Foreign exchange, often abbreviated as forex or FX, refers to the global marketplace where currencies are bought and sold. The forex market is the largest and most liquid financial market in the world, facilitating the exchange of currencies between participants, including governments, financial institutions, corporations, and individual traders.

Here are some key aspects to understand about foreign exchange:

1. **Currency Pairs:**
    - In forex trading, currencies are quoted in pairs, such as EUR/USD (Euro/US Dollar) or USD/JPY (US Dollar/Japanese Yen).
    - The first currency in the pair is called the "base currency," and the second is the "quote currency." The exchange rate represents how much of the quote currency is needed to purchase one unit of the base currency.
2. **Exchange Rate:**
    - The exchange rate is the price of one currency in terms of another. It can be expressed in two ways: direct quote and indirect quote.
    - A direct quote expresses the value of one unit of the domestic currency in terms of the foreign currency. An indirect quote expresses the value of one unit of the foreign currency in terms of the domestic currency.
3. **Market Participants:**
    - The forex market is decentralized and operates 24 hours a day, five days a week. Major participants include central banks, commercial banks, hedge funds, multinational corporations, and individual traders.
    - Central banks play a significant role in influencing exchange rates through monetary policy decisions and interventions.
4. **Speculation and Hedging:**
    - Participants engage in forex trading for various reasons, including speculation and hedging.
    - Speculators aim to profit from currency price movements. They buy a currency pair anticipating its value will rise (going long) or sell it expecting a decline (going short).
    - Hedgers, such as multinational corporations, use the forex market to protect themselves from adverse currency movements by entering into positions that offset potential losses.
5. **Leverage:**
    - Forex trading often involves the use of leverage, allowing traders to control a larger position with a relatively small amount of capital. While leverage magnifies potential profits, it also increases the risk of significant losses.
6. **Forex Brokers:**
    - Individual traders typically access the forex market through brokers. These brokers provide a platform for trading and facilitate transactions between buyers and sellers.
    - Forex brokers may offer different types of accounts, trading platforms, and leverage options.
7. **Factors Influencing Exchange Rates:**

- Exchange rates are influenced by various factors, including economic indicators (such as interest rates, inflation, and economic growth), geopolitical events, market sentiment, and central bank actions.

The motivation behind creating this application appears to be providing a user-friendly and interactive tool for technical analysis of forex (foreign exchange) data. Here are some potential motivations:

1. **Accessibility for Non-Technical Users**: The application is built using Streamlit, a framework designed to simplify the process of creating web applications with Python. This makes it accessible to users who may not have extensive programming or data analysis skills.
2. **Customizable Technical Analysis**: The application allows users to input parameters such as currency pairs, date range, and select from a variety of technical indicators. This customization empowers users to perform tailored technical analysis based on their preferences and trading strategies.
3. **Visualization of Technical Indicators**: The graphical representation of candlestick charts along with various technical indicators like Moving Averages, RSI, MACD, etc., provides users with a visual understanding of the market trends and potential entry or exit points.
4. **Real-Time Data Analysis**: While not explicitly mentioned in the provided code, the use of libraries like `yfinance` suggests the potential for real-time or near-real-time data analysis. This can be valuable for traders who rely on up-to-date information for decision-making.
5. **Educational Purposes**: The application can serve as an educational tool for individuals looking to learn about technical analysis in the context of forex trading. The interactive nature of the app allows users to experiment with different indicators and observe their impact on the charts.
6. **Decision Support Tool**: Traders often use technical analysis to make informed decisions about buying or selling assets. This application can serve as a decision support tool by providing a consolidated platform for analyzing multiple indicators simultaneously.
7. **Data Export Functionality**: The inclusion of a data export functionality, allowing users to download the forex data as a CSV file, suggests an intention to enable users to perform further analysis or integrate the data into other tools.
8. **Responsive Design**: The use of Streamlit's features, such as tabs and expanders, suggests an effort to create a responsive and organized user interface, making it easier for users to navigate and utilize the various features of the application.

