Below is a detailed explanation of your code ‘**Portfolio\_streamlit5.py’** , including each section and the libraries used:

1. **Importing Libraries**

import streamlit as st

import yfinance as yf

import pandas as pd

import matplotlib.pyplot as plt

import plotly.graph\_objects as go

* **streamlit**: Used to create interactive web applications. Here, it's used for the dashboard UI.
* **yfinance**: A Python library for fetching stock market data from Yahoo Finance.
* **pandas**: For data manipulation and analysis.
* **matplotlib.pyplot**: For creating static visualizations.
* **plotly.graph\_objects**: For interactive visualizations (although it's not utilized fully in this code).

1. **Streamlit Page Configuration**

st.set\_page\_config(page\_title="Stock Portfolio Dashboard", layout="wide")

* Configures the Streamlit app's title and layout.
* page\_title: Sets the browser tab title.
* layout="wide": Expands the app to full-width for better visibility.

1. **Dashboard Title**

st.title("Stock Portfolio Analysis Dashboard")

st.write("Analyze the performance of stocks from the Indian stock market with daily and cumulative returns.")

* st.title: Displays the main title of the app.
* st.write: Adds a brief description below the title.

1. **User Inputs**

st.sidebar.header("User Inputs")

tickers = st.sidebar.text\_input("Enter stock tickers (comma-separated):", "TCS.NS, INFY.NS")

start\_date = st.sidebar.date\_input("Start Date", value=pd.to\_datetime("2023-01-01"))

end\_date = st.sidebar.date\_input("End Date", value=pd.to\_datetime("2023-12-31"))

* **Inputs via Sidebar**:
  + **Ticker Symbols**: User enters stock tickers (e.g., "TCS.NS, INFY.NS").
  + **Start Date & End Date**: User selects a date range for analysis.

1. **Fetch and Process Data**

if st.sidebar.button("Analyze"):

* Triggers the data fetching and processing when the "Analyze" button is clicked.

**Ticker Input Validation**

if tickers:

tickers\_list = [ticker.strip() for ticker in tickers.split(",")]

* Splits the user-provided ticker string into a list of individual ticker symbols.

**Data Fetching**

data = yf.download(ticker, start=start\_date, end=end\_date)

* Downloads historical stock data using the Yahoo Finance API for the given ticker and date range.

**Price Selection**

if 'Adj Close' in data.columns:

data['Price'] = data['Adj Close']

elif 'Close' in data.columns:

data['Price'] = data['Close']

else:

st.warning(f"No valid price data found for {ticker}. Skipping.")

* **Adj Close**: Adjusted closing price, accounting for dividends and splits.
* **Close**: Unadjusted closing price.
* **Fallback**: Skips tickers without valid price data.

**Data Processing**

data['Daily Return'] = data['Price'].pct\_change()

data['Cumulative Return'] = (1 + data['Daily Return']).cumprod()

* **Daily Return**: Percentage change between consecutive days.
* **Cumulative Return**: Product of daily returns, showing growth over time.

1. **Data Visualization**

for ticker, data in processed\_data.items():

st.subheader(ticker)

* Iterates through each stock ticker and its corresponding data for visualization.

**Price Trend Plot**

fig, ax = plt.subplots(figsize=(10, 5))

ax.plot(data.index, data['Price'], label='Price')

ax.set\_title(f"{ticker} Price Trend")

ax.set\_xlabel("Date")

ax.set\_ylabel("Price (INR)")

ax.legend()

st.plotly\_chart(fig, use\_container\_width=True)

* Creates a line plot of stock prices over time.
* **st.plotly\_chart**: Displays the plotly version of the chart (misused here; should use st.pyplot for matplotlib).

**Cumulative Return Plot**

fig, ax = plt.subplots(figsize=(10, 5))

ax.plot(data.index, data['Cumulative Return'], label='Cumulative Return', color='green')

ax.set\_title(f"{ticker} Cumulative Return")

ax.set\_xlabel("Date")

ax.set\_ylabel("Cumulative Return")

ax.legend()

st.plotly\_chart(fig, use\_container\_width=True)

* Creates a line plot for cumulative returns.

1. **Error Handling**

except Exception as e:

st.error(f"Error fetching or processing data for {ticker}: {e}")

* Catches and displays errors encountered during data fetching or processing.

1. **Footer**

st.sidebar.write("Developed by Manas Bhise - Stock Portfolio Analysis")

* Adds a footer with developer attribution.

**Summary of Key Points**

1. **Streamlit** provides the interface for user interaction and visualization.
2. **Yfinance** fetches real-time stock data.
3. **Pandas** processes the data for daily and cumulative returns.
4. **Matplotlib** generates static plots (but st.pyplot should replace st.plotly\_chart for correct usage).
5. **Error Handling** ensures user-friendly error messages.