' for i in test_df['Cabin']])
```

## Bedah Kode: pd.Series(...) List Comprehension

Kode Ryan (yang kamu copas):

```
train_df['Cabin'] = pd.Series([i[0] if not pd.isnull(i) else 'x' for i in train_df['Cabin']])
```

Jujur, kode ini agak **berlebihan/redundant**. Kenapa? Karena di baris sebelumnya (`fillna('U')`), kita sudah mengisi semua `NaN` dengan huruf `'U'`. Jadi `pd.isnull(i)` itu sebenarnya nggak akan pernah terjadi (karena sudah nggak ada yang null).

**Cara yang Lebih Simpel & Masuk Akal:** Sebenarnya tujuannya cuma satu: "**Ambil Huruf Pertama dari Cabin**".

- `C85` → Ambil `C`.
- `B42` → Ambil `B`.
- `U` (yang tadi kita isi) → Ambil `U`.

Kode simpelnya harusnya begini:

```
# 1. Isi yang kosong dengan 'U' (Unknown)
train_df['Cabin'] = train_df['Cabin'].fillna('U')

# 2. Ambil huruf pertama (Index 0)
train_df['Cabin'] = train_df['Cabin'].apply(lambda x: x[0])
(Hasilnya sama persis sama kode panjang Ryan tadi, tapi lebih enak dibaca).
```

```
In [68]: train_df.groupby(['Cabin'], as_index=False)[['Survived']].agg(['count', 'mean'])
```

Out[68]:

	Cabin	count	mean
0	A	15	0.47
1	B	47	0.74
2	C	59	0.59
3	D	33	0.76
4	E	32	0.75
5	F	13	0.62
6	G	4	0.50
7	T	1	0.00
8	U	687	0.30

Interesting, kalau diliat dari data ini, untuk cabin cabin yang diassigned tingkat survived diatas rata rata (0.38). Sedangkan yang U atau Unassigned cukup rendah dibandingkan yang lain dan dibawah rata-rata. Dan mending dijadikan satu semuanya jadi Cabin\_Assigned buat lebih tegas aja dan mudah diterima oleh model. Mungkin sekali lagi besar kemungkinan juga kalau misalnya setiap huruf punya dampaknya masing masing.

In [69]:

```
train_df['Cabin_Assigned'] = train_df['Cabin'].apply(lambda x: 0 if x in ['U'] else 1)
test_df['Cabin_Assigned'] = test_df['Cabin'].apply(lambda x: 0 if x in ['U'] else 1)
```

In [70]:

```
train_df.groupby(['Cabin_Assigned'], as_index=False)[['Survived']].agg(['count', 'mean'])
```

Out[70]:

	Cabin_Assigned	count	mean
0	0	687	0.30
1	1	204	0.67

In [71]:

```
train_df.head()
```

Out[71]:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin
0	1	0	3	Braund, Mr. Owen Harris	male	2.00	1	0	A/5 21171	0.00	
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...)	female	5.00	1	0	PC 17599	5.00	
2	3	1	3	Heikkinen, Miss. Laina	female	3.00	0	0	STON/O2. 3101282	1.00	
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	5.00	1	0	113803	5.00	
4	5	0	3	Allen, Mr. William Henry	male	5.00	0	0	373450	1.00	



In [72]: `test_df.head()`

Out[72]:

	PassengerId	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Emba
0	892	3	Kelly, Mr. James	male	5.00	0	0	330911	1.00	U	
1	893	3	Wilkes, Mrs. James (Ellen Needs)	female	6.00	1	0	363272	0.00	U	
2	894	2	Myles, Mr. Thomas Francis	male	7.00	0	0	240276	2.00	U	
3	895	3	Wirz, Mr. Albert	male	3.00	0	0	315154	2.00	U	
4	896	3	Hirvonen, Mrs. Alexander (Helga E Lindqvist)	female	2.00	1	1	3101298	2.00	U	



In [73]: `train_df.shape`

Out[73]: (891, 24)

In [74]: `test_df.shape`

Out[74]: (418, 23)

In [75]: `train_df.info()`

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 891 entries, 0 to 890
Data columns (total 24 columns):
 #   Column           Non-Null Count  Dtype  
--- 
 0   PassengerId      891 non-null    int64  
 1   Survived         891 non-null    int64  
 2   Pclass            891 non-null    int64  
 3   Name              891 non-null    object  
 4   Sex               891 non-null    object  
 5   Age               714 non-null    float64 
 6   SibSp            891 non-null    int64  
 7   Parch            891 non-null    int64  
 8   Ticket            891 non-null    object  
 9   Fare              891 non-null    float64 
 10  Cabin             891 non-null    object  
 11  Embarked          889 non-null    object  
 12  Family_Size       891 non-null    int64  
 13  Family_Size_Grouped 891 non-null    object  
 14  Age_Cut           714 non-null    category 
 15  Fare_Cut          891 non-null    category 
 16  Title              891 non-null    object  
 17  Name_Length        891 non-null    int64  
 18  Name_LengthGB      891 non-null    category 
 19  Name_Size          891 non-null    float64 
 20  TicketNumber       891 non-null    object  
 21  TicketNumberCounts 891 non-null    int64  
 22  TicketLocation     891 non-null    object  
 23  Cabin_Assigned     891 non-null    int64  
dtypes: category(3), float64(3), int64(9), object(9)
memory usage: 149.8+ KB
```

In [76]: `test_df.info()`

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 418 entries, 0 to 417
Data columns (total 23 columns):
 #   Column           Non-Null Count  Dtype  
--- 
 0   PassengerId      418 non-null    int64  
 1   Pclass            418 non-null    int64  
 2   Name              418 non-null    object  
 3   Sex               418 non-null    object  
 4   Age               332 non-null    float64 
 5   SibSp            418 non-null    int64  
 6   Parch            418 non-null    int64  
 7   Ticket            418 non-null    object  
 8   Fare              418 non-null    float64 
 9   Cabin             418 non-null    object  
 10  Embarked          418 non-null    object  
 11  Family_Size       418 non-null    int64  
 12  Family_Size_Grouped 418 non-null    object  
 13  Age_Cut           332 non-null    category 
 14  Fare_Cut          418 non-null    category 
 15  Title              418 non-null    object  
 16  Name_Length        418 non-null    int64  
 17  Name_LengthGB      418 non-null    category 
 18  Name_Size          418 non-null    float64 
 19  TicketNumber       418 non-null    object  
 20  TicketNumberCounts 418 non-null    int64  
 21  TicketLocation     418 non-null    object  
 22  Cabin_Assigned     418 non-null    int64  
dtypes: category(3), float64(3), int64(8), object(9)
memory usage: 67.5+ KB
```

In [77]: `train_df.columns`

Out[77]: `Index(['PassengerId', 'Survived', 'Pclass', 'Name', 'Sex', 'Age', 'SibSp',  
           'Parch', 'Ticket', 'Fare', 'Cabin', 'Embarked', 'Family_Size',  
           'Family_Size_Grouped', 'Age_Cut', 'Fare_Cut', 'Title', 'Name_Length',  
           'Name_LengthGB', 'Name_Size', 'TicketNumber', 'TicketNumberCounts',  
           'TicketLocation', 'Cabin_Assigned'],  
           dtype='object')`

In [78]: `train_df['Age'] = train_df['Age'].fillna(train_df['Age'].mean())  
           test_df['Age'] = test_df['Age'].fillna(test_df['Age'].mean())  
           # train_df['Age'].fillna(train_df['Age'].mean(), inplace=True) not the best use/prac  
           # test_df['Age'].fillna(test_df['Age'].mean(), inplace=True)  
           # test_df['Fare'].fillna(test_df['Fare'].mean(), inplace=True) ini baru ditulis di v`

Sip, saya sudah baca semuanya (kode kamu dan transkrip Ryan). Posisi kita sekarang sangat solid!

Mari kita bahas dua poin penting yang Ryan ragukan (dan kamu tanyakan):

## 1. Masalah `inplace=True`

Ryan bilang: "*I know it's not the best usage case you're not technically supposed to do that*"

**Kenapa?** Seperti yang sudah kita bahas tadi (saat muncul warning merah), `inplace=True` itu "Jadul" dan akan segera dihapus oleh Pandas di masa depan.

- **Cara Jadul:** `df.fillna(..., inplace=True)` (Memodifikasi tabel langsung).
- **Cara Modern:** `df = df.fillna(...)` (Membuat salinan baru lalu menimpa variabel lama).

**Saran Saya:** Sebaiknya kamu pakai **Cara Modern** (seperti yang kamu lakukan di `Fare` tadi) biar kodinganmu awet dan tidak kena *Deprecation Warning* merah-merah lagi.

Ganti kode `Age` Ryan yang ini:

```
train_df['Age'].fillna(train_df['Age'].mean(), inplace=True)
```

Menjadi ini:

```
train_df['Age'] = train_df['Age'].fillna(train_df['Age'].mean())
test_df['Age'] = test_df['Age'].fillna(test_df['Age'].mean())
```

---

## 2. Masalah "Blank Ages Impact" (Imputasi Pakai Rata-rata)

Ryan bilang: "*I don't know if that's 100% the correct thing to do... maybe blank ages had an impact on survival.*"

**Analisis:** Ryan benar. Mengisi `Age` kosong dengan **Rata-rata (Mean)** itu cara paling gampang (Naif), tapi bisa jadi menyesatkan.

**Kenapa Menyesatkan?** Ingat tadi kita sudah ubah `Age` jadi **Kategori (0-7)**?

- Kalau kamu isi rata-rata (misal `29.7`), nanti dia masuk ke kategori tertentu (misal kategori 4).
- Padahal, bisa jadi orang yang umurnya kosong itu sebenarnya Bayi (Kategori 0) atau Kakek-kakek (Kategori 7).
- Dengan makna ngisi "Rata-rata", kita **memalsukan data**.

**Alternatif yang Lebih Canggih (Masa Depan):** Kita bisa menebak umur berdasarkan **Gelar (Title)**.

- Kalau gelarnya "Master" → Pasti anak-anak (Isi umur 5 tahun).
- Kalau gelarnya "Mrs" → Pasti dewasa (Isi umur 30 tahun).
- Kalau gelarnya "Miss" → Bisa remaja (Isi umur 20 tahun).

**Tapi...** Untuk tutorial pemula ini, cara Ryan (isi rata-rata) sudah **CUKUP**. Jangan mempersulit diri dulu. Nanti kalau modelnya jelek, baru kita balik lagi ke sini buat benerin cara ngisi umurnya (Part 2 kata Ryan).

---

## Kesimpulan Langkah Ini

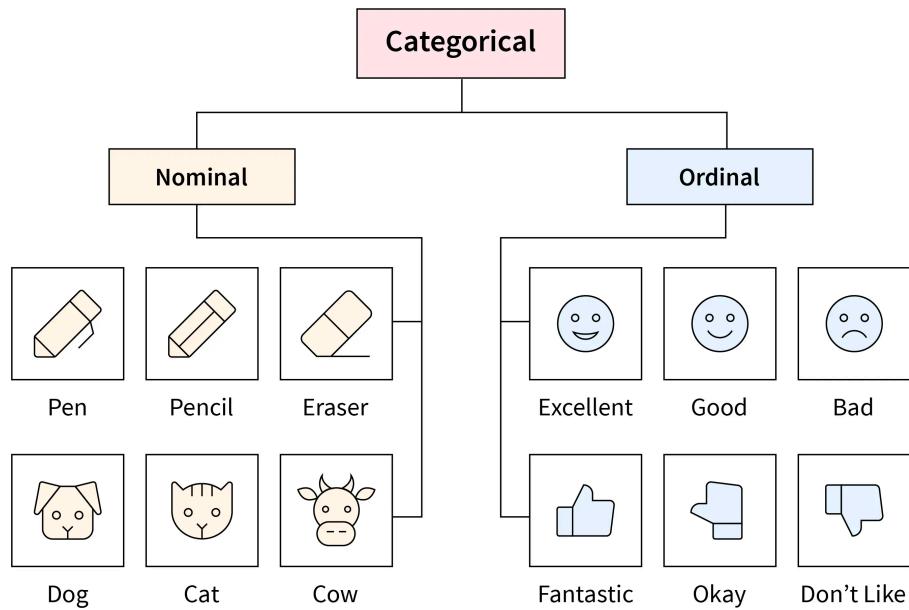
1. **Status:** Kita sudah mengisi semua lubang (`Age` dan `Fare`).
2. **Next:** `Embarked` juga masih ada bolong 2 biji di `train`. Ryan pasti bakal ngisi itu juga sebentar lagi (biasanya pakai Modus/'S').
3. **Hati-hati:** Jangan pakai `inplace=True` lagi ya.

```
In [79]: ohe = OneHotEncoder(sparse_output=False)
ode = OrdinalEncoder
SI = SimpleImputer(strategy='most_frequent')
```

```
In [80]: ode_cols = ['Family_Size_Grouped']
ohe_cols = ['Sex', 'Embarked']
```

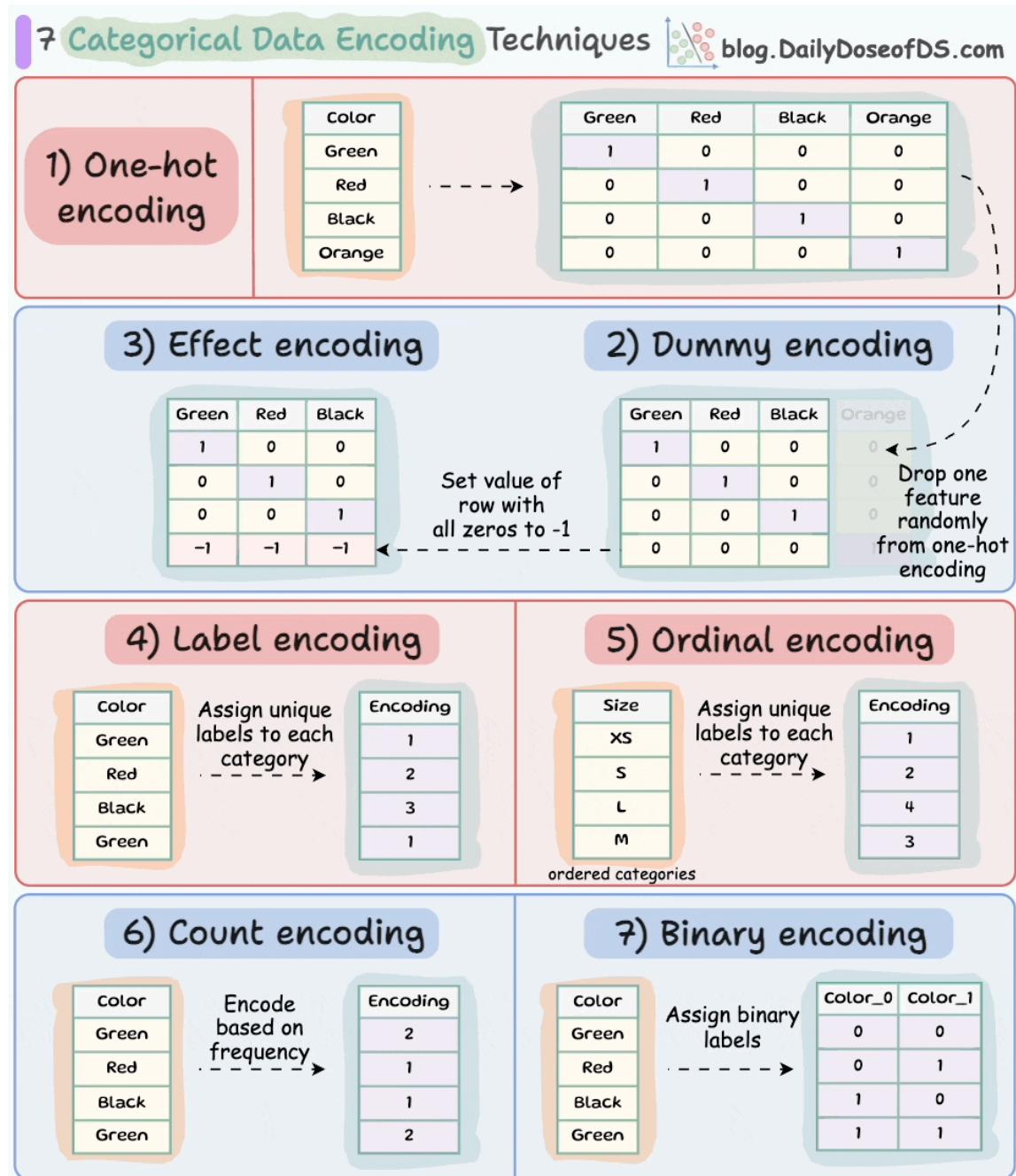
## Bedanya apa sih antara metode metode encoding yang ada?

Nah sebelum itu kita harus tau dulu bahwasanya, model machine learning tidak bisa mempelajari kategori data yang masih berupa teks. Misalnya, pada contoh ini, model tidak tau apa itu `male` dan `female`, lalu juga pada kolom `Embarked` : C, Q, S. lalu ada juga misalnya yang berupa `Family_Size_Grouped` didalamnya isinya adalah small, medium, large. Nah ada kategorikal data yang dimana berupa seperti ranking dan yang jauh berbeda. Biar lebih paham coba lihat ini:



Gambar 3. Perbedaan antara Data ordinal dan Data Nominal

Nahh, untuk itu kita perlu beberapa strategi untuk mengatasi permasalahan tersebut, Penjelasan gambar di bawah ini sudah sangat intuitive. Namun coba nanti kedepannyaaa pelajari hal hal seperti dampak negatifnya (kayak dimensional curse), atau benefitnya agar bisa menerapkan yang terbaik.



Gambar 4. 7 Categorical Data Encoding Techniques

Hah aku masih gak nangkepp, kek misalnya yang small medium large, kan kemungkinan survived itu yang medium kan yang laping banyak seinget saya. Nahh nantinya pas ketemu 3, kan survived lebih dikit. Apakah berarti kata kamu karena 3?2 jadinya lebih baik? aku masih ga paham

*Logical Fallacy* (Kesesatan Logika) yang sering bikin Data Scientist senior pun kepeleset.

Mari kita luruskan. Ini penting banget.

## Kasus: Family Size Grouped

- **Small** (Keluarga Kecil) → Survival Tinggi (Bagus).
- **Medium** (Keluarga Sedang) → Survival Rendah.
- **Large** (Keluarga Besar) → Survival Paling Rendah.

Kalau kita pakai **Ordinal Encoder** secara naif:

- Small = 1
- Medium = 2
- Large = 3

**Apa yang dipikirkan Model Linear (Regresi)?** Model mikir: "Semakin besar angkanya (1 → 3), berarti semakin 'kuat' efeknya." Padahal faktanya: **Small (1) justru yang paling selamat!**

Jadi kalau kita kasih urutan 1, 2, 3... Model akan bingung: "Lho kok angka 1 malah selamat, angka 3 malah mati? Bukannya 3 lebih besar?" Model Linear akan kesulitan menemukan garis lurusnya.

---

## Solusi Cerdas (Yang Mungkin Dipakai Ryan)

Ada 2 cara mengatasinya:

**Cara 1: One-Hot Encoding (Aman)** Ubah jadi 3 kolom terpisah:

- `Is_Small` : 1 (Ya), 0 (Tidak).
- `Is_Medium` : 1 (Ya), 0 (Tidak).
- `Is_Large` : 1 (Ya), 0 (Tidak).
- *Efeknya:* Model bebas menentukan bobot masing-masing.
  - Model: "Oke, `Is_Small` aku kasih poin +100 (Selamat). `Is_Large` aku kasih poin -100 (Mati)."

**Cara 2: Custom Mapping (Ordinal yang Disengaja)** Kita tentukan sendiri angkanya berdasarkan **PELUANG SELAMAT**:

- Large = 0 (Paling ampas).
  - Medium = 1 (Lumayan ampas).
  - Small = 2 (Paling bagus).
  - *Efeknya:* Sekarang urutannya sesuai dengan "Kebaikan Nasib". Model Linear bakal seneng.
- 

## Tapi... Model Pohon (Decision Tree / Random Forest) itu Beda!

Ingat, kita nanti mau pakai **Random Forest**. Model Pohon itu **PINTER BANGET**. Dia nggak peduli urutan linear.

Dia cara mikirnya gini:

- "Apakah angkanya  $\leq 1.5$  (Small)?" → Jika Ya, Selamat.
- "Apakah angkanya  $> 1.5$  (Medium/Large)?" → Jika Ya, Mati.

Jadi, khusus untuk **Random Forest / XGBoost**, mau kamu kasih urutan:

- Small=1, Medium=2, Large=3
- atau
- Small=3, Medium=1, Large=2

Dia **TETAP BISA** memecahkannya dengan benar. Karena dia motong-motong data (If-Else), bukan ngitung garis lurus.

### Kesimpulan:

1. Kalau pakai **Linear Regression**: Hati-hati sama urutan angka.
2. Kalau pakai **Random Forest** (Rencana kita): Angka 1, 2, 3 gak masalah, dia bisa handle pola "naik-turun-naik" sekalipun.

Paham Mas? Jadi untuk project ini, Ordinal Encoder aman-aman aja karena kita pakai model canggih. 😊