Technical Analysis is a method used in finance and investment to evaluate and forecast future price movements of financial instruments, such as stocks, currencies, commodities, and other assets. It involves studying historical price data, trading volume, and other market-related statistics to identify patterns and trends. Here are key components and reasons why technical analysis is used:

1. **Price Patterns and Trends:**
    - Technical analysts believe that historical price movements tend to repeat themselves. They analyze charts and patterns, such as head and shoulders, triangles, and trend lines, to identify potential future price movements.
2. **Market Indicators:**
    - Various technical indicators are used to assess the strength and direction of a trend. Examples include moving averages, relative strength index (RSI), moving average convergence divergence (MACD), and stochastic oscillators.
3. **Support and Resistance Levels:**
    - Analysts identify levels at which an asset's price has historically had difficulty moving above (resistance) or below (support). These levels are believed to impact future price movements.
4. **Volume Analysis:**
    - Trading volume is considered alongside price movements to validate trends. High volume during an upward or downward movement may suggest the strength of that trend.
5. **Psychological Factors:**
    - Technical analysis takes into account investor psychology and market sentiment. For example, if a stock repeatedly fails to break through a certain resistance level, it may be due to psychological factors affecting investor behavior.
6. **Timing Entry and Exit Points:**
    - Traders use technical analysis to time their entry and exit points in the market. Indicators and patterns are interpreted to identify potential buying or selling opportunities.
7. **Risk Management:**
    - Technical analysis can help traders and investors manage risk by setting stop-loss orders and identifying points at which a trade may be invalidated.
8. **Algorithmic Trading:**
    - Many quantitative trading strategies and algorithmic trading systems rely on technical analysis. Automated trading systems often use predefined technical indicators and rules to execute trades.
9. **Short-Term Trading:**
    - Technical analysis is particularly popular among short-term traders, such as day traders and swing traders, who aim to profit from short-term price movements.
10. **Complementary to Fundamental Analysis:**
    - Technical analysis is often used in conjunction with fundamental analysis. While fundamental analysis focuses on the intrinsic value of an asset, technical analysis is more concerned with historical price data and market trends.
11. **Availability of Historical Data:**

- Technical analysis relies on historical price data, which is widely available for various financial instruments. Traders use this historical data to identify patterns and trends that may repeat in the future.

1. User Interface Setup:

- The "Raw Data" tab is part of the Streamlit application and can be accessed by users along with the "Graph" tab.

2. DataFrame Generation:

- In the `ta_analysis_DB` function, after the technical analysis section (`with tabb:`), the code extracts the historical forex data (`forex_data`) and assigns it to the variable `df`. The data is then converted to a CSV format using the `convert_df` function.

3. Download Button:

- The `st.download_button` function is used to create a button labeled "Download Data." When clicked, this button allows users to download the raw historical forex data in CSV format. The parameters of the download button include the button label, the data to be downloaded (`csv`), the suggested filename (`f"{interval}_{currencypair1}{currencypair2}.csv"`), the file type (`"text/csv"`), and a key identifier (`key='download-csv'`).

4. Data Table Display:

- Below the download button, the `st.table` function is used to display a table of the historical forex data (`forex_data`). The table is created by transposing the DataFrame (`forex_data.T[:-1].T`) to present it in a more readable format.

5. DataFrame Conversion Function:

- The `convert_df` function takes a DataFrame (`df`) and converts it to a CSV format encoded in UTF-8. This function is used when creating the download button.

6. Data Accessibility:

- Users can explore the raw data table to inspect the details of the historical price, volume, and other relevant information for the selected currency pair and time period.

Summary:

The "Raw Data" feature provides users with direct access to the raw historical forex data that serves as the input for their backtesting analysis. Users can download the data in CSV format for

further analysis or use it outside of the Streamlit application. This feature enhances transparency and allows users to verify and explore the dataset used in their backtesting process.

The "Graph" feature in the provided Streamlit application is a visual representation of the technical analysis of a selected currency pair based on user-defined parameters. Here's an explanation of how the "Graph" feature works:

1. User Interface Setup:

- The "Graph" tab is part of the Streamlit application and can be accessed by users along with the "Raw Data" tab.

