

Read Dataset

Untuk membaca dan menganalisa dataset digunakan `pandas`. Pada `pandas` terdapat fungsi `read_csv()` untuk membaca keseluruhan isi dari berkas berformat `.csv`, `head()` untuk menampilkan n baris pertama data, dan `tail()` untuk menampilkan n baris terakhir data.

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

```
df = pd.read_csv("../datasets/titanic_numeric_and_categoric_features.csv", index_col="PassengerId")
```

```
In [2]: df.head(n=5)
```

```
Out[2]:
```

	Survived	Pclass	Sex	Age	SibSp	Parch	Fare	Embarked
PassengerId								
1	0	3	male	22.0	1	0	7.2500	S
2	1	1	female	38.0	1	0	71.2833	C
3	1	3	female	26.0	0	0	7.9250	S
4	1	1	female	35.0	1	0	53.1000	S
5	0	3	male	35.0	0	0	8.0500	S

```
In [3]: df.tail(n=5)
```

```
Out[3]:
```

	Survived	Pclass	Sex	Age	SibSp	Parch	Fare	Embarked
PassengerId								
887	0	2	male	27.0	0	0	13.00	S
888	1	1	female	19.0	0	0	30.00	S
889	0	3	female	NaN	1	2	23.45	S
890	1	1	male	26.0	0	0	30.00	C
891	0	3	male	32.0	0	0	7.75	Q

Dataset Information

Informasi dari dataset, seperti jumlah baris (sampel), jumlah kolom, nama kolom, tipe data setiap kolom, ruang penyimpanan, dan lain-lain dapat diakses dengan menggunakan fungsi `info()`.

```
In [4]: df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 891 entries, 1 to 891
Data columns (total 8 columns):
 #   Column      Non-Null Count  Dtype
---  -
 0   Survived    891 non-null    int64
 1   Pclass      891 non-null    int64
 2   Sex         891 non-null    object
 3   Age         891 non-null    float64
 4   SibSp       891 non-null    int64
 5   Parch       891 non-null    int64
 6   Fare        891 non-null    float64
 7   Embarked    891 non-null    object
```

```
3   Age      714 non-null    float64
4   SibSp     891 non-null    int64
5   Parch     891 non-null    int64
6   Fare      891 non-null    float64
7   Embarked  889 non-null    object
dtypes: float64(2), int64(4), object(2)
memory usage: 62.6+ KB
```

Quick EDA (Exploratory Data and Analysis)

`pandas profiling` menyediakan fungsionalitas untuk melakukan EDA (*Exploratory Data and Analysis*) dengan cepat. Laporan akan dibuat dalam format `.html`.

```
In [5]: from pandas_profiling import ProfileReport

profile = ProfileReport(df, title="Pandas Profiling Report")
profile

Summarize dataset:   0%|          | 0/5 [00:00<?, ?it/s]
Generate report structure:  0%|          | 0/1 [00:00<?, ?it/s]
Render HTML:      0%|          | 0/1 [00:00<?, ?it/s]
```



Overview

Overview

Alerts 21

Reproduction

Dataset statistics

Number of variables	9
Number of observations	891
Missing cells	179
Missing cells (%)	2.2%
Duplicate rows	0
Duplicate rows (%)	0.0%
Total size in memory	62.8 KiB
Average record size in memory	72.1 B

Variable types

Numeric	5
Categorical	4

Variables

Out[5]:

Dataset Splitting

Proses pemisahan dataset dapat dilakukan dengan menggunakan fungsi `train_test_split` yang ada pada `scikit-learn`.

```
In [6]: from sklearn.model_selection import train_test_split
X = df.drop(columns="Survived")
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, stratify=y)

print(f"X_train shape : {X_train.shape}")
print(f"X_train shape : {y_train.shape}")
print(f"X_test shape : {X_test.shape}")
print(f"y_test shape : {y_test.shape}")

X_train shape : (712, 7)
X_train shape : (712,)
X_test shape : (179, 7)
y_test shape : (179,)
```