```
In [81]: X = train_df.drop(['Survived'], axis=1)
y = train_df['Survived']
X_test = test_df.drop(['Age_Cut', 'Fare_Cut'], axis=1)
```

```
In [82]: X.head(1)
```

	<b>PassengerId</b>	<b>Pclass</b>	<b>Name</b>	<b>Sex</b>	<b>Age</b>	<b>SibSp</b>	<b>Parch</b>	<b>Ticket</b>	<b>Fare</b>	<b>Cabin</b>	<b>Embarked</b>
<b>0</b>	1	3	Braund, Mr. Owen Harris	male	2.00	1	0	A/5 21171	0.00	U	S

```
In [83]: y.head(10)
```

```
Out[83]: 0    0
         1    1
         2    1
         3    1
         4    0
         5    0
         6    0
         7    0
         8    1
         9    1
Name: Survived, dtype: int64
```

```
In [84]: X_test.head(1)
```

	PassengerId	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
0	892	3	Kelly, Mr. James	male	5.00	0	0	330911	1.00	U	Q

```
In [85]: # X = train_df.drop(['Survived'], axis=1)
# y = train_df['Survived']
# X_test = test_df.drop(['Age_Cut', 'Fare_Cut'], axis=1)
X_train, X_valid, y_train, y_valid = train_test_split(X, y, test_size=0.2, stratify
```

## 1. Apa itu `X` dan `y` ?

- **`X` (Soal Ujian):** Ini adalah seluruh data `train_df`, TAPI kolom `Survived` (Kunci Jawaban) sudah **DIBUANG** (`drop`).
  - Isinya: Pclass, Name, Sex, Age, Fare, dll.
  - *Kenapa?* Karena kita mau melatih model buat nebak. Kalau dikasih kuncinya, dia gak belajar dong, cuma nyontek.
- **`y` (Kunci Jawaban):** Ini adalah kolom `Survived` yang tadi dibuang.
  - Isinya cuma angka 0 dan 1.

---

## 2. Proses Pembelahan (`train_test_split`)

Bayangkan kamu punya **891 Soal** (Baris Data). Kode: `test_size=0.2` artinya kita mau membelah jadi:

- **80% (sekitar 712 soal)** buat Latihan Belajar (**Train**).
- **20% (sekitar 179 soal)** buat Ujian Try Out (**Valid**).

**Outputnya jadi 4 Variabel:**

1. `X_train` : 712 baris Soal Latihan. (Tanpa jawaban).
2. `y_train` : 712 Kunci Jawaban untuk Soal Latihan tadi.

3. `X_valid` : 179 baris Soal Try Out. (Disimpan buat nanti).
  4. `y_valid` : 179 Kunci Jawaban Try Out. (Disimpan di laci guru).
- 

### 3. Bedah Parameter Penting

- `stratify = y` : (INI PENTING BANGET!)
    - *Masalah:* Bayangkan kalau pas kita ambil 20% soal secara acak, eh kebetulan yang keambil soal "Meninggal" semua. Nanti pas Try Out, kita gak bisa ngetes kemampuan model nebak yang "Selamat".
    - *Solusi Stratify:* "Eh komputer, tolong pastiin perbandingan Hidup:Mati di soal Latihan dan soal Try Out itu **SAMA PERSIS** kayak aslinya (38% Hidup : 62% Mati)."
    - Jadi pembagiannya adil.
  - `random_state = 21` :
    - Biar hasil kocokannya **TETAP SAMA**. Kalau besok kamu run lagi, soal nomor 1 di `X_train` tetap si Budi, bukan ganti jadi si Siti. Biar hasil eksperimen konsisten.
- 

### Analogi Sekolah

1. **Guru (Kamu):** Punya Bank Soal 891 nomor.
2. `X & y` : Guru memisahkan lembar soal dan lembar jawaban.
3. `train_test_split` : Guru mengambil 179 soal secara acak (tapi proporsional) buat disimpan jadi bahan Ujian Akhir (Validasi). Sisa 712 soal dikasih ke murid (Model) buat PR harian.
4. **Murid Belajar:** Murid ngerjain 712 soal sambil liat kuncinya (`fit(X_train, y_train)` ).
5. **Ujian:** Murid disuruh ngerjain 179 soal sisa (`predict(X_valid)` ). Hasilnya dicocokin sama kunci yang disimpan guru (`y_valid` ).

Oke, Akunyaaaa masih bingung dengan workflownya, cobaa yaa mari kita buat peta alur kerjanya (Workflow) biar jelas siapa ngapain apa.

### Aktor Utama:

1. **Si Robot (Model):** Murid yang mau kita latih.
  2. `X_train & y_train` : Buku Pelajaran & Kunci Jawaban (80%).
  3. `X_valid & y_valid` : Soal Try Out & Kunci Jawaban (20%).
  4. `X_test` : Soal Ujian Nasional (Dari Kaggle). **GAK ADA KUNCI JAWABANNYA.**
- 

### Alur Cerita (Proses Kodingan Nanti):

## 1. Fase Belajar (Training)

- **Perintah:** `model.fit(X_train, y_train)`
- **Kejadian:**
  - Si Robot dikasih 712 soal (`X_train`) dan 712 jawaban (`y_train`).
  - Dia nyari pola: "Hmm, kalau `Sex=Female` dan `PClass=1`, di kunci jawaban nilainya 1 (Selamat). Oke catat."
  - Dia melakukan ini berulang-ulang sampai nemu rumus matematika terbaik.
  - *Output:* Robot jadi Pinter.

## 2. Fase Try Out (Validation)

- **Perintah:** `prediksi_tryout = model.predict(X_valid)`
- **Kejadian:**
  - Si Robot dikasih 179 soal (`X_valid`) **TANPA KUNCI**.
  - Robot nebak: "Nomor 1 hidup, Nomor 2 mati, Nomor 3 hidup..."
- **Perintah Evaluasi:** `score = accuracy_score(y_valid, prediksi_tryout)`
- **Kejadian:**
  - Kamu (Guru) ngeluarin kunci jawaban dari laci (`y_valid`).
  - Kamu cocokin sama tebakan robot.
  - "Wah, kamu bener 80% nih. Lumayan."
  - *Gunanya:* Kalau nilainya jelek, kita setel ulang robotnya (Tuning).

## 3. Fase Ujian Nasional (Testing / Kaggle Submission)

- **Perintah:** `prediksi_akhir = model.predict(X_test)`
  - **Kejadian:**
    - Karena nilai Try Out udah bagus, kita kasih soal sesungguhnya dari Kaggle (`X_test`).
    - Robot nebak 418 soal.
    - Hasil tebakannya kita simpan jadi file CSV (`submission.csv`).
    - File ini kita upload ke Kaggle. Kaggle yang pegang kunci jawabannya. Kaggle yang bakal ngasih nilai akhir.
- 

## Kenapa `X_test` Ada di Kodingan Tadi?

Di baris ini: `X_test = test_df.drop(['Age_Cut', 'Fare_Cut'], axis=1)`

Ini cuma persiapan. Kita nyiapin lembar soal Ujian Nasional biar formatnya sama persis kayak lembar soal Latihan (kolomnya harus sama, kolom sampah dibuang).

Kalau di `train` kolom `Age_Cut` dibuang, maka di `test` juga WAJIB dibuang. Kalau beda satu kolom aja, Robot bakal error: "*Lho kok soal ujiannya beda format sama buku pelajaran?*"

Jelas alurnya? **Belajar (Train) → Try Out (Valid) → Ujian Beneran (Test)**.

# Pipelines

Katanya ini bagian paling sulittt, mari kita pahamiii.