2. Graphical Representation:

- The main graph is generated using the Plotly library (`plotly.graph_objects`). It displays a candlestick chart of the historical forex data, including open, high, low, and close prices.

3. Technical Analysis Indicators:

- Users can customize the technical analysis indicators displayed on the graph through the sidebar input parameters. These parameters include currency pair selection, start and end dates, data interval (daily, weekly, monthly, or hourly), and various technical indicators such as moving averages, ADX, RSI, Stochastic Oscillator, Bollinger Bands, MACD, CCI, Pivot Points, and Parabolic SAR.

4. User-Defined Input Parameters:

- Users can input their preferred currency pair (e.g., EUR/USD), start and end dates for the historical data, and choose the data interval (daily, weekly, monthly, or hourly). Additionally, users can select and customize various technical indicators and their parameters through the Streamlit interface.

5. Dynamic Updating:

- The graph dynamically updates based on the user's input parameters. When users make changes to the input parameters, the corresponding technical analysis indicators are recalculated, and the graph is updated accordingly.

6. Subplots for Indicators:

- The graph includes subplots for each selected technical indicator. For example, there are separate subplots for moving averages, ADX, RSI, Stochastic Oscillator, Bollinger Bands, MACD, CCI, Pivot Points, and Parabolic SAR.

7. Full Graph View Option:

- Users have the option to view the full graph or focus on a specific time range. There is a checkbox labeled "View Full Graph," and if unchecked, the graph adjusts the x-axis range based on the selected data interval.

8. Visual Indicators:

- Visual indicators are added to the graph to represent the calculated values of the technical indicators, such as lines for moving averages, dots for pivot points, and markers for Parabolic SAR.

Summary:

The "Graph" feature provides users with an interactive and customizable visualization of the technical analysis of a selected currency pair. Users can analyze historical price movements, identify trends, and assess the impact of various technical indicators on the forex data. The dynamic and user-friendly interface enhances the exploration and understanding of the technical aspects of the selected currency pair.

1. Input Parameters:

1.1 Currency Pairs:

Currency pairs refer to the combination of two different currencies in the foreign exchange market. In trading and analysis, understanding how one currency performs against another is crucial. For example, in the pair EUR/USD, the Euro is compared to the US Dollar. Currency pairs are fundamental for forex trading and analysis.

1.2 Date Range:

The date range is the period of time selected for analysis. It is essential to define a specific time frame to gain meaningful insights. Whether you're analyzing historical data or planning for future predictions, setting the date range allows you to focus on a particular segment of time.

1.3 Interval:

Interval refers to the time increments used in data presentation. Common intervals include minutes, hours, days, weeks, etc. The chosen interval impacts the granularity of the data. Short intervals provide detailed, real-time information, while longer intervals smooth out fluctuations and reveal broader trends.

2. Moving Averages:

2.1 Simple Moving Averages (SMA):

SMA is a calculation that takes the average closing prices over a specified period. It is a lagging indicator, smoothing out price data to create a single flowing line. Traders use SMAs to identify trends, confirm trend reversals, and generate buy or sell signals.

2.2 Exponential Moving Averages (EMA):

EMA gives more weight to recent prices, making it more responsive to current market conditions compared to SMA. It reacts faster to price changes, making it suitable for traders who want to capture short-term trends. EMA is often preferred in fast-moving markets.

3. Trend Indicators:

3.1 Parabolic SAR:

Parabolic SAR (Stop and Reverse) is a trend-following indicator that provides potential reversal points. It appears as dots on a chart, above or below the price, indicating potential trend shifts. Traders may use it to set stop-loss levels or identify when a trend might be losing momentum.

3.2 Average Directional Index (ADX):

ADX is used to quantify the strength of a trend. It doesn't specify the direction but rather the strength of the trend. A high ADX suggests a strong trend, while a low ADX indicates a weak trend. Traders use ADX to filter out ranging markets and focus on trending conditions.

4. Momentum Indicators:

4.1 Relative Strength Index (RSI):

RSI measures the magnitude of recent price changes to evaluate overbought or oversold conditions in a market. RSI ranges from 0 to 100, with readings above 70 indicating overbought conditions and readings below 30 indicating oversold conditions.

