

# PEMBELAJARAN MESIN DAN PEMBELAJARAN MENDALAM

# Modul 3

## Data Scaling and Feature Engineering

# Tim Pengampu PMDPM

Semester Genap  
2022-2023

Judul: Modul 3-Data Scaling and Feature Engineering

Kontributor: Aloysius Gonzaga Pradnya Sidhwara

Versi: Maret 2023

## Daftar Isi

Tujuan pembelajaran .....	2
Konsep: Data Preprocessing dan Scaling .....	2
Konsep: Dimensional Reduction: PCA .....	3
Konsep: Automatic Feature Selection .....	4
Statistik Univariat .....	4
Pemilihan Fitur Berbasis Model .....	5
Pemilihan Fitur Iteratif .....	5
Tugas .....	6
Referensi .....	9

## Tujuan pembelajaran

Setelah menyelesaikan modul ini, mahasiswa dapat:

1. Mahasiswa dapat memahami konsep preprocessing dan mampu melakukan preprocessing transformasi data.
2. Mahasiswa dapat memahami konsep representasi fitur dan rekayasa fitur.
3. Mahasiswa mampu melakukan transformasi untuk representasi fitur.
4. Mahasiswa mampu melakukan rekayasa fitur menggunakan automatic feature selection.

## Konsep: Data Preprocessing dan Scaling

Transformasi dataset secara unsupervised merupakan algoritme yang menciptakan representasi data yang baru bagi manusia ataupun algoritme pembelajaran mesin agar lebih mudah dipahami [1]. Kita memahami bahwa beberapa algoritme seperti MLP dan SVM (algoritme berbasis persamaan linier) sangat sensitif terhadap penskalaan data. Maka dari itu, pengaturan feature dapat dilakukan agar data memiliki representasi yang lebih sesuai untuk algoritme tersebut. Ada beberapa metode penskalaan data dalam library scikit-learn:

1. StandardScaler: mengubah nilai rerata tiap feature menjadi 0 dan nilai varians menjadi 1. Scaler ini membuat semua feature ada pada besaran yang sama. Namun tidak mengatur nilai minimum dan maksimum untuk setiap feature.
2. RobustScaler: mengubah data pada besaran skala yang sama dengan menggunakan nilai median dan kuartil. Scaler ini mengabaikan data yang sangat berbeda (biasa disebut *outlier*) yang dapat menyebabkan masalah ketika menggunakan metode penskalaan yang lain.
3. MinMaxScaler: mengubah data pada setiap feature agar memiliki nilai minimum 0 dan maksimum 1.
4. Normalizer: mengubah data menjadi vector dengan panjang Euclidean 1. Scaler ini memproyeksikan data menjadi lingkaran dengan jari – jari 1. Normalizer digunakan hanya ketika arah atau sudut dari data menjadi hal yang penting.

## Konsep: Dimensional Reduction: PCA

Penerapan lain dari transformasi data secara unsupervised adalah pengurangan dimensionalitas. Salah satu tujuan pengurangan jumlah feature menjadi dua dimensi biasanya untuk kebutuhan visualisasi data. Kita dapat menggunakan metode Principal Component Analysis atau PCA untuk mengubah set features pada dataset menjadi jumlah yang lebih kecil dengan upaya mendapatkan sebanyak mungkin informasi dari dataset asli [2].

Analogi dari PCA seperti kita mengambil foto digital. Misalkan kita mengambil foto sekelompok orang berjalan dari arah depan. Objek yang kita ambil adalah 3 dimensi, sedangkan foto adalah 2 dimensi. Walaupun pengurangan dimensi ini menyebabkan kita kehilangan beberapa detail, kita masih dapat menangkap sebagian besar informasi. Misalnya kita dapat memperkirakan posisi orang yang dekat dengan kamera atau jauh dari kamera dengan adanya ukuran relatif dari tiap –tiap orang yang terekam dalam foto. Menggunakan PCA kita dapat mengurangi jumlah feature (yang disebut sebagai principal components) tanpa menghilangkan banyak informasi dari dataset asli. Kelebihan dari penggunaan PCA adalah:

1. Menghilangkan feature yang berkorelasi. PCA dapat membantu untuk menghapus feature yang berkorelasi (multi-collinear).
2. Meningkatkan performa algoritme pembelajaran mesin. Jumlah feature yang lebih sedikit membuat model lebih cepat untuk dilatih.
3. Mengurangi overfitting. Menghapus feature yang kurang penting membantu mengatasi overfitting pada model.

Kekurangan dari PCA adalah :

1. Feature yang independen kurang dapat diinterpretasikan. Principal component merupakan kombinasi linier dari beberapa feature asli sehingga kurang dapat terbaca dan diinterpretasikan.
2. Kehilangan informasi. Kehilangan data dapat terjadi ketika kita tidak memilih jumlah principal component yang tepat.

3. Penskalaan feature. PCA membutuhkan feature untuk diubah menjadi satu skala dengan besaran yang sama sebelum dilakukan prosesnya.

PCA memiliki parameter `n_components` yang dapat diisikan nilai berupa: integer untuk berapa banyak principal components yang ingin dibentuk, bisa juga diisi dengan float antara 0 sampai 1 yang akan mengembalikan jumlah komponen yang dibutuhkan untuk menangkap persentase dari variabilitas pada data, dan `None` yang berarti jumlah komponen sama dengan jumlah feature yang ada pada dataset.