```
In [86]: ordinal_pipeline = Pipeline(steps=[  
    ('impute', SimpleImputer(strategy='most_frequent')),  
    ('ord', OrdinalEncoder(handle_unknown= 'use_encoded_value', unknown_value=-1))  
])
```

```
In [87]: ohe_pipeline = Pipeline(steps=[  
    ('impute', SimpleImputer(strategy='most_frequent')),  
    ('one-hot ', OneHotEncoder(handle_unknown='ignore', sparse_output=False))  
])
```

```
In [88]: col_trans = ColumnTransformer(transformers=[  
    ('impute', SI, ['Age']),  
    ('ord_pipeline', ordinal_pipeline, ode_cols),  
    ('ohe_pipeline', ohe_pipeline, ohe_cols),  
    ('passthrough', 'passthrough', ['Pclass', 'TicketNumberCounts', 'Cabin_Assigned'],  
    ),  
    remainder='drop',  
    n_jobs=-1)
```

Mari kita bedah `ColumnTransformer` ini sampai ke akar-akarnya, biar kamu paham kemana hilangnya `Sex` dan teman-temannya.

## Bedah Kode `ColumnTransformer` (Si Pabrik)

Kode Ryan:

```
col_trans = ColumnTransformer(transformers=[  
    ('impute', SI, ['Age']),  
    ('ord_pipeline', ordinal_pipeline, ode_cols),  
    ('ohe_pipeline', ohe_pipeline, ohe_cols),          # <--- LIHAT INI!  
    ('passthrough', 'passthrough', ['Pclass', 'TicketNumberCounts',  
    'Cabin_Assigned', 'Name_Size', 'Age', 'Fare'])  
],  
    remainder='drop',  
    n_jobs=-1)
```

**Penjelasan Baris per Baris:**

1. `('impute', SI, ['Age'])`

- **Tugas:** "Eh, tolong kolom `Age` diisi pakai rata-rata (`SimpleImputer`) kalau masih ada yang bolong."
- *Catatan:* Padahal tadi kita udah isi manual. Ini cuma jaga-jaga (redundant).

2. ('ord\_pipeline', ordinal\_pipeline, ode\_cols)
    - **Tugas:** "Eh, tolong kolom-kolom yang ada di daftar `ode_cols` diproses pakai Ordinal Encoder."
    - **Isi `ode_cols`:** Tadi kamu definisikan `ode_cols = ['Family_Size_Grouped']`.
    - **Hasil:** Kolom `Family_Size_Grouped` (Small, Medium) berubah jadi angka (0, 1, 2).
  3. ('ohe\_pipeline', ohe\_pipeline, ohe\_cols) ← INI JAWABANNYA!
    - **Tugas:** "Eh, tolong kolom-kolom yang ada di daftar `ohe_cols` diproses pakai One-Hot Encoder."
    - **Isi `ohe_cols`:** Tadi kamu definisikan `ohe_cols = ['Sex', 'Embarked']`.
    - **Hasil:**
      - `Sex` → Jadi `Sex_male`, `Sex_female`.
      - `Embarked` → Jadi `Embarked_S`, `Embarked_C`, `Embarked_Q`.
    - **Jadi `Sex` TIDAK HILANG!** Dia masuk lewat jalur ini.
  4. ('passthrough', 'passthrough', ['Pclass', ...])
    - **Tugas:** "Eh, kolom-kolom ini udah angka dan udah bersih. Langsung lolosin aja (`passthrough`), jangan diapa-apain."
    - **Daftar Kolom:** `Pclass`, `TicketNumberCounts`, `Cabin_Assigned`, `Name_Size`, `Age` (Lho kok ada `Age` lagi?), `Fare`.
  5. `remainder='drop'`
    - **Tugas:** "Sisa kolom yang GAK Disebut di atas, BUANG KE SAMPAH!"
    - **Yang Dibuang:** `Name`, `Ticket`, `PassengerId`, `TicketLocation` (karena Ryan gak jadi pake), `Age_Cut`, `Fare_Cut`.
- 

## Audit Kolom (Apakah Ada yang Hilang?)

Mari kita cek satu-satu:

- **Sex:** Masuk lewat `ohe_cols` (One-Hot).
- **Embarked:** Masuk lewat `ohe_cols` (One-Hot).
- **Family\_Size\_Grouped:** Masuk lewat `ode_cols` (Ordinal).
- **Pclass:** Masuk lewat `passthrough`.
- **TicketNumberCounts:** Masuk lewat `passthrough`.
- **Cabin\_Assigned:** Masuk lewat `passthrough`.
- **Name\_Size:** Masuk lewat `passthrough`.
- **Age:** Masuk lewat `passthrough` (dan juga impute, agak dobel sih).
- **Fare:** Masuk lewat `passthrough`

Siap! Mari kita bedah **Syntax Pipeline** yang emang suka bikin pusing ini. Ini ibarat kita lagi ngerakit robot di pabrik.

## Bagian 1: Menyiapkan "Tukang" (Instansiasi)

```
ohe = OneHotEncoder(sparse_output=False)
ode = OrdinalEncoder
SI = SimpleImputer(strategy='most_frequent')
```

### 1. `sparse_output=False` :

- Masalah:* Secara default, `OneHotEncoder` menghasilkan matriks "Sparse" (Hemat Memori). Bentuknya aneh, bukan tabel biasa.
- Solusi:* Kita paksa `False` supaya hasilnya jadi **Array Biasa (Numpy Array)** yang bisa dibaca manusia dan model standar.

### 2. `strategy='most_frequent'` :

- Tugas:* Kalau ada data bolong (NaN), isi pakai apa?
- Jawab:* Isi pakai **MODUS** (Data yang paling sering muncul).

## Bagian 2: Menentukan "Target Operasi"

```
ode_cols = ['Family_Size_Grouped']
ohe_cols = ['Sex', 'Embarked']
```

- Ini cuma daftar belanjaan. Kita kasih label: "Ini kolom-kolom yang mau di-Ordinal", "Ini kolom-kolom yang mau di-OneHot".

## Bagian 3: Merakit "Mesin Kecil" (Sub-Pipeline)

Di sini kita bikin mesin rakitan. Formatnya: `Pipeline(steps=[('NAMA_BEBAS', ALAT), ...])`

### Mesin A: `ordinal_pipeline`

```
ordinal_pipeline = Pipeline(steps=[
    ('impute', SimpleImputer(strategy='most_frequent')),
    ('ord', OrdinalEncoder(handle_unknown='use_encoded_value',
unknown_value=-1))
])
```

- Langkah 1** (`'impute'`): Tambal dulu kalau ada yang bolong.
- Langkah 2** (`'ord'`): Ubah jadi angka urutan (0, 1, 2).
- 'ord' itu apa?** Itu cuma **NAMA PANGGILAN (Alias)**. Kamu boleh ganti jadi `'tukang_urut'` atau `'step_2'`. Gunanya nanti kalau mau ngecek hasil step ini, kita panggil namanya.
- handle\_unknown**: Kalau nanti di data ujian (`test`) ada kategori baru yang gak ada di `train` (misal Family Size "Extra Large"), jangan error, tapi kasih angka `-1`.

### Mesin B: `ohe_pipeline`