4.2 Stochastic Oscillator:

The Stochastic Oscillator compares a security's closing price to its price range over a given time period. It generates values between 0 and 100, helping identify potential overbought or oversold conditions. Crosses above 80 indicate overbought, while crosses below 20 indicate oversold.

5. Volatility Indicators:

5.1 Bollinger Bands:

Bollinger Bands consist of a middle band being an N-period simple moving average and upper and lower bands representing standard deviations from the moving average. They visually depict volatility, expanding during volatile periods and contracting during stable ones.

6. Oscillators:

6.1 Moving Average Convergence Divergence (MACD):

MACD is a trend-following momentum indicator that shows the relationship between two moving averages of a security's price. It consists of a MACD line, a signal line, and a histogram. MACD crossovers and histogram patterns are used to generate buy or sell signals.

6.2 Commodity Channel Index (CCI):

CCI measures the statistical variation from the average. It is often used to identify overbought or oversold conditions in a market. Readings above +100 indicate overbought conditions, while readings below -100 indicate oversold conditions.

7. Support and Resistance Indicators:

7.1 Pivot Points:

Pivot Points are technical indicators used to identify potential levels of support and resistance. They are calculated based on the previous day's high, low, and close prices. Traders use pivot points to determine potential entry and exit points.

These indicators and parameters collectively provide traders and analysts with tools to make informed decisions based on historical and real-time market data. The combination of these elements allows for a comprehensive analysis of market trends, momentum, volatility, and potential support/resistance levels.

1. **Moving Averages (SMA and EMA):**
    - In the sidebar, provide input fields for users to enter SMA and EMA parameters.
2. **Average Directional Index (ADX):**
    - Allow users to input ADX window in the sidebar.
3. **Relative Strength Index (RSI):**
    - Allow users to input RSI window in the sidebar.
4. **Stochastic Oscillator:**
    - Provide sliders or input fields for users to set K and D periods.
5. **Bollinger Bands:**
    - Allow users to set Bollinger Bands window and standard deviation using input fields.
6. **Moving Average Convergence Divergence (MACD):**

    - Provide input fields for users to set short-term, long-term, and signal window parameters.
7. **Commodity Channel Index (CCI):**
    - Allow users to input CCI window in the sidebar.
8. **Pivot Points:**
    - Since Pivot Points are calculated without additional parameters, you can provide a checkbox to enable/disable them.
9. **Parabolic SAR:**
    - Provide sliders or input fields for users to set SAR acceleration and maximum parameters.

The backtesting page in the provided code allows users to test trading strategies on historical data to evaluate their performance. It utilizes the `backtrader` library, which is a popular Python framework for backtesting trading strategies.

Components of the Backtesting Page:

1. **Settings Sidebar:**
    - Users can set the base currency, quote currency, start date, and end date for the historical data to be used in the backtest.
    - Users can also select the time interval (Daily, Weekly, Monthly, Hourly) for the data.
2. **Editor for Strategy Code:**
    - Users can input their custom trading strategy code in the editor. This is where the `st_ace` function from `streamlit_ace` is used to provide an interactive code editor.
3. **Example Strategy Code:**
    - The page provides three example strategy codes that users can load and use for their backtests. These examples cover moving average crossover, RSI, and mean reversion Bollinger Bands strategies.
4. **Maximum Balance and Desired Balance:**
    - Users can set their maximum balance and select a desired balance for the backtest.
5. **Backtest Button:**
    - Clicking the "Backtest" button runs the backtest using the specified strategy and parameters.

How to Use the Editor and Example Codes:

1. **Editor:**
    - Users can write their custom strategy code in the editor. This code should define a class inheriting from `bt.Strategy` and implement the `__init__` and `next` methods.
    - The `__init__` method is used to initialize strategy parameters and indicators.
    - The `next` method is called for each trading day and contains the logic for making buy/sell decisions.
2. **Example Codes:**
    - Three example strategies are provided: Moving Average Crossover, RSI, and Mean Reversion Bollinger Bands.