## Konsep: Automatic Feature Selection

Menambahkan lebih banyak fitur membuat semua model lebih kompleks, sehingga meningkatkan kemungkinan overfitting. Saat menambahkan fitur baru, atau menggunakan dataset berdimensi tinggi, sebaiknya kurangi jumlah fitur menjadi yang paling berguna saja dan eliminasi sisanya. Model yang dibuat menjadi lebih sederhana dan dapat melakukan generalisasi dengan lebih baik. Tapi bagaimana kita bisa tahu setiap fiturnya berguna atau tidak? Ada tiga strategi dasar yang digunakan dalam pemilihan fitur otomatis: statistik univariat, pemilihan berbasis model, dan pemilihan iterative [1].

### Statistik Univariat

Dalam statistik univariat, kita menghitung apakah ada hubungan yang signifikan secara statistik antara setiap fitur dan target. Kemudian fitur yang terkait dengan kepercayaan tertinggi dipilih menggunakan metode analisis varians (ANOVA). Sifat utama dari uji ini adalah univariat, artinya metode uji hanya mempertimbangkan setiap fitur secara individual. Akibatnya, sebuah fitur akan dibuang jika fitur tersebut bersifat informatif hanya bila digabungkan dengan fitur lain.

Untuk pemilihan fitur univariat dalam scikit-learn, biasanya menggunakan `f_classif` (default) untuk klasifikasi atau `f_regression` untuk regresi, dan metode untuk mengeliminasi fitur berdasarkan nilai-p yang ditentukan dalam pengujian. Eliminasi fitur menggunakan nilai ambang batas (`alpha`) untuk eliminasi semua fitur dengan nilai p yang terlalu tinggi (yang berarti tidak terkait dengan target) dan menggunakan semua fitur dengan nilai p di bawah nilai ambang batas.

Metode dalam scikit-learn berbeda dalam cara mereka menghitung ambang batas ini. SelectKBest adalah metode paling sederhana yang memilih sejumlah "k" fitur yang tetap. SelectPercentile memilih berdasarkan persentase yang tetap dari jumlah fitur yang ada pada dataset.

### Pemilihan Fitur Berbasis Model

Pemilihan fitur berbasis model menggunakan model supervised learning untuk menilai tingkat pentingnya setiap fitur dan hanya menyimpan fitur-fitur yang paling penting. Model supervised learning yang digunakan untuk pemilihan fitur tidak harus sama dengan model yang digunakan untuk klasifikasi atau regresi. Model untuk pemilihan fitur perlu memberikan beberapa ukuran kepentingan untuk setiap fitur, sehingga setiap fitur dapat diberi peringkat berdasarkan ukuran ini. Model berbasis tree menyediakan atribut feature\_importances yang secara langsung mengkodekan tingkat pentingnya setiap fitur. Model linier memiliki koefisien yang juga dapat digunakan untuk menangkap tingkat pentingnya fitur dengan memperhitungkan nilai absolut yang ditampilkan.

### Pemilihan Fitur Iteratif

Dalam pemilihan fitur iteratif, serangkaian model dibangun dengan jumlah fitur yang bervariasi. Ada dua metode dasar yaitu dimulai tanpa fitur dan menambahkan fitur satu per satu hingga beberapa kriteria penghentian iterasi tercapai; atau dimulai dengan semua fitur dan menghapus fitur satu per satu sampai beberapa kriteria iterasi berhenti tercapai. Karena membutuhkan serangkaian model untuk dibangun, metode ini jauh lebih berat secara komputasi dibandingkan metode-metode yang telah kita bahas sebelumnya. Salah satu metode khusus semacam ini adalah eliminasi fitur rekursif (Recursive Feature Elimination), yang dimulai dengan menggunakan semua fitur, membangun model, dan mengeliminasi fitur yang paling tidak penting menurut model. Kemudian model baru dibangun menggunakan fitur yang tersisa, melakukan eliminasi fitur yang tidak penting, dan dilakukan berulang seterusnya sampai hanya sejumlah fitur yang telah ditentukan yang tersisa.

Untuk code latihan, dataset yang digunakan adalah Heart Failure Prediction Dataset (<https://www.kaggle.com/datasets/fedesoriano/heart-failure-prediction>).

## Tugas

**Studi kasus:** Stroke Prediction Dataset

(<https://www.kaggle.com/datasets/fedesoriano/stroke-prediction-dataset>)

**Deskripsi:** Menurut WHO, stroke adalah penyebab kematian nomor 2 di dunia, sekitar 11% dari total kematian. Dataset ini digunakan untuk memprediksi apakah pasien memiliki kemungkinan terkena stroke berdasarkan input seperti gender, usia, penyakit bawaan, dan status merokok [3]. Setiap baris data menyediakan informasi relevan mengenai pasien.

**Informasi atribut:**

- 1) id: unique identifier
- 2) gender: "Male", "Female" or "Other"
- 3) age: age of the patient
- 4) hypertension: 0 if the patient doesn't have hypertension, 1 if the patient has hypertension
- 5) heart\_disease: 0 if the patient doesn't have any heart diseases, 1 if the patient has a heart disease
- 6) ever\_married: "No" or "Yes"
- 7) work\_type: "children", "Govt\_jov", "Never\_worked", "Private" or "Self-employed"
- 8) Residence\_type: "Rural" or "Urban"
- 9) avg\_glucose\_level: average glucose level in blood
- 10) bmi: body mass index
- 11) smoking\_status: "formerly smoked", "never smoked", "smokes" or "Unknown"\*
- 12) stroke: 1 if the patient had a stroke or 0 if not

\*Note: "Unknown" in smoking\_status means that the information is unavailable for this patient

**Ketentuan tugas:** Buatlah notebook untuk data scaling dan feature engineering dari dataset stroke prediction. Terapkan pemrosesan dataset dari loading, data cleansing (jika perlu), data scaling, feature selection, hingga evaluasi model secara sederhana menggunakan code yang sudah dipelajari dalam latihan.