```
ohe_pipeline = Pipeline(steps=[  
    ('impute', SimpleImputer(strategy='most_frequent')),  
    ('one-hot ', OneHotEncoder(handle_unknown='ignore',  
sparse_output=False))  
])
```

- Sama kayak di atas. Tambal dulu, baru One-Hot.
  - `handle_unknown='ignore'` : Kalau ada kategori asing, **CUEKIN AJA** (Isinya jadi 0 semua).
- 

## Bagian 4: Merakit "Pabrik Besar" (ColumnTransformer)

Ini Bos Besar yang mengatur lalu lintas. Format: `ColumnTransformer(transformers=[('NAMA_BEBAS', MESIN, KOLOM_TARGET), ...])`

```
col_trans = ColumnTransformer(transformers=[  
    ('impute', SI, ['Age']),  
    ('ord_pipeline', ordinal_pipeline, ode_cols),  
    ('ohe_pipeline', ohe_pipeline, ohe_cols),  
    ('passthrough', 'passthrough', ['Pclass', ...])  
], remainder='drop')
```

1. `('impute', SI, ['Age'])` :
    - "Hei SI (SimpleImputer), tolong urus kolom `Age`."
  2. `('ord_pipeline', ordinal_pipeline, ode_cols)` :
    - "Hei Mesin Rakitan A (`ordinal_pipeline`), tolong urus kolom-kolom yang ada di daftar `ode_cols`."
    - `'ord_pipeline'` (*kiri*): Ini cuma nama panggilan buat langkah ini di dalam Transformer.
  3. `('ohe_pipeline', ohe_pipeline, ohe_cols)` :
    - "Hei Mesin Rakitan B (`ohe_pipeline`), tolong urus kolom `Sex` dan `Embarked`."
  4. `('passthrough', ...)` :
    - "Kolom-kolom ini lolosin aja, gak usah diapa-apain."
  5. `remainder='drop'` :
    - "Kolom sisanya (PassengerId, Name, Ticket...) BUANG KE LAUT!"
- 

### Kesimpulan Syntax:

- Setiap kali ada `('nama', fungsii)`, string di kiri itu cuma **Label/Nama** biar rapi.
- Strukturnya selalu: **Siapa yang kerja? → Alatnya apa? → Bahan yang dikerjain mana?**

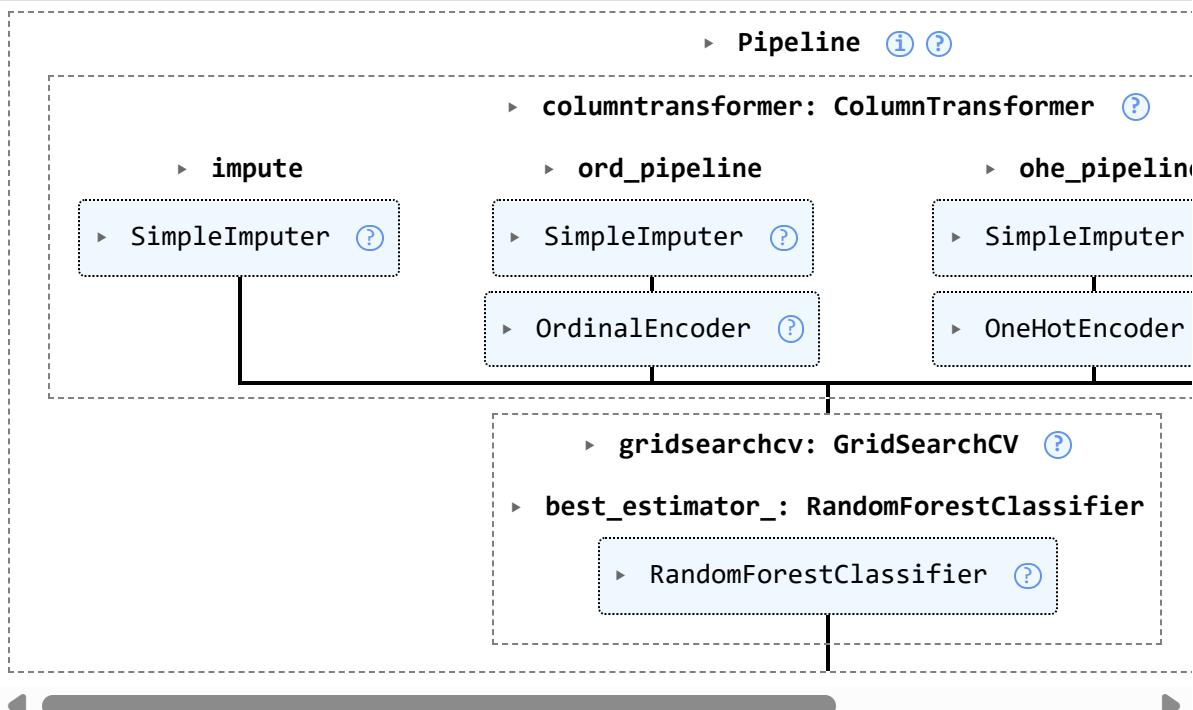
In [89]: `rfc = RandomForestClassifier()`

```
In [90]: param_grid = {
    'n_estimators': [100, 150, 200],
    'min_samples_split': [5, 10, 15],
    'max_depth': [8, 9, 10, 15, 20],
    'min_samples_leaf': [1, 2, 4],
    'criterion': ['gini', 'entropy'],
}
```

```
In [91]: CV_rfc = GridSearchCV(estimator=rfc, param_grid=param_grid, cv=StratifiedKFold(n_sp
```

```
In [93]: pipefinalrfc = make_pipeline(col_trans, CV_rfc)
pipefinalrfc.fit(X_train, y_train)
```

Out[93]:



```
In [94]: print(CV_rfc.best_params_)
print(CV_rfc.best_score_)
```

```
{'criterion': 'entropy', 'max_depth': 20, 'min_samples_leaf': 1, 'min_samples_split': 10, 'n_estimators': 100}
0.8328572835615089
```

```
In [95]: Y_pred = pipefinalrfc.predict(X_test)
```

```
In [96]: submission = pd.DataFrame({
    'PassengerID' : test_df['PassengerId'],
    'Survived' : Y_pred
})
```

```
In [97]: submission.to_csv('../data/submissionTitanic_1.csv', index=False)
```

Kita mulai dari setelah kita melakukan exploring data, bikin fitur baru seperti name-length, age\_cut dan lain sebagainya, kita mendapatkan hasil akhir kita adalah seperti ini ( setelah dari kodingan Cabin\_Assigned ) :

**Train Data Head:** train\_df.head()

PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare
1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.25
2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.28
3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.92
4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803	53.10
5	0	3	Allen, Mr. William Henry	male	35.0	0	0	373450	8.05

**Test Data Head:** test\_df.head()

PassengerId	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
892	3	Kelly, Mr. James	male	34.5	0	0	330911	7.8292	NaN	
893	3	Wilkes, Mrs. James (Ellen Needs)	female	47.0	1	0	363272	7.0000	NaN	
894	2	Myles, Mr. Thomas Francis	male	62.0	0	0	240276	9.6875	NaN	
895	3	Wirz, Mr. Albert	male	27.0	0	0	315154	8.6625	NaN	
896	3	Hirvonen, Mrs. Alexander	female	22.0	1	1	3101298	12.2875	NaN	

PassengerId	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
		(Helga E Lindqvist)								

Untuk tampilan lebih singkatnyaa, seperti ini : **Train Data Info**

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 891 entries, 0 to 890
Data columns (total 24 columns):
 #   Column           Non-Null Count  Dtype  
--- 
 0   PassengerId      891 non-null    int64  
 1   Survived         891 non-null    int64  
 2   Pclass            891 non-null    int64  
 3   Name              891 non-null    object  
 4   Sex               891 non-null    object  
 5   Age               714 non-null    float64 
 6   SibSp             891 non-null    int64  
 7   Parch             891 non-null    int64  
 8   Ticket            891 non-null    object  
 9   Fare              891 non-null    float64 
 10  Cabin             891 non-null    object  
 11  Embarked          889 non-null    object  
 12  Family_Size       891 non-null    int64  
 13  Family_Size_Grouped 891 non-null    object  
 14  Age_Cut           714 non-null    category 
 15  Fare_Cut          891 non-null    category 
 16  Title              891 non-null    object  
 17  Name_Length        891 non-null    int64  
 18  Name_LengthGB      891 non-null    category 
 19  Name_Size           891 non-null    float64 
 20  TicketNumber       891 non-null    object  
 21  TicketNumberCounts 891 non-null    int64  
 22  TicketLocation      891 non-null    object  
 23  Cabin_Assigned      891 non-null    int64  
dtypes: category(3), float64(3), int64(9), object(9)
memory usage: 149.8+ KB
```