    - To use an example strategy, click on the respective expander to reveal the code.
    - Copy and paste the code into the editor to load the example strategy.
3. **Running a Backtest:**
    - After setting the parameters and entering the strategy code, click the "Backtest" button.
    - The backtest results, including starting and ending portfolio values, are displayed.
    - A plot of the portfolio value over time is shown.

Example:

```
python
# Example Strategy (Moving Average Crossover):
class MyStrategy(bt.Strategy):
    params = (
        ('short_period', 20),
        ('long_period', 50),
        ('position_size', 0.1),
    )

    def __init__(self):
        self.short_ma = bt.indicators.SimpleMovingAverage(self.data.close,
period=self.params.short_period)
        self.long_ma = bt.indicators.SimpleMovingAverage(self.data.close,
period=self.params.long_period)

    def next(self):
        if self.position:
            if self.short_ma < self.long_ma:
                self.sell()
        else:
            if self.short_ma > self.long_ma:
                self.buy()
```

Users can modify the example code or write their own strategies to test different trading approaches. The backtesting framework provides insights into how a strategy would have performed historically.

Devlopment Duration: The development of this application spanned over two and a half weeks. It was an engaging and enjoyable project that involved integrating various technologies to create a feature-rich tool for technical analysis and backtesting.

Fun Project: Working on this project was a fun and rewarding experience. Exploring financial data, implementing technical indicators, and creating an interactive backtesting environment added a dynamic element to the development process.

Potential Future Improvements: While the current version provides a robust set of features, there are always opportunities for enhancement. Some potential future improvements include:

- **User Authentication:** Implementing user authentication to allow users to save and load their custom strategies and analysis settings.

- **More Technical Indicators:** Adding support for additional technical indicators to provide users with a broader range of tools for analysis.
- **Optimization:** Optimizing the application for performance, especially when dealing with a large dataset or complex strategies.
- **Educational Resources:** Incorporating educational resources and tooltips to help users understand the significance of different technical indicators and how to interpret backtesting results.

These potential improvements can contribute to making the application more user-friendly, informative, and versatile for both novice and experienced traders.

Importing Libraries

```
python
import yfinance as yf
import pandas as pd
import numpy as np
import streamlit as st
import plotly.graph_objects as go
from datetime import datetime, timedelta
import simple_TA_lib
```

- Importing necessary libraries for data manipulation, visualization, and the Streamlit web app framework.
- `yfinance` is used for fetching financial data, `pandas` for data manipulation, `numpy` for numerical operations, `streamlit` for building the web app, and `plotly` for creating interactive plots.
- `simple_TA_lib` appears to be a custom module for calculating technical indicators.

Setting Default Dates and Streamlit Page Configuration

```
python
today = datetime.today()
default_start_date = today.replace(day=1)
default_end_date = today + timedelta(days=1)

st.set_page_config(
    page_title="Analysis",
    page_icon="🔍",
    layout="wide",  # Set the layout to wide
)
```

- Setting default start and end dates for the analysis.
- Configuring Streamlit page with a title, icon, and wide layout.

Streamlit Analysis Dashboard Function

```
python
def ta_analysis_DB():
    taba, tabb = st.tabs(['Graph', 'Raw Data'])
    with taba:
        # ... (Tab A content)
    with tabb:
        # ... (Tab B content)

    ta_analysis_DB()
```

- Defining a function `ta_analysis_DB` that represents the entire analysis dashboard.
- Creating two tabs: 'Graph' and 'Raw Data' using Streamlit's `st.tabs`.
- The function is then called to run the analysis dashboard.

Input Parameters Tab

```
python
with st.sidebar:
    tab1, tab2 = st.tabs(['Input Parameters', 'Technical Indicators'])
with tab1:
    # ... (Input parameters form and session state initialization)
```

- Using Streamlit's sidebar to create tabs for 'Input Parameters' and 'Technical Indicators'.
- Within 'Input Parameters', users can input currency pairs, date range, and select the data interval.
- Initializing session state variables to store selected moving averages, exponential moving averages, and ADX values.

Technical Indicators Tab

```