Jawab pertanyaan berikut di dalam notebook menggunakan cell markdown:

1. Apakah ada pengaruh penskalaan data pada dataset terhadap performa model machine learning?
2. Apa saja feature-feature yang penting untuk membedakan antara pasien yang rawan terkena stroke dengan yang sehat?

**Ketentuan data scaling:**

- Untuk NPM **ganjil**: gunakan **MinMaxScaler**
- Untuk NPM **genap**: gunakan **StandardScaler**

**Ketentuan feature selection:**

- Untuk NPM **ganjil**: gunakan **SelectKBest** dan **SelectFromModel** dengan ketentuan: k=5 dan model menggunakan RandomForestClassifier(n\_estimators=100, random\_state=0). Terapkan **SelectKBest** dan **SelectFromModel** pada dataset yang diubah dengan **MinMaxScaler**.
- Untuk NPM **genap**: gunakan **RFE** dan **SelectPercentile** dengan ketentuan: percentile=30 dan model RFE menggunakan RandomForestClassifier( n\_estimators=200, random\_state=42) dan n\_features\_to\_select=10. Terapkan **RFE** dan **SelectPercentile** pada dataset yang diubah dengan **StandardScaler**.

**Ketentuan modelling:**

- Untuk NPM **ganjil** gunakan **LogisticRegression** dengan parameter setting C=0.1 dan max\_iter=1000. Latih model algoritme tersebut untuk masing-masing X\_train dan ujikan pada X\_test yang **belum diubah** dan **sudah diubah** menggunakan

***MinMaxScaler+SelectKBest*** dan ***MinMaxScaler+SelectFromModel***.

Bandingkan hasil akurasi **ketiganya**!

- Untuk NPM **genap** gunakan **SVM** dengan parameter setting C=0.01, gamma=1. Latih model algoritme tersebut untuk masing-masing X\_train dan ujikan pada X\_test yang **belum diubah**, dan **sudah diubah** menggunakan **StandardScaler+RFE** dan **StandardScaler+SelectPercentile**. Bandingkan hasil akurasi **ketiganya**!

## Referensi

- [1] A. C. & G. S. Müller, Introduction to machine learning with Python: a guide for data scientists., O'Reilly Media, Inc., 2016.
- [2] W. Lee, "Using Principal Component Analysis (PCA) for Machine Learning," 2022. [Online]. Available: <https://towardsdatascience.com/using-principal-component-analysis-pca-for-machine-learning-b6e803f5bf1e>.
- [3] Fedesoriano, "Kaggle: Stroke Prediction Dataset," 2021. [Online]. Available: <https://www.kaggle.com/datasets/fedesoriano/stroke-prediction-dataset>.

# Penskalaan dan Pengkodean Data

Load dataset kedalam notebook dan ubah menjadi DataFrame (sesuaikan path dari file dataset Anda)

In [ ]:

```
from google.colab import drive  
drive.mount('/content/drive')
```

Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force\_remount=True).

In [ ]:

```
#load data kedalam dataframe  
import pandas as pd  
  
df_heart = pd.read_csv('/content/drive/MyDrive/Colab Notebooks/heart.csv')  
df_heart.head(10)
```

Out[ ]:

	Age	Sex	ChestPainType	RestingBP	Cholesterol	FastingBS	RestingECG	MaxHR	ExerciseAngina	Oldpeak	ST_Slope	HeartDisease
0	40	M	ATA	140	289	0	Normal	172	N	0.0	Up	0
1	49	F	NAP	160	180	0	Normal	156	N	1.0	Flat	1
2	37	M	ATA	130	283	0	ST	98	N	0.0	Up	0
3	48	F	ASY	138	214	0	Normal	108	Y	1.5	Flat	1
4	54	M	NAP	150	195	0	Normal	122	N	0.0	Up	0
5	39	M	NAP	120	339	0	Normal	170	N	0.0	Up	0
6	45	F	ATA	130	237	0	Normal	170	N	0.0	Up	0
7	54	M	ATA	110	208	0	Normal	142	N	0.0	Up	0
8	37	M	ASY	140	207	0	Normal	130	Y	1.5	Flat	1
9	48	F	ATA	120	284	0	Normal	120	N	0.0	Up	0

Split data menjadi training set dan test set dengan rasio test size 30%

In [ ]:

```
from sklearn.model_selection import train_test_split  
  
X = df_heart.drop('HeartDisease', axis=1)  
y = df_heart['HeartDisease']  
  
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=1)  
  
print(X_train.shape)  
print(X_test.shape)
```

(642, 11)  
(276, 11)

Ubah feature kategorikal menggunakan one hot encoder

In [ ]:

```
from sklearn.preprocessing import OneHotEncoder  
from sklearn.compose import make_column_transformer  
  
cat_cols = ['Sex', 'ChestPainType', 'RestingECG', 'ExerciseAngina', 'ST_Slope']  
  
transformer = make_column_transformer(  
    (OneHotEncoder(), cat_cols),  
    remainder='passthrough'  
)  
  
X_train_enc = transformer.fit_transform(X_train)  
X_test_enc = transformer.transform(X_test)  
  
df_train_enc=pd.DataFrame(X_train_enc,columns=transformer.get_feature_names_out())  
df_test_enc=pd.DataFrame(X_test_enc,columns=transformer.get_feature_names_out())  
  
df_train_enc.head(10)  
df_test_enc.head(10)
```

Out[ ]:

onehotencoder\_\_Sex\_F onehotencoder\_\_Sex\_M onehotencoder\_\_ChestPainType\_ASY onehotencoder\_\_ChestPainType\_ATA onehotencoder\_\_C

0	0.0	1.0	1.0	0.0
1	0.0	1.0	1.0	0.0
2	0.0	1.0	1.0	0.0
3	0.0	1.0	1.0	0.0
4	0.0	1.0	1.0	0.0
5	0.0	1.0	0.0	1.0
6	0.0	1.0	0.0	0.0
7	0.0	1.0	0.0	1.0
8	1.0	0.0	0.0	1.0
9	0.0	1.0	0.0	0.0

## 1. Min-Max Scaler

Import MinMaxScaler dan lakukan fit terhadap train set dan transformasikan bersama dengan test set. Tampilkan nilai minimum dan maksimum sebelum scaling dan sesudah scaling.