### Test Data Info

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 418 entries, 0 to 417
Data columns (total 23 columns):
 #   Column           Non-Null Count  Dtype  
--- 
 0   PassengerId      418 non-null    int64  
 1   Pclass            418 non-null    int64  
 2   Name              418 non-null    object  
 3   Sex               418 non-null    object  
 4   Age               332 non-null    float64
```

```

5   SibSp           418 non-null    int64
6   Parch           418 non-null    int64
7   Ticket          418 non-null    object
8   Fare            418 non-null    float64
9   Cabin           418 non-null    object
10  Embarked        418 non-null    object
11  Family_Size     418 non-null    int64
12  Family_Size_Grouped 418 non-null    object
13  Age_Cut         332 non-null    category
14  Fare_Cut        418 non-null    category
15  Title           418 non-null    object
16  Name_Length     418 non-null    int64
17  Name_LengthGB   418 non-null    category
18  Name_Size        418 non-null    float64
19  TicketNumber     418 non-null    object
20  TicketNumberCounts 418 non-null    int64
21  TicketLocation   418 non-null    object
22  Cabin_Assigned   418 non-null    int64
dtypes: category(3), float64(3), int64(8), object(9)
memory usage: 67.5+ KB

```

---

Okee, coba lihat dulu ya dan baca secara lengkap nih, masing masing kolom gimana. Nah perbedaan utama data `test` dan `train` yaitu kolom `survived` yang dimana, seperti yang kita tau, untuk data `test` kita tujuan akhirnya adalah menebak itu. Sedangkan data `train` ada, karena akan kita pake buat belajar dan bikin model.

```

train_df['Age'] = train_df['Age'].fillna(train_df['Age'].mean())
test_df['Age'] = test_df['Age'].fillna(test_df['Age'].mean())

```

langkah selanjutnya, adalah mengisi kolom age, baik di data `test` dan `train`. Seperti yang dilihat, kolom `Age` banyaknya enggak sama dengan kolom lainnya. Sehingga masih ada beberapa hal yang NaN. Metode yang kita gunakan adalah menggunakan nilai rata-rata. Perlu diingat, nilai rata-rata yang digunakan disini adalah data setelah kita fitur engineering, bukan data aslinya. Kita sudah mengubah data aslinya menjadi sebuah kelompok (menggunakan `loc`). Metode ini mungkin adalah metode cepat namun kurang efektif. Kita bisa saja mengisi data ini sedikit lebih akurat, yaitu kita cek kolom lainnya, misalnya aja, kolom `name`, tadi kan kita sudah memisalkan antara gelar dan nama lengkapnya ya kan, yaitu kolom `Title`. Nah dari situ kita bisa nebak nih, kalau panggilannya A, maka umurnya sekian, kalau B, sekian dan seterusnya.

---

Okee sekarang kolom `Age` udah aman nih. langkah selanjutnya yang bisa kita lihat adalah :

```

ohe = OneHotEncoder(sparse_output=False)
ode = OrdinalEncoder
SI = SimpleImputer(strategy='most_frequent')

```

Nah disini itu, kta cuma nyiapin dulu nih, proses encoding apa yang akan kita gunakan. Kan encoding ada banyak caranya dan setiap metode seperti yang udah dijelasin

sebelumnya punya kecocokan dan plus minus masing masing. Kita disini menggunakan metode diatas.

Nah maksud dari `sparse_output=False` itu apa sih? Jadi, secara singkat ajaa nih yaa. Bedanya itu sebenarnya cuman masalah enak dilihat dan penggunaan memori. Secara garis besar kalau `sparse = True` bakalan ditampilkan dalam matriks, dan lebih hemat memori, tapi susah nih buat dibaca, sedangkan kebalikannya, lebih boros memori namun lebih enak dilihat.

---

`strategy='most_frequent'` itu berarti apabila ada data yang kosong, isilah dengan data yang paling sering muncul (Modus) di data.

Next kodingan :

```
ode_cols = ['Family_Size_Grouped']
ohe_cols = ['Sex', 'Embarked']
```

Singkat ajaa, ini kita menentukan kolom ini itu mau pakai metode apa. Kita lihat yaa, di **Train Data Info** yang masih belum berupa angka atau masih berupa kategori (teks), harus diubah jika ingin digunakan. Karena model gak bakalan tau selain angka angka.

---

Next kodingan :

```
X = train_df.drop(['Survived'], axis=1)
y = train_df['Survived']
X_test = test_df.drop(['Age_Cut', 'Fare_Cut'], axis=1)

X_train, X_valid, y_train, y_valid = train_test_split(X, y, test_size=0.2,
stratify = y, random_state=21)
```

oke maksud dari kodigan itu apa sih? tujuannya juga buat apa sihh?? Good to know aja dulu : variabel `x` biasanya berkaitan dengan data yang akan digunakan untuk belajar(soal soal) dan `y` berkaitan dengan hal hal yang berupa seperti kunci jawaban. Nah dilihat ya

`X = train_df.drop(['Survived'], axis=1)` berarti kita bikin variable X yang isinya adalah data `train` yang dimana kunci jawabannya kita drop. `axis = 1` berarti menghapus kolom (ke bawah) Kalau `axis = 0` menghapus suatu baris (ke samping).

Di Pandas:

axis	Artinya
<code>axis=0</code>	arah ke bawah → baris (row)
<code>axis=1</code>	arah ke samping → kolom (column)

 Bayangan tabel:

	A	B	C
row 0	1	2	3
row 1	4	5	6

- Buang **kolom B** → `axis=1`
- Buang **baris row 0** → `axis=0`

`y = train_df['Survived']` ini berarti kita bikin variabel `y` yang isinya cuman kunci jawabannya doank.

`X_test = test_df.drop(['Age_Cut', 'Fare_Cut'], axis=1)` kalau ini berarti kita membuat variabel `X_test` yang isinya adalah data `test` yang akan kita tebak jawabannya (`survived` 1 atau 0). Dilakukan drop pada kolom `Age_cut` dan `Fare_cut`. Jujur disini aku bingung nihh, kenapa kok cuman 2 itu aja yang di drop? saya tau sih karena keduanya berupa rentang data, yang dimana model gak bakalan tau maksudnya apa. Tapi kenapa si `Name_LengthGB` gak ikut dihapus?? Kita coba dulu lanjut lagi yaa.

`X_train, X_valid, y_train, y_valid = train_test_split(X, y, test_size=0.2, stratify = y, random_state=21)` memiliki beberapa parameter yang perlu diketahui. Maksud dari `test_size` = 0.2 berarti membagi data untuk `train` dan `valid` sebanyak 80:20. `stratify = y` berarti agar model belajar dengan komposisi yang sama. Memastikan perbandingan Hidup:Mati di soal Latihan dan soal Try Out itu **SAMA PERSIS** kayak aslinya (38% Hidup : 62% Mati). Jadi pembagiannya adil. `random_state = 21` untuk memastikan agar kalau codingan di run lagi, dia akan tetep menghasilkan hasil yang sama.

- `X_train` : Soal-soal sebanyak 80% dari banyaknya total data `train` (Variabel `X`) yang udah gak ada kunci jawabannya. Ini yang digunakan untuk belajar modelnya
- `y_train` : Isi kunci jawaban untuk soal latihan dari `X_train`. Jadi model akan belajar dari sini, dia akan coba misalnya kalau baris ke satu dengan karakteristik data begini, jawabannya adalah 1(survived), kalau baris ke 10 dengan karakteristik seperti ini jawabannya adalah 0 (not survived).
- `X_valid` : Soal-soal sebanyak 20% dari banyaknya total data `train` (Variabel `X`) yang udah gak ada kunci jawabannya. Disini model akan menebak dari soal yang diberikan (bukan lagi jawabannya).
- `y_valid` : Model akan mengecek jawaban dia dari kunci jawaban ini.

Next kodingan :

```
ordinal_pipeline = Pipeline(steps=[
    ('impute', SimpleImputer(strategy='most_frequent')),
    ('ord', OrdinalEncoder(handle_unknown= 'use_encoded_value',
unknown_value=-1))
])
```

Ini adalah sebuah pipeline. Pipeline ini akan mengolah data kolom yang perlu di encode menggunakan metode ordinal. `steps` langkah-langkah gimana sih. Nah awalnya adalah, kalau ketemu data yang masih kosong. Maka gunakan metode `SimpleImputer`, lalu untuk encoder gunakan metode `OrdinalEncoder` dengan beberapa parameter yaitu, kalau ada data yang enggak sesuai dengan apa yang kita tentukan, maka atasi dengan menempelkan angka -1 untuk baris tersebut.