Build Model

Pada *notebook* ini model dibuat dengan menggunakan mekanisme *pipeline*. *Pipeline* adalah salah satu modul pada `scikit-learn` yang berfungsi untuk membungkus setiap tahapan yang biasanya dilakukan secara terpisah pada saat membuat model *machine learning*. Dengan menggunakan modul *pipeline* kode akan menjadi lebih rapi dan ringkas, sehingga mudah untuk dikelola dan dikembangkan bila ada kasus tertentu yang mengharuskan mengubah alur dari kode. Tujuan lain dari penggunaan *pipeline* adalah mengurangi *data leakage* (kebocoran informasi). Seperti yang diketahui pada saat proses *training model*, model harus dipastikan tidak mengetahui pola atau informasi dari dataset pengujian. Dengan penggunaan *pipeline* skenario tersebut dapat dilakukan dengan mudah.



Numerical Preprocessor Pipeline

Pipeline ini berfungsi untuk menangani kolom yang bertipe numerikal (continous value).

```
In [7]: from sklearn.impute import SimpleImputer
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import MinMaxScaler

numerical_prep_pipeline = Pipeline(
    steps=[
        ("step1_imputer", SimpleImputer(strategy="mean")),
        ("step2_scaler", MinMaxScaler())
    ]
)
```

Categorical Preprocessor Pipeline

Pipeline ini berfungsi untuk menangani kolom yang bertipe kategorikal (discrete value).

```
In [8]: from sklearn.impute import SimpleImputer
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import OneHotEncoder

categorical_prep_pipeline = Pipeline(
    steps=[
        ("step1_imputer", SimpleImputer(strategy="most_frequent")),
        ("step2_encoder", OneHotEncoder())
    ]
)
```

Data Preprocessor Pipeline

Pipeline ini merupakan pembungkus untuk dua *pipeline* sebelumnya, yaitu *numerical pipeline* dan *categorical pipeline*. Pipeline ini akan memilah kolom berdasarkan tipe datanya dan diteruskan ke *pipeline* yang sesuai dengan kategorinya.

```
In [9]: from sklearn.compose import ColumnTransformer, make_column_selector

preprocess_pipeline = ColumnTransformer(
    transformers=[
        ("step1_numerical_pipeline", numerical_prep_pipeline, make_column_selector(dtype
        ("step2_categorical_pipeline", categorical_prep_pipeline, make_column_selector(d
    ])
)
```

Model Pipeline

Pipeline ini merupakan pembungkus paling akhir dari semua *pipeline* yang telah dibuat sebelumnya. Pada *pipeline* ini terdiri dari 2 langkah, yaitu yang pertama transformasi fitur dan langkah kedua adalah melakukan *training model*

```
In [10]: from sklearn.neighbors import KNeighborsClassifier

model_pipeline = Pipeline([
    ("step1_preprocess_pipeline", preprocess_pipeline),
    ("step2_algo", KNeighborsClassifier(n_neighbors=3))
])
```

Feature Transform

Karena pada pipeline sebelumnya sudah termasuk pipeline transformasi fitur, maka transformasi fitur dapat diakses melalui pipeline transformasi fitur dengan mengakses fungsi `fit_transform()`.

```
In [11]: df_transform = pd.DataFrame(preprocess_pipeline.fit_transform(X_train))
df_transform.head(10)
```

```
Out[11]:
```