In [ ]:

```
from sklearn.preprocessing import MinMaxScaler
mm_scaler = MinMaxScaler()
mm_scaler.fit(X_train_enc)
X_train_mmscaled=mm_scaler.transform(X_train_enc)
X_test_mmscaled=mm_scaler.transform(X_test_enc)

# print dataset properties before and after scaling
print("X_train per-feature minimum sebelum scaling:\n {}".format(X_train_enc.min(axis=0)))
print("X_train per-feature maximum sebelum scaling:\n {}".format(X_train_enc.max(axis=0)))
print("X_train per-feature minimum setelah scaling:\n {}".format(X_train_mmscaled.min(axis=0)))
print("X_train per-feature maximum setelah scaling:\n {}".format(X_train_mmscaled.max(axis=0)))

X_train per-feature minimum sebelum scaling:
[ 0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.]
[28. 80.  0.  0. 60. -2.6]
X_train per-feature maximum sebelum scaling:
[ 1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.]
[ 1.  1.  76. 200. 603.  1. 202.  6.2]
X_train per-feature minimum setelah scaling:
[0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
X_train per-feature maximum setelah scaling:
[1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.]
```

## 2. Standard Scaler

Import StandardScaler dan lakukan fit terhadap train set dan transformasikan bersama dengan test set. Tampilkan nilai standard deviation sebelum scaling dan sesudah scaling pada X\_train. Tampilkan juga nilai mean sebelum scaling dan sesudah scaling pada X\_test.

In [ ]:

```
from sklearn.preprocessing import StandardScaler
st_scaler = StandardScaler()
X_train_stscaled=st_scaler.fit(X_train_enc).transform(X_train_enc)
X_test_stscaled=st_scaler.transform(X_test_enc)

print("Standard deviasi X_train per-feature sebelum scaling:\n {}".format(X_train_enc.std(axis=0)))
print("Standard deviasi X_train per-feature setelah scaling:\n {}".format(X_train_stscaled.std(axis=0)))
print("Rerata X_test per-feature sebelum scaling:\n{}".format(X_test_enc.mean(axis=0)))
print("Rerata X_test per-feature sesudah scaling:\n{}".format(X_test_stscaled.mean(axis=0)))

Standard deviasi X_train per-feature sebelum scaling:
[ 0.40861065  0.40861065  0.49871488  0.38858792  0.42017116
 0.21762304  0.40185708  0.49148177  0.40414483  0.48931727
 0.48931727  0.23006605  0.49996118  0.49750935  9.30466404
 17.70297375 111.42468453  0.42017116  25.18483727  1.06775546]
Standard deviasi X_train per-feature setelah scaling:
[1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.]
Rerata X_test per-feature sebelum scaling:
[2.06521739e-01 7.93478261e-01 5.50724638e-01 1.95652174e-01
 2.02898551e-01 5.07246377e-02 2.10144928e-01 6.23188406e-01
 1.66666667e-01 5.79710145e-01 4.20289855e-01 9.78260870e-02
 5.18115942e-01 3.84057971e-01 5.37246377e+01 1.32456522e+02]
```

```

2.02688406e+02 2.42753623e-01 1.36050725e+02 9.45289855e-01]
Rerata X test per-feature sesudah scaling:
[-0.01301059  0.01301059  0.02987497  0.02649058 -0.06205426  0.00404553
 0.01904338  0.06366074 -0.0963536 -0.04719554  0.04719554  0.18147537
 0.04869672 -0.13285739  0.03284746  0.00484694  0.04990528  0.03280011
-0.04307311  0.07757281]

```

### 3. Robust Scaler

Import RobustScaler dan lakukan fit terhadap train set dan transformasikan bersama dengan test set. Buatlah model SVM dengan parameter C=100 dan gamma=0.1, kemudian latih dengan X\_train yang belum diubah skalanya dan uji akurasi pada test set.

```
In [ ]:
from sklearn.preprocessing import RobustScaler
from sklearn.svm import SVC

rb_scaler = RobustScaler()

X_train_rbscaled=rb_scaler.fit(X_train_enc).transform(X_train_enc)
X_test_rbscaled=rb_scaler.transform(X_test_enc)

svm = SVC(C=100,gamma=0.1)
svm.fit(X_train_enc, y_train)
print("Akurasi SVM pada train set: {:.2f}".format(svm.score(X_train_enc, y_train)))
print("Akurasi SVM pada test set: {:.2f}".format(svm.score(X_test_enc, y_test)))

Akurasi SVM pada train set: 1.00
Akurasi SVM pada test set: 0.62
```

