**Mudah Belajar Otodidak Data Science**

**(Praktek Menggunakan Python3)**

**Edisi 2 Tahun 2023**

**Disusun Oleh:**

**Aryajaya Alasmsyah, S.Kom., M.Kom., MTA.**

**Materi Praktikum Data Science**

1. Pengantar Bahasa Python
2. Visualisasi Data Menggunakan Matplotlib dan Seaborn
3. Visualisasi Data Menggunakan Framework Streamlit
4. Teknik-Teknik Praproses Data – Data Tabular, Time Series, dan Spasial
5. Algoritma Klasifikasi Data
6. Algoritma Klastering Data
7. Ujian Tengah Semester (UTS)
8. Algoritma Regresi Linier
9. Prediksi Cryptocurrency Menggunakan Algoritma SBi-LSTM dan SBi-GRU
10. Prediksi Cryptocurrency Menggunakan Algoritma XGBoost dan Prophet
11. Teknik Penggabungan metode SBi-LSTM-XGBoost dan SBi-GRU-XGBoost
12. Presentasi Projek
13. Presentasi Projek
14. Ujian Akhir Semester (UAS)

|  |
| --- |
| **Pertemuan 3 - Visualisasi Plotly dan Framework Streamlit** |

|  |
| --- |
| **Tujuan pembelajaran**   * Mahasiswa mampu menggunakan library seperti plotly dan framework streamlit. * Mahasiswa mampu membuat visualisasi data seperti barplot, lineplot, pieplot, histogram, scatterplot, boxplot, heatmap, dan lain-lain. |

**Studi Kasus: Visualisasi Iris Dataset**

**class\_data.py**

|  |
| --- |
| # lib manipulation data  import pandas as pd  import numpy as np    # lib praproses data  from sklearn.preprocessing import MinMaxScaler    # func getData by csv file  def getDataset(df):      # load dataset    dataset = pd.read\_csv("../../dataset/"+df)    return dataset    # func normalized by min-max method  def normalized(df):      # Set features and Labels    x = df[["sepal\_length","sepal\_width","petal\_length","petal\_width"]].values    y = df["species"].values      # prosess normalized data    scaler = MinMaxScaler(feature\_range=(-1, 1))    scaled = scaler.fit\_transform(x)      # convert numpy to pandas dataframe    results = pd.concat([      pd.DataFrame(scaled, columns=["sepal\_length","sepal\_width","petal\_length","petal\_width"]),      pd.DataFrame(y, columns=["species"]),    ], axis=1)      # return values    return results |

**class\_visualization.py**

|  |
| --- |
| # lib manipulation dat  import pandas as pd  import numpy as np    # lib visualization data  import plotly.express as px  # ---------------------------------------------------------------------------    # func build barplot  def barplot(df, title):      # calculate sum of species    data = pd.DataFrame(df.value\_counts()).reset\_index()      # create barplot    fig = px.bar(data, x="species", y="count", color="species")      # custom layout    fig.update\_layout(      title=title,      xaxis\_title="",      yaxis\_title="",      legend=dict(title='', orientation='h', yanchor='top', y=1.05, xanchor='center', x=0.5)    )      # return values    return fig  # ---------------------------------------------------------------------------    # func build heatmap coor  def heatmap(df, title):      # calculate correlation use pearson    z = df.corr(method="pearson", numeric\_only=True)      # create imshow    fig = px.imshow(      z, zmin=-1, zmax=1, text\_auto=True, aspect="auto",    )      # custom layout    fig.update\_layout(      title=title,      xaxis\_title="",      yaxis\_title="",    )      # return values    return fig  # ---------------------------------------------------------------------------    # func build scatter plot  def scatter(df, x, y, title):      # create scatter    fig = px.scatter(df, x=x, y=y, color="species")      # custom scatter    fig.update\_traces(      marker\_size=7    )      # custom layout    fig.update\_layout(      title=title,      xaxis\_title="",      yaxis\_title="",      legend=dict(title='', orientation='h', yanchor='top', y=1.05, xanchor='center', x=0.5)    )      # return values    return fig  # ---------------------------------------------------------------------------    # func build boxplot  def boxplot(df, x, y, title):      # create boxplot    fig = px.box(df, x=x, y=y, color="species")      # custom layout    fig.update\_layout(      title=title,      xaxis\_title="",      yaxis\_title="",      legend=dict(title='', orientation='h', yanchor='top', y=1.05, xanchor='center', x=0.5)    )      # return values    return fig  # ---------------------------------------------------------------------------    # func build histogram  def histogram(df, x, title):      # create histogram    fig = px.histogram(df, x=x, color="species")      # custom layout    fig.update\_layout(      title=title,      xaxis\_title="",      yaxis\_title="",      legend=dict(title='', orientation='h', yanchor='top', y=1.05, xanchor='center', x=0.5)    )      # return values    return fig  # --------------------------------------------------------------------------- |

**main.py**

|  |
| --- |
| # import library streamlit  import streamlit as st  import plotly.express as px  import plotly.graph\_objects as go    # import cumstom func  from class\_data import \*  from class\_visualization import \*    # config web streamlit  st.set\_page\_config(    page\_title="My Dasboard - Iris Dataset", layout="wide", initial\_sidebar\_state="auto",  ) |

|  |
| --- |
| # load dataset iris  dataset = getDataset("iris.csv")    # normalized data  dataset = normalized(dataset) |

|  |
| --- |
| # container-header  with st.container():    st.markdown("# Data Visualization of Iris Dataset")    # container-visualization data  with st.container():      # split two columns    col1, col2 = st.columns([1,1], gap="medium")      # col-barplot    col1.plotly\_chart(      barplot(dataset["species"], "Bar Chart to Find the Number of Classes"),      use\_container\_width=True    )      # col-heatmap    col2.plotly\_chart(      heatmap(dataset, "Heatmap Corr to calculate correlation between features"),      use\_container\_width=True    )    # --------------------------------------------------------------------------- |
|  |
|  |

|  |
| --- |
|  |
|  |
|  |

|  |
| --- |
|  |
|  |
|  |

|  |
| --- |
|  |
|  |
|  |

|  |
| --- |
|  |
|  |
|  |