	0	1	2	3	4	5	6	7	8	9
0	1.0	0.266105	0.125	0.2	0.030726	0.0	1.0	1.0	0.0	0.0
1	1.0	0.402340	0.000	0.0	0.015086	0.0	1.0	0.0	1.0	0.0
2	1.0	0.578690	0.000	0.0	0.012590	0.0	1.0	0.0	0.0	1.0
3	0.5	0.565099	0.000	0.0	0.025374	0.0	1.0	0.0	0.0	1.0
4	0.5	0.361239	0.000	0.0	0.025374	0.0	1.0	0.0	0.0	1.0
5	0.0	0.782550	0.000	0.2	0.299539	1.0	0.0	0.0	0.0	1.0
6	0.0	0.633052	0.000	0.0	0.101497	0.0	1.0	0.0	0.0	1.0
7	1.0	0.402340	0.000	0.0	0.015103	0.0	1.0	0.0	1.0	0.0
8	0.5	0.456374	0.000	0.0	0.025374	0.0	1.0	0.0	0.0	1.0
9	1.0	0.381625	0.000	0.0	0.014110	0.0	1.0	1.0	0.0	0.0

Train Model with Grid Search CV Scenario

```
In [16]: from sklearn.model_selection import GridSearchCV

parameters = {
    "step2_algo__n_neighbors": range(1, 51, 2),
    "step2_algo__weights": ["uniform", "distance"],
    "step2_algo__p": [1, 2]
}

model = GridSearchCV(estimator=model_pipeline, param_grid=parameters, cv=5, scoring="acc")
model.fit(X_train, y_train)
pd.DataFrame(model.cv_results_).sort_values(by="rank_test_score").iloc[:5, :]
```

```
Out[16]:
```

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_step2_algo__n_neighbors	param_step2_alg
37	0.048798	0.012305	0.027226	0.004507		19
36	0.032932	0.002868	0.036778	0.004298		19
45	0.028603	0.001857	0.017129	0.001963		23
29	0.031911	0.001894	0.020571	0.003462		15
13	0.034472	0.006354	0.014846	0.002793		7

Evaluate Model

Pada `scikit-learn` telah tersedia rangkuman metrik dari klasifikasi yang dapat diakses menggunakan fungsi `classification_report`. Pada fungsi tersebut berisi rangkuman akurasi, *precision*, *recall*, *f1-score*, dan lain-lain untuk setiap kelas.

```
In [17]: from sklearn.metrics import classification_report

y_train_preds = model.predict(X_train)
print(classification_report(y_train, y_train_preds))
```

	precision	recall	f1-score	support
0	0.99	1.00	0.99	439
1	0.99	0.98	0.99	273
accuracy			0.99	712
macro avg	0.99	0.99	0.99	712
weighted avg	0.99	0.99	0.99	712

```
In [18]: y_test_preds = model.predict(X_test)
print(classification_report(y_test, y_test_preds))
```

	precision	recall	f1-score	support
0	0.74	0.82	0.78	110
1	0.66	0.55	0.60	69

macro avg	0.70	0.68	0.69	179
weighted avg	0.71	0.72	0.71	179

Predict use New Dataset

Prediksi dapat dilakukan dengan mengakses fungsi `predict()` pada objek model yang telah dilatih.

```
In [19]: df_new = pd.DataFrame(
    data=[[1, "male", 34., 2, 7, 300., "S"],
          [3, "female", 50., 0, 0, 7.34, "Q"]],
    columns=X_train.columns,
    index=[892, 893]
)

new_preds = model.predict(df_new)

df_new["Survived Prediction"] = new_preds
df_new
```

```
Out[19]:
```

	Pclass	Sex	Age	SibSp	Parch	Fare	Embarked	Survived Prediction
892	1	male	34.0	2	7	300.00	S	0
893	3	female	50.0	0	0	7.34	Q	1

Save Model

Setelah *training* dan evaluasi model, model dapat disimpan dengan menggunakan pustaka `joblib`. Fungsi `dump` untuk menyimpan objek model dan `load` untuk menggunakan objek yang telah disimpan sebelumnya.

```
In [20]: from joblib import dump, load

dump(model, '../pretrained_models/model_complex_workflow.joblib')
model = load("../pretrained_models/model_complex_workflow.joblib")
model.predict(df_new)
```

```
Out[20]: array([0, 1], dtype=int64)
```

Semoga bermanfaat yah 😊

Dibuat dengan penuh ❤️ oleh [haloapping](#)