Latih kembali model SVM tersebut menggunakan train set yang sudah diubah skalanya dengan MinMaxScaler, StandardScaler, dan RobustScaler; kemudian uji akurasi pada test set yang sudah diubah skalanya.

```
In [ ]:
svm.fit(X_train_mmscaled, y_train)
print("Akurasi SVM pada Minmax-scaled train set: {:.2f}".format(svm.score(X_train_mmscaled, y_train)))
print("Akurasi SVM pada Minmax-scaled test set: {:.2f}".format(svm.score(X_test_mmscaled, y_test)))

svm.fit(X_train_stscaled, y_train)
print("\nAkurasi SVM pada Standard-scaled train set: {:.2f}".format(svm.score(X_train_stscaled, y_train)))
print("Akurasi SVM pada Standard-scaled test set: {:.2f}".format(svm.score(X_test_stscaled, y_test)))

svm.fit(X_train_rbscaled, y_train)
print("\nAkurasi SVM pada Robust-scaled train set: {:.2f}".format(svm.score(X_train_rbscaled, y_train)))
print("Akurasi SVM pada Robust-scaled test set: {:.2f}".format(svm.score(X_test_rbscaled, y_test)))

Akurasi SVM pada Minmax-scaled train set: 0.92
Akurasi SVM pada Minmax-scaled test set: 0.85

Akurasi SVM pada Standard-scaled train set: 1.00
Akurasi SVM pada Standard-scaled test set: 0.84

Akurasi SVM pada Robust-scaled train set: 0.99
Akurasi SVM pada Robust-scaled test set: 0.80
```

## PCA

Buatlah model SVC dengan kernel RBF, C=10, dan gamma=0.1. Kemudian latih model dengan train set (X\_train\_enc) dan uji akurasinya dengan test set (X\_test\_enc). Data yang digunakan adalah data sebelum dimanipulasi menggunakan PCA.

```
In [ ]:
import numpy as np
import matplotlib.pyplot as plt
from sklearn.svm import SVC
from sklearn.model_selection import train_test_split
import pandas as pd

svm=SVC(kernel='rbf',C=10,gamma=0.1)
svm.fit(X_train_enc,y_train)
print("Akurasi SVM pada train set sebelum PCA: {:.2f}".format(svm.score(X_train_enc, y_train)))
print("Akurasi SVM pada test set sebelum PCA: {:.2f}".format(svm.score(X_test_enc, y_test)))

Akurasi SVM pada train set sebelum PCA: 1.00
Akurasi SVM pada test set sebelum PCA: 0.62
```

Selanjutnya kita akan mencoba menggunakan PCA untuk mereduksi dimensi dari dataset. Sebelum diproses oleh PCA, data harus melewati preprocessing menggunakan StandardScaler. Setelah komponen ditentukan menggunakan fit maka kita dapat menampilkan explained variances seperti berikut:

```
In [ ]: from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA

st_scaler=StandardScaler()
X_train_st=st_scaler.fit_transform(X_train_enc)

pca = PCA(n_components=None)
pca.fit(X_train_st)

print("Variances (percentage):")
print(pca.explained_variance_ratio_*100)
```

```
Variances (percentage):
[2.22518648e+01 1.03965499e+01 9.29037134e+00 7.39939075e+00
 6.76571944e+00 6.38082473e+00 5.92322795e+00 5.67814555e+00
 5.02878073e+00 4.35969568e+00 4.23350411e+00 4.12266584e+00
 3.11877096e+00 2.83322437e+00 2.21726387e+00 1.64169688e-29
 3.43637564e-30 2.27020740e-30 8.12122355e-31 3.11232447e-31]
```

Angka tersebut menunjukkan bahwa komponen pertama dapat menangkap 22.25% variabilitas pada data, komponen kedua menangkap 10.39% variabilitas data, dan ketika seluruh komponen dijumlahkan maka dapat menangkap 100% variabilitas data.

Kita dapat menampilkan persentase explained variances secara kumulatif dan dalam bentuk grafik menggunakan kode berikut:

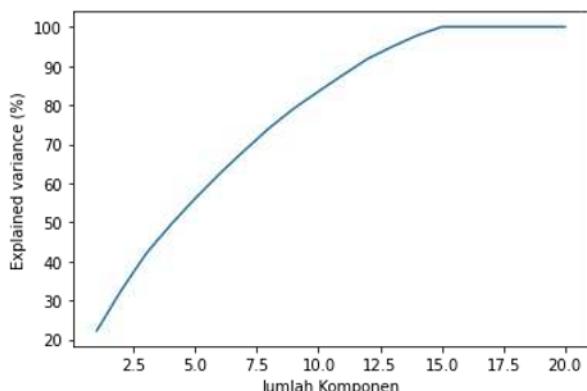
```
In [ ]: print("Cumulative Variances (percentage):")
print(pca.explained_variance_ratio_.cumsum()*100)

Cumulative Variances (percentage):
[ 22.2518648   32.64841469   41.93878603   49.33817678   56.10389622
  62.48472095   68.4079489    74.08609445   79.11487518   83.47457085
  87.70807496   91.8307408    94.94951176   97.78273613   100.
 100.          100.          100.          100.          ]
```

Kita dapat mengintepretasikan bahwa secara kumulatif: komponen pertama hanya dapat menangkap 22.29% variabilitas data, dua komponen pertama dapat menangkap 32.45% variabilitas data, sepuluh komponen pertama dapat menangkap 83.71% dari variabilitas data.

```
In [ ]: components = len(pca.explained_variance_ratio_)

plt.plot(range(1,components+1),
          np.cumsum(pca.explained_variance_ratio_*100))
plt.xlabel("Jumlah Komponen")
plt.ylabel("Explained variance (%)")
plt.show()
```



Sekarang kita dapat menerapkan PCA berdasarkan jumlah berapa persen variabilitas dari data yang ingin ditangkap pada komponen.

```
In [ ]: from sklearn.decomposition import PCA

pca=PCA(n_components=0.85)
pca.fit(X_train_st)
```