```
ohe_pipeline = Pipeline(steps=[  
    ('impute', SimpleImputer(strategy='most_frequent')),  
    ('one-hot ', OneHotEncoder(handle_unknown='ignore',  
sparse_output=False))  
])
```

Sudah cukup intuitif dengan penjelasan yang sama dengan sebelumnya.

---

Next kodingan :

```
col_trans = ColumnTransformer(transformers=[  
    ('impute', SI, ['Age']),  
    ('ord_pipeline', ordinal_pipeline, ode_cols),  
    ('ohe_pipeline', ohe_pipeline, ohe_cols),  
    ('passthrough', 'passthrough', ['Pclass', 'TicketNumberCounts',  
'Cabin_Assigned', 'Name_Size', 'Age', 'Fare'])  
],  
remainder='drop',  
n_jobs=-1)
```

Ini akan menentukan, model belajar menggunakan kolom apa aja sih. Disini kolom yang digunakan adalah `Age` (Disini dilakukan lagi imput data kosong dengan metode `SimpleImputer` yang sebenarnya sudah kita lakukan di luar pipeline, saat awal tadi, sehingga ini redundant, namun mungkin biar aman saja). Lalu kita akan memasukkan kolom yang berada di list `ode_cols` ('Family\_Size\_Grouped') dan `ohe_cols` ('Sex', 'Embarked') dengan mengubah mereka (encode ke angka) dengan metode yang sesuai.

Setelah itu, kita akan meloloskan langsung(tanpa diolah dan sebagainya) untuk kolom kolom : 'Pclass', 'TicketNumberCounts', 'Cabin\_Assigned', 'Name\_Size', 'Age', 'Fare'. Disini `Age` sebenarnya enggak usah ditulis lagi.

Nah berarti kolom kolom yang digunakan adalah :

- **Sex**: Masuk lewat `ohe_cols` (One-Hot).
- **Embarked**: Masuk lewat `ohe_cols` (One-Hot).
- **Family\_Size\_Grouped**: Masuk lewat `ode_cols` (Ordinal).
- **Pclass**: Masuk lewat `passthrough`.
- **TicketNumberCounts**: Masuk lewat `passthrough`.
- **Cabin\_Assigned**: Masuk lewat `passthrough`.
- **Name\_Size**: Masuk lewat `passthrough`.
- **Age**: Masuk lewat `passthrough` (dan juga impute, agak dobel sih).

- **Fare**: Masuk lewat `passthrough`

Berikut adalah daftar kolom yang **di-drop (dibuang)** beserta alasannya:

- **PassengerId**: Hanya indeks urutan, tidak punya nilai prediksi.
- **Name**: Kardinalitas terlalu tinggi (semua unik), fiturnya sudah diwakili oleh `Name_Size`.
- **SibSp**: Sudah digabungkan informasinya ke dalam `Family_Size_Grouped`.
- **Parch**: Sudah digabungkan informasinya ke dalam `Family_Size_Grouped`.
- **Ticket**: String acak/unik, informasinya sudah diambil lewat `TicketNumberCounts`.
- **Cabin**: Terlalu banyak *missing values*, sudah diubah menjadi `Cabin_Assigned` (0/1).
- **Family\_Size**: Redundan (dobel) karena kita sudah pakai versi kategorinya yaitu `Family_Size_Grouped`.
- **Age\_Cut**: Hanya kolom pembantu (*binning*) untuk EDA, model (Random Forest) lebih suka angka `Age` aslinya.
- **Fare\_Cut**: Sama seperti Age\_Cut, hanya pembantu EDA. Model memakai `Fare` asli.
- **Title**: Tidak dimasukkan ke pipeline (mungkin karena kardinalitas lumayan tinggi atau dianggap kurang signifikan dibanding `Sex` dan `Pclass` di model ini).
- **Name\_Length**: Redundan, karena kita memilih memakai `Name_Size` (yang sudah dinormalisasi).
- **Name\_LengthGB**: Hanya kolom pembantu (*binning*).
- **TicketNumber**: Data mentah tiket, sudah diwakili `TicketNumberCounts`.
- **TicketLocation**: Tidak dimasukkan ke pipeline.

Jujur beberapa hal saya bingung sekali kenapa enggak dipake, seperti yang `Title` dimana kan kalau kita lihat, beberapa sebutan memiliki `kesempatan survived` yang berbeda dan ada polanya. Apakah akan mengganggu kah? karena ada banyak? yaitu ada 8 title berbeda? sehingga kalau di encoding bakalan kebanyakan? (di encode kan pasti kalau mau dimasukin?). Untuk yang kolom lainnya, sih udah jelas sih karena mereka diwakilkan oleh kolom lainnya yang udah sesuai biar gak redundant. Misalnya seperti `TicketLocation` `TicketNumber` yang udah diwakili oleh `Cabin_Assigned`.

Next kodingan :

```
rfc = RandomForestClassifier()
param_grid = {
    'n_estimators': [100, 150, 200],
    'min_samples_split': [5, 10, 15],
    'max_depth': [8, 9, 10, 15, 20],
    'min_samples_leaf': [1, 2, 4],
    'criterion': ['gini', 'entropy'],
}
CV_rfc = GridSearchCV(estimator=rfc, param_grid=param_grid,
cv=StratifiedKFold(n_splits=5))
```

Disini, kita kan ada beberapa model ya, kalau dilihat di line yang paling awal bangett,

```
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.naive_bayes import GaussianNB
from xgboost import XGBClassifier
from sklearn.ensemble import RandomForestClassifier, AdaBoostClassifier,
GradientBoostingClassifier, ExtraTreesClassifier, VotingClassifier
sebenarnya di video aslinya hanya ada 6 model aja, cuman di versi kaggle notebook dia, ada beberapa tambahan. Ini versi aslinya paling awal :
```

```
import pandas as pd
import numpy as np

import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline

from sklearn.preprocessing import OrdinalEncoder, OneHotEncoder

from sklearn.impute import SimpleImputer

from sklearn.compose import make_column_transformer, ColumnTransformer
from sklearn.pipeline import Pipeline, make_pipeline

from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn.ensemble import RandomForestClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.naive_bayes import GaussianNB

from sklearn.model_selection import cross_val_score, StratifiedKFold, train_test_split,
```



Gambar 5. Perbedaan antara model awal yang digunakan dengan yang terbaru

Nah di awal atau kodingan yang kita lakukan, kita memilih menggunakan algoritma `RandomForestClassifier` (Untuk detail pembelajaran mengenai algoritma ini, akan kita bahas di lain waktu, kita cukup paham dulu alurnya terlebih dahulu). `param_grid` sendiri digunakan untuk mengatur parameter parameter yang ada di masing-masing algoritma nantinya.

```
CV_rfc = GridSearchCV(estimator=rfc, param_grid=param_grid,
cv=StratifiedKFold(n_splits=5))
```

- `GridSearchCV` : Robot pencari setingen terbaik. Dia akan mencoba **Ratusan Kombinasi** secara otomatis.
- `cv=StratifiedKFold(n_splits=5)` : Metode ujiannya pakai **Cross Validation 5-Fold**.
  - Artinya: Data latihan dibagi 5 bagian. 4 bagian buat belajar, 1 bagian buat tes. Dilulang 5 kali gantian. Ini biar nilainya valid dan gak hoki-hokian doang.

```
pipefinalrfc = make_pipeline(col_trans, CV_rfc)
pipefinalrfc.fit(X_train, y_train)
```

- Kita gabungkan **Pabrik Pengolahan Data** (`col_trans`) dengan **Robot Pencari Setingen** (`CV_rfc`).
  - Jadi alurnya: Data Mentah masuk → Diolah ColumnTransformer (OneHot, Ordinal, Passthrough) → Masuk ke Random Forest yang lagi di-tuning.
- 

Pertanyaan Kritis: Bagaimana Cara Kerja Pipeline Sebenarnya?

Jujur saja, di bagian ini saya sempat benar-benar bingung. Pertanyaan besar di kepala saya adalah:

*"Data apa yang sebenarnya digunakan untuk belajar? Bukannya `X_train` itu isinya masih kotor (masih ada `Name`, `PassengerId`, dll)? Kenapa kita tidak membersihkan `X_train` dan `X_test` dulu sebelum masuk ke model? Terus, `X_valid` nasibnya gimana?"*

Mari kita bedah **Misteri di Balik Layar Pipeline** ini.

---

### A. Misteri `make_pipeline` vs `.fit()`

Kode:

```
pipefinalrfc = make_pipeline(col_trans, CV_rfc)
```

**Apa yang terjadi di sini?** Ini **BELUM** proses belajar. Ini baru tahap **Merakit Alat Masak**. Kita memberi instruksi kepada komputer:

*"Hei Komputer, nanti kalau ada data masuk, tolong prosesnya begini ya:  
Pertama, cuci bersih dulu pakai `col_trans`. Kedua, baru dimasak pakai  
`CV_rfc` (Random Forest)."*

Kode:

