**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 Bagian 1
6. Algoritma Klastering Data Bagian 2
7. Ujian Tengah Semester (UTS)
8. Algoritma Klastering Data Bagian 1
9. Algoritma Klastering Data Bagian 2
10. Algoritma Regresi Linier
11. Algoritma Neural Network: SBi-LSTM dan SBi-GRU
12. Algoritma Neural Network: SBi-LSTM-XGBoost dan SBi-GRU-XGBoostost
13. Presentasi Projek
14. Ujian Akhir Semester (UAS)

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

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

**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    ) | |
|  |  |
| Gambar x. Output program | |

|  |  |
| --- | --- |
| # col-scatter    col1.plotly\_chart(      scatter(        dataset, "petal\_length", "sepal\_length", "Scatterplot to see linearity between features"      ), use\_container\_width=True    )    # col-scatter    col2.plotly\_chart(      scatter(        dataset, "petal\_length", "petal\_width", "Scatterplot to see linearity between features"      ), use\_container\_width=True    )    # col-scatter    col1.plotly\_chart(      scatter(        dataset, "sepal\_length", "sepal\_width", "Scatterplot to see linearity between features"      ), use\_container\_width=True    )    # col-scatter    col2.plotly\_chart(      scatter(        dataset, "sepal\_length", "petal\_width", "Scatterplot to see linearity between features"      ), use\_container\_width=True    ) | |
|  |  |
|  |  |
| Gambar x. Output program | |

|  |  |
| --- | --- |
| # col-boxplot    col1.plotly\_chart(      boxplot(        dataset, "species", "sepal\_length", "Boxplot to see the outlier value in each feature"      ), use\_container\_width=True    )    # col-boxplot    col2.plotly\_chart(      boxplot(        dataset, "species", "sepal\_width", "Boxplot to see the outlier value in each feature"      ), use\_container\_width=True    )    # col-boxplot    col1.plotly\_chart(      boxplot(        dataset, "species", "petal\_length", "Boxplot to see the outlier value in each feature"      ), use\_container\_width=True    )    # col-boxplot    col2.plotly\_chart(      boxplot(        dataset, "species", "petal\_width", "Boxplot to see the outlier value in each feature"      ), use\_container\_width=True    )    # --------------------------------------------------------------------------- | |
|  |  |
|  |  |
| Gambar x. Output program | |

|  |  |
| --- | --- |
| # col-histogram    col1.plotly\_chart(      histogram(        dataset, "sepal\_length", "Histogram to see the distribution of data between features"      ), use\_container\_width=True    )    col2.plotly\_chart(      histogram(        dataset, "sepal\_width", "Histogram to see the distribution of data between features"      ), use\_container\_width=True    )    col1.plotly\_chart(      histogram(        dataset, "petal\_length", "Histogram to see the distribution of data between features"      ), use\_container\_width=True    )    col2.plotly\_chart(      histogram(        dataset, "petal\_width", "Histogram to see the distribution of data between features"      ), use\_container\_width=True    ) | |
|  |  |
|  |  |
| Gambar x. Output program | |

**Studi Kasus: Visualisasi Cryptocurrency and Stock Price**

**class\_data.py**

|  |
| --- |
| # lib getdataset from yahoo  import yfinance as yf    # lib manipulation dataset  import pandas as pd  import numpy as np    # func getData by csv file  def getData(ticker, startDate, endDate):      # set the ticker and datetime    dataset = ticker.history(start=startDate, end=endDate).reset\_index()      # set the feature    dataset = dataset[["Date", "Open", "High", "Low", "Close"]]      # convert datetime format    dataset["Date"] = pd.to\_datetime(dataset["Date"], format="%d-%m-%Y")      # return values    return dataset |

**class\_visualization.py**

|  |
| --- |
| # lib visualization data  import plotly.express as px  import plotly.graph\_objects as go    # func build heatmap coor  def timeseries\_plot(df, title):      # create a plot    fig = go.Figure()      # add lineplot with graph object    for column in df.columns[1:]:      fig.add\_trace(        go.Scatter(            x=df["Date"],y=df[column], mode='lines', name=column        )      )      # add colors on lineplot    colorscale = px.colors.diverging.Portland\_r    for i, trace in enumerate(fig.data):      trace.update(line=dict(color=colorscale[i]))      # update layout lineplot    fig.update\_layout(      title = "Data Visualization of "+str(title),      xaxis\_title = "",      yaxis\_title = "",      legend=dict(title='', orientation='h', yanchor='top', y=1.1, xanchor='center', x=0.5),    )      # return values    return fig |

**main.py**

|  |
| --- |
| # lib getdataset from yahoo  import yfinance as yf    # import library streamlit  import streamlit as st    # lib visualization data  import plotly.express as px  import plotly.graph\_objects as go    # import custom func  from class\_data import \*  from class\_visualization import \*    # config web streamlit  st.set\_page\_config(    page\_title="My Dasboard - Cryptocurrency", layout="wide", initial\_sidebar\_state="auto",  )    # container-header  with st.container():    st.markdown("# Visualization Data of Cryptocurrency and Stock Price")    # split two columns  col1, col2 = st.columns([0.3, 0.7], gap="small")    # col1 - config dataset  with col1:      # set labels    st.success("Config Dataset")      with st.form("my-form"):      cryptocurrency = st.selectbox(        "Choose a cryptocurrency", ("BTC-USD", "ETH-USD", "AMZN", "AAPL", "GOOG", "MSFT"),        placeholder="Choose a cryptocurrency", index=None      )      start = st.date\_input(        label="Start Date", value=None, min\_value=None, max\_value=None,      )      end = st.date\_input(        label="End Date", value=None, min\_value=None, max\_value=None,      )      submit = st.form\_submit\_button(label="Submit", type="secondary", use\_container\_width=True)    # col1 – Developer Info  with col1:    st.info("Created by Aryajaya Alamsyah")    # col1 - Exploration Data Analysis  with col2:      # set labels    st.success("Exploration Data Analysis")      # Set Default dataset    if submit:      # BTC-USD, ETH-USD, AMZN, AAPL, GOOG, MSFT      ticker      = yf.Ticker(cryptocurrency)      startDate   = start      endDate     = end    else:      # BTC-USD, ETH-USD, AMZN, AAPL, GOOG, MSFT      ticker      = yf.Ticker("BTC-USD")s      startDate   = "2015-01-01"      endDate     = "2024-06-01"      # load dataset    dataset = getData(ticker, startDate, endDate)      # load timeseries plot    st.plotly\_chart(timeseries\_plot(dataset, cryptocurrency)) |
|  |
| Gambar x. Output program |

**Selesai, Selamat Mencoba :3**