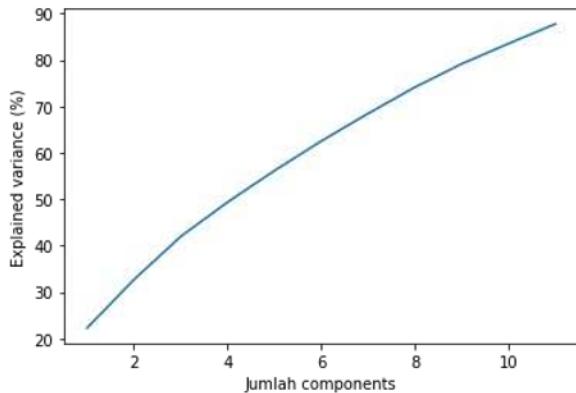
```

print("Cumulative Variances (Percentage):")
print(np.cumsum(pca.explained_variance_ratio_ * 100))
components = len(pca.explained_variance_ratio_)
print(f'Jumlah Komponen: {components}')
plt.plot(range(1, components + 1), np.cumsum(pca.explained_variance_ratio_ * 100))
plt.xlabel("Jumlah komponen")
plt.ylabel("Explained variance (%)")

```

Cumulative Variances (Percentage):  
[22.2518648 32.64841469 41.93878603 49.33817678 56.10389622 62.48472095  
68.4079489 74.08609445 79.11487518 83.47457085 87.70807496]  
Jumlah Komponen: 11

Out[ ]:



Dari grafik yang ditampilkan kita dapat melihat bahwa sebanyak 11 komponen dibutuhkan untuk mencakup 85% variabilitas dalam data.

Sekarang kita akan mencoba menerapkan PCA dengan 11 komponen pada model SVC menggunakan Pipeline. Tampilkan akurasi pada train dan test set.

In [ ]:

```

from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
from sklearn.svm import SVC

sc = StandardScaler()
pca = PCA(n_components = 11)
model = SVC(C=10, gamma=0.1)
svm_model = Pipeline([
    ('std_scaler', sc),
    ('pca', pca),
    ('svm', model)
])

svm_model.fit(X_train_enc,y_train)

print("Akurasi SVM pada train set setelah PCA: {:.2f}".format(svm_model.score(X_train_enc, y_train)))
print("Akurasi SVM pada test set setelah PCA: {:.2f}".format(svm_model.score(X_test_enc, y_test)))

```

Akurasi SVM pada train set setelah PCA: 0.95  
Akurasi SVM pada test set setelah PCA: 0.86

## Pemilihan Fitur

### 1. Select K-Best

Data yang digunakan adalah data sebelum dimanipulasi menggunakan PCA. Gunakan StandardScaler untuk melakukan penskalaan data. Buatlah obyek SelectKBest untuk seleksi fitur. Seleksi 15 fitur terbaik pada train set dengan menggunakan fungsi fit dan eliminasi fitur yang kurang penting dengan menggunakan fungsi transform dari SelectKBest. Tampilkan bentuk dari X\_train dan X\_train setelah dilakukan seleksi fitur.

In [ ]:

```

from sklearn.feature_selection import SelectKBest
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt
import numpy as np

```

```

from sklearn.preprocessing import StandardScaler
std_sc = StandardScaler()
std_sc.fit(X_train_enc)
X_train_sc=std_sc.transform(X_train_enc)
X_test_sc=std_sc.transform(X_test_enc)

SKBest = SelectKBest(k=15)
SKBest.fit(X_train_sc, y_train)

X_train_skbest = SKBest.transform(X_train_sc)

print("X_train: {}".format(X_train_enc.shape))
print("X_train_skbest: {}".format(X_train_skbest.shape))

```

X\_train: (642, 20)  
X\_train\_skbest: (642, 15)

Tampilkan fitur yang dipilih oleh SelectKbest dengan mengakses atribut mask.

```

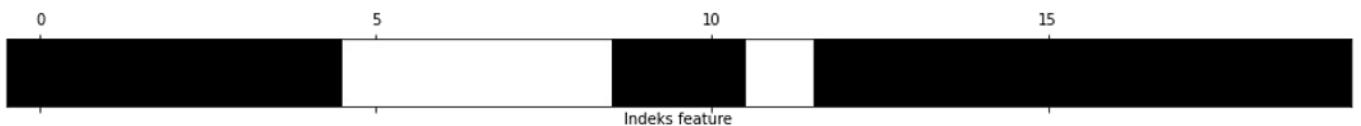
In [ ]:
mask = SKBest.get_support()
print("Feature yang terpilih: ",df_train_enc.columns[mask].values)

plt.matshow(mask.reshape(1, -1), cmap='gray_r')
plt.xlabel("Indeks feature")
plt.yticks(())

```

Feature yang terpilih: ['onehotencoder\_Sex\_F' 'onehotencoder\_Sex\_M'  
'onehotencoder\_ChestPainType\_ASY' 'onehotencoder\_ChestPainType\_ATA'  
'onehotencoder\_ChestPainType\_NAP' 'onehotencoder\_ExerciseAngina\_N'  
'onehotencoder\_ExerciseAngina\_Y' 'onehotencoder\_ST\_Slope\_Flat'  
'onehotencoder\_ST\_Slope\_Up' 'remainder\_Age' 'remainder\_RestingBP'  
'remainder\_Cholesterol' 'remainder\_FastingBS' 'remainder\_MaxHR'  
'remainder\_Oldpeak']

Out[ ]: ([], <a list of 0 Text major ticklabel objects>)