```
pipefinalrfc.fit(X_train, y_train)
```

**Apa yang terjadi di sini?** Ini baru **SAATNYA MEMASAK (Training)**. Di sinilah keajaiban Pipeline terjadi secara otomatis:

1. **Input Data Kotor:** Kita memasukkan `X_train` yang masih mentah (masih ada kolom sampah seperti `Name`, `PassengerId`, `Name_LengthGB`).

2. **Gerbang 1 (Column Transformer):**

- Pipeline secara otomatis memanggil `col_trans`.
- Mesin ini melihat daftar belanjaan kita (`ode_cols`, `ohe_cols`, `passthrough`).
- Ia mengambil kolom yang terdaftar (`Sex`, `Age`, `Pclass`, dll) dan memprosesnya (One-Hot, Impute).
- **PENTING:** Kolom yang **TIDAK TERDAFTAR** (seperti `Name_LengthGB`, `PassengerId`) akan otomatis **DIBUANG** (`remainder='drop'`).
- **Output:** Sebuah tabel baru yang bersih, berisi hanya angka-angka matang.

### 3. Gerbang 2 (Random Forest + GridSearch):

- Data bersih tadi langsung disodorkan ke algoritma Random Forest.
- Robot mulai belajar pola dari data bersih tersebut.

**Kesimpulan:** Kita **TIDAK PERLU** membersihkan `X_train` secara manual di luar, karena proses pembersihan itu terjadi **DI DALAM** fungsi `.fit()` milik Pipeline.

---

### B. Misteri Nasib `X_valid` dan `y_valid`

"Kenapa `X_valid` dan `y_valid` tidak dipakai di kode `fit`?"

Jawabannya ada pada parameter `cv=StratifiedKFold(n_splits=5)` di dalam `GridSearchCV`.

- **Cara Manual:** Kita memotong 20% data sebagai `X_valid` untuk Try Out manual.
- **Cara Otomatis (GridSearch):**
  - GridSearchCV sangat pintar. Dia meminjam data `X_train` yang kita berikan.
  - Di dalam "perutnya", dia membelah-belah sendiri data itu menjadi 5 bagian (Fold).
  - Dia melakukan simulasi "Try Out" internal sebanyak 5 kali secara bergantian.
  - Oleh karena itu, `X_valid` yang kita buat di awal (`train_test_split`) memang **sedang menganggur (nggak kepakai)** dalam proses pencarian parameter ini.

**Lalu buat apa `X_valid` dibuat?** Sebagai **Final Check** (Ujian Akhir Sekolah). Setelah GridSearch selesai menemukan model terbaik, kita bisa mengetes model tersebut menggunakan `X_valid` yang benar-benar "perawan" (belum pernah disentuh sama sekali oleh GridSearch) untuk memastikan nilainya jujur.

---

### C. Misteri `predict(X_test)`

Kode:

```
Y_pred = pipefinalrfc.predict(X_test)
```

"Bukannya `X_test` masih ada kolom sampahnya? Nanti error gak?"

**Jawabannya: AMAN.** Sama seperti saat `.fit()`, perintah `.predict()` pada Pipeline juga menjalankan prosedur yang sama:

1. Menerima `X_test` (yang masih kotor).
2. Otomatis masuk ke `col_trans` dulu → Kolom sampah dibuang, kolom penting di-encode.
3. Hasil bersihnya baru masuk ke Model untuk ditebak.

Jadi, Pipeline adalah **Mesin Otomatis** yang menjamin perlakuan terhadap Data Latihan (`train`) dan Data Ujian (`test`) itu **SAMA PERSIS**, sehingga kita tidak perlu repot melakukan *preprocessing* manual dua kali.

## Final Step: Creating Submission File

Setelah model berhasil dilatih (`.fit`), langkah terakhir adalah menggunakan model tersebut untuk memprediksi data ujian (`X_test`) dan menyimpannya dalam format yang diminta oleh Kaggle.

```
# 1. Melakukan Prediksi pada Data Test
Y_pred = pipefinalrfc.predict(X_test)
```

- **Penjelasan:** Kita menyuruh pipeline `pipefinalrfc` (yang sudah berisi model pintar) untuk menebak nasib penumpang di `X_test`. Pipeline akan otomatis membersihkan data `X_test` terlebih dahulu sebelum menebak. Hasilnya (`Y_pred`) adalah deretan angka 0 dan 1.

```
# 2. Membuat DataFrame untuk Submission
submission = pd.DataFrame({
    'PassengerId' : test_df['PassengerId'],
    'Survived': Y_pred
})
```

- **Penjelasan:** Kaggle hanya meminta dua kolom: `PassengerId` (sebagai identitas) dan `Survived` (tebakan kita). Kita menggabungkan ID dari data asli dengan hasil prediksi kita menjadi satu tabel baru.

```
# 3. Simpan ke File CSV
submission.to_csv('../data/submissionTitanic_1.csv', index=False)
```

- **Penjelasan:** Kita menyimpan tabel tersebut menjadi file Excel/CSV.
- `index=False` : Penting! Kita tidak mau nomor baris (0, 1, 2...) ikut tersimpan di file, karena Kaggle bisa menolak file kita kalau formatnya salah.

**Hasil Akhir:** Sebuah file bernama `submissionTitanic_1.csv` akan muncul di folder `data`. File inilah yang siap diunggah ke kompetisi Kaggle untuk dinilai skor akurasinya.

The screenshot shows the Kaggle interface for the 'Titanic - Machine Learning from Disaster' competition. On the left is a sidebar with various icons. At the top right is a search bar, a user profile icon, and a 'Submit Prediction' button. The main title 'Titanic - Machine Learning from Disaster' is displayed prominently. Below it is a subtitle: 'Start here! Predict survival on the Titanic and get familiar with ML basics'. A navigation bar below the title includes links for Data, Code, Models, Discussion, Leaderboard, Rules, Team, and Submissions. The 'Submissions' section is active. It shows a table with one row of data. The row contains a green checkmark icon, the file name 'submissionTitanic\_1.csv', the status 'Complete · now', the public score '0.76555', and three small circular icons on the far right.

Gambar 6. Hasil skor file submisi

bentar bentar, sebelum lulus aku masih penasaran, emangnya kapan ya si X kepare? dan y kepare? owhh pas disini yaaa ? X\_train, X\_valid, y\_train, y\_valid = train\_test\_split(X, y, test\_size=0.2, stratify = y, random\_state=21) ?? tapi kan ujung ujungnya si X\_train kalau misalnya si X nya gak ngedrop dulu si survived, kan bakalan di drop lagi di column transformtion??? sehingga, kalau misalnya X = train\_df doank juga boleh gak sih? karena si model kan belajar dari sini pipefinalrfc.fit(X\_train, y\_train) ini??? yang dimana di fit tadi kita udah ada proses col trans

Haha, saya suka cara kamu berpikir kritis sampai ke akar-akarnya! Kamu tidak mau menerima "pokoknya gitu" tapi ngejar "kenapa harus gitu". 🍏

Mari kita bedah skenario "**Apa jadinya kalau X = train\_df (tanpa nge-drop Survived )**".

## Skenario Eksperimen: X Masih Punya Kunci Jawaban

Misal kamu coding begini:

```
X = train_df # (Masih ada kolom Survived di dalamnya)
y = train_df['Survived']

# Split
X_train, X_valid, y_train, y_valid = train_test_split(X, y, ...)

Lalu masuk ke Pipeline ( fit ):

col_trans = ColumnTransformer(transformers=[
    # ... (Ohe, Ordinal, Passthrough) ...
], remainder='drop') # <--- Kunci di sini
```

### Pertanyaan Kamu:

"Kan Survived gak ada di daftar passthrough atau ohe\_cols, berarti nanti otomatis kebuang dong sama remainder='drop'? Berarti aman kan?"

**Jawabannya: SECARA TEKNIS: YA, AMAN (Di kasus ini).**

Karena kamu mendefinisikan kolom yang mau dipakai secara **Eksplisit (Manual)** ( ['Age', 'Sex', 'Pclass'... ] ), maka kolom apapun yang tidak disebut (termasuk Survived) akan dibuang oleh remainder='drop' .

Jadi, Model **TIDAK AKAN** melihat kunci jawaban. Model tidak akan curang.

---

## TAPI... Kenapa Orang Selalu Nge-drop Survived di Awal? (Best Practice)

Ada 2 alasan kenapa senior selalu nge-drop y dari X di awal:

### 1. Mencegah Kebodoohan di Masa Depan (Human Error):

- Bayangkan besok kamu ubah strategi Pipeline:
  - Dari: Pilih kolom A, B, C.
  - Jadi: Pakai SEMUA kolom kecuali Nama.
- Kalau Survived masih nempel di X, dan kamu lupa nge-drop di Pipeline, **TAMAT**. Model kamu bakal dapet akurasi 100% (karena dia dikasih kunci jawaban). Ini namanya **Data Leakage**.

### 2. Standar Industri (Scikit-Learn Standard):

- Fungsi `fit(X, y)` itu dirancang dengan filosofi:
  - X = Fitur (Soal).
  - y = Target (Jawaban).

- Mencampur jawaban ke dalam soal itu secara konsep "salah kamar", walaupun secara teknis bisa diakali di pipeline.

## Kesimpulan Akhir:

Hipotesis kamu **BENAR**. Kalau kamu pakai `ColumnTransformer` yang ketat (milih kolom satu-satu), kamu **BISA** membiarkan `Survived` ada di `X`, dan dia bakal kebuang otomatis.

Tapi... jangan biasakan ya. Itu kebiasaan "berbahaya" kalau nanti pindah ke metode lain.  
Pisahkan Soal dan Jawaban sejak dini biar hidup tenang. 😊 🙌