Buatlah model klasifikasi Logistic Regression dan latih serta uji akurasinya pada train dan test set asli yang memuat semua fitur. Bandingkan akurasinya dengan model yang dilatih menggunakan train dan test set yang menggunakan 15 fitur terbaik.

```

In [ ]:
from sklearn.linear_model import LogisticRegression
X_test_skbest = SKBest.transform(X_test_sc)

LogReg1 = LogisticRegression(max_iter=10000)
LogReg1.fit(X_train_enc, y_train)
print("Akurasi Logistic Regression menggunakan semua feature: {:.2f}".format(LogReg1.score(X_test_enc, y_test)))

LogReg1.fit(X_train_skbest, y_train)
print("Akurasi Logistic Regression menggunakan feature terpilih SelectKBest: {:.2f}".format(LogReg1.score(X_test_))

```

Akurasi Logistic Regression menggunakan semua feature: 0.88  
Akurasi Logistic Regression menggunakan feature terpilih SelectKBest: 0.87

## 2. Select Percentile

Import library yang dibutuhkan, kemudian lakukan train-test split dengan rasio 75:25. Gunakan MinMaxScaler untuk melakukan penskalaan data. Buatlah obyek SelectPercentile untuk seleksi fitur. Seleksi 30% fitur terbaik pada train set dengan menggunakan fungsi fit dan eliminasi fitur yang kurang penting dengan menggunakan fungsi transform dari SelectPercentile. Tampilkan bentuk dari X\_train dan X\_train setelah dilakukan seleksi fitur.

```

In [ ]:
from sklearn.feature_selection import SelectPercentile
from sklearn.preprocessing import MinMaxScaler
mm_sc=MinMaxScaler()

```

```

mm_sc.fit(X_train_enc)
X_train_sc=mm_sc.transform(X_train_enc)
X_test_sc=mm_sc.transform(X_test_enc)

SP = SelectPercentile(percentile=30)
SP.fit(X_train_sc, y_train)

X_train_sp = SP.transform(X_train_sc)

print("X_train: {}".format(X_train_enc.shape))
print("X_train_sp: {}".format(X_train_sp.shape))

X_train: (642, 20)
X_train_sp: (642, 6)

```

Tampilkan fitur yang dipilih oleh SelectPercentile dengan mengakses atribut mask.

```

In [ ]: mask = SP.get_support()
print("Feature yang terpilih: ",df_train_enc.columns[mask].values)

plt.matshow(mask.reshape(1, -1), cmap='gray_r')
plt.xlabel("Indeks Feature")
plt.yticks(())

```

```

Feature yang terpilih:  ['onehotencoder_ChestPainType_ASY' 'onehotencoder_ExerciseAngina_N'
 'onehotencoder_ExerciseAngina_Y' 'onehotencoder_ST_Slope_Flat'
 'onehotencoder_ST_Slope_Up' 'remainder_Oldpeak']

[], <a list of 0 Text major ticklabel objects>

```

```
Out[ ]:
```



Buatlah model klasifikasi Logistic Regression dan latih serta uji akurasinya pada train dan test set asli yang memuat semua fitur. Bandingkan akurasinya dengan model yang dilatih menggunakan train dan test set yang menggunakan 6 fitur terbaik.

```

In [ ]: X_test_sp = SP.transform(X_test_sc)

LogReg2 = LogisticRegression(max_iter=10000)
LogReg2.fit(X_train_enc, y_train)
print("Akurasi Logistic Regression menggunakan semua feature: {:.2f}".format(LogReg2.score(X_test_enc, y_test)))
LogReg2.fit(X_train_sp, y_train)
print("Akurasi Logistic Regression menggunakan feature terpilih SelectPercentile: {:.2f}".format(
    LogReg2.score(X_test_sp, y_test)))

Akurasi Logistic Regression menggunakan semua feature: 0.88
Akurasi Logistic Regression menggunakan feature terpilih SelectPercentile: 0.87

```

### 3. Model-Based Feature Selection

Import library yang dibutuhkan dan buat obyek dari SelectFromModel yang berisikan model Random Forest yang digunakan sebagai penyeleksi fitur. Fit dan transform X\_train dengan seleksi fitur dan tampilkan hasilnya.

```

In [ ]: from sklearn.feature_selection import SelectFromModel
from sklearn.ensemble import RandomForestClassifier

selectM = SelectFromModel(
    RandomForestClassifier(n_estimators=100, random_state=0),
    threshold="median")

selectM.fit(X_train_enc, y_train)
X_train_smodel = selectM.transform(X_train_enc)
print("X_train: {}".format(X_train_enc.shape))
print("X_train_smodel: {}".format(X_train_smodel.shape))

X_train: (642, 20)
X_train_smodel: (642, 10)

```

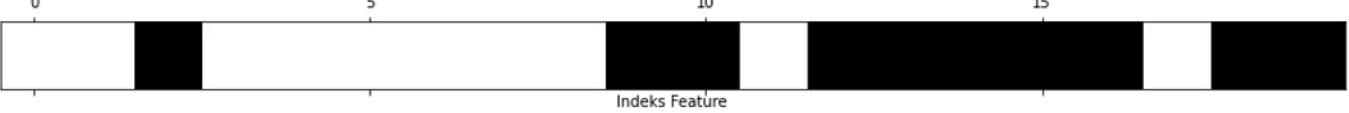
Tampilkan fitur yang dipilih oleh SelectKbest dengan mengakses atribut mask.

```
In [ ]: mask = selectM.get_support()
print("Feature yang terpilih: ",df_train_enc.columns[mask].values)

plt.matshow(mask.reshape(1, -1), cmap='gray_r')
plt.xlabel("Indeks Feature")
plt.yticks(())

Feature yang terpilih:  ['onehotencoder_ChestPainType_ASY' 'onehotencoder_ExerciseAngina_N'
 'onehotencoder_ExerciseAngina_Y' 'onehotencoder_ST_Slope_Flat'
 'onehotencoder_ST_Slope_Up' 'remainder_Age' 'remainder_RestingBP'
 'remainder_Cholesterol' 'remainder_MaxHR' 'remainder_Oldpeak']

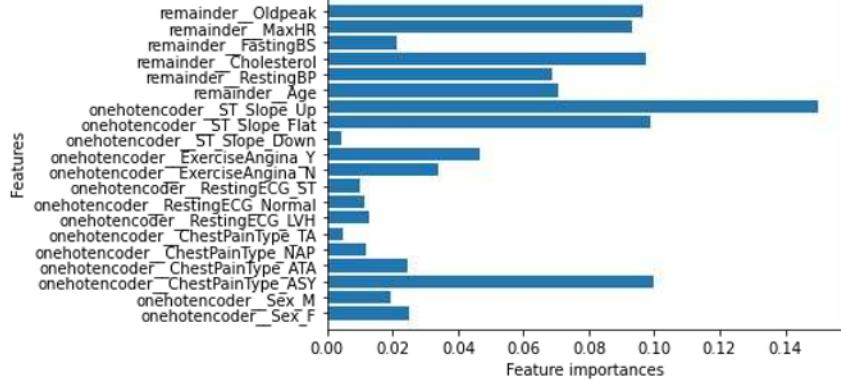
Out[ ]: ([], <a list of 0 Text major ticklabel objects>)
```



Tampilkan tingkat kepentingan masing-masing feature dengan mengakses feature\_importances dari estimator.

```
In [ ]: def plot_feature_importances(classifier,dataset, feat_names):
    n_feat=dataset.shape[1]
    plt.barh(range(n_feat),classifier.feature_importances_,align='center')
    plt.yticks(np.arange(n_feat),feat_names)
    plt.xlabel("Feature importances")
    plt.ylabel("Features")
    plt.ylim(-1,n_feat)
    plt.show()

plot_feature_importances(selectM.estimator_,df_train_enc,df_train_enc.columns)
```



Tampilkan hasil uji model LogisticRegression terhadap test set yang sudah diseleksi fiturnya.

```
In [ ]: X_test_select = selectM.transform(X_test_enc)
score = LogisticRegression(max_iter=10000).fit(X_train_smodel, y_train) \
        .score(X_test_select, y_test)
print("Test score before feature selection: {:.3f}".format(LogisticRegression(max_iter=10000) \
        .fit(X_train_enc,y_train).score(X_test_enc,y_test)))
print("Test score after feature selection: {:.3f}".format(score))

Test score before feature selection: 0.877
Test score after feature selection: 0.877
```

## 4. Iterative Feature Selection

Import library yang dibutuhkan dan buat obyek dari RFE yang berisikan model Random Forest yang digunakan sebagai penyeleksi fitur. Fit dan transform X\_train dengan seleksi fitur dan tampilkan hasilnya menggunakan atribut mask.

```
In [ ]: from sklearn.feature_selection import RFE
```

```

selectRFE = RFE(RandomForestClassifier(n_estimators=100, random_state=42),
                 n_features_to_select=15)

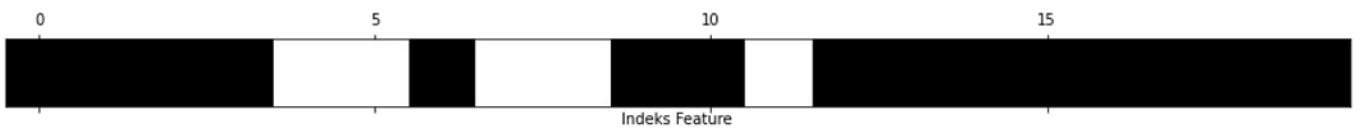
selectRFE.fit(X_train_enc, y_train)
mask = selectRFE.get_support()
print("Feature yang terpilih: ", df_train_enc.columns[mask])

plt.matshow(mask.reshape(1, -1), cmap='gray_r')
plt.xlabel("Indeks Feature")
plt.yticks(())

```

Feature yang terpilih: Index(['onehotencoder\_Sex\_F', 'onehotencoder\_Sex\_M',  
 'onehotencoder\_ChestPainType\_ASY', 'onehotencoder\_ChestPainType\_ATA',  
 'onehotencoder\_RestingECG\_LVH', 'onehotencoder\_ExerciseAngina\_N',  
 'onehotencoder\_ExerciseAngina\_Y', 'onehotencoder\_ST\_Slope\_Flat',  
 'onehotencoder\_ST\_Slope\_Up', 'remainder\_Age', 'remainder\_RestingBP',  
 'remainder\_Cholesterol', 'remainder\_FastingBS', 'remainder\_MaxHR',  
 'remainder\_Oldpeak'],  
 dtype='object')

Out[ ]: ([] , <a list of 0 Text major ticklabel objects>)



Tampilkan hasil uji model LogisticRegression terhadap test set yang sudah diseleksi fiturnya.

```

In [ ]:
X_train_rfe = selectRFE.transform(X_train_enc)
X_test_rfe = selectRFE.transform(X_test_enc)

score = LogisticRegression(max_iter=10000).fit(X_train_rfe, y_train) \
        .score(X_test_rfe, y_test)
print("Test score before feature selection: {:.3f}".format(LogisticRegression(max_iter=10000) \
        .fit(X_train_enc,y_train).score(X_test_enc,y_test)))
print("Test score after feature selection: {:.3f}".format(score))

```

Test score before feature selection: 0.877  
Test score after feature selection: 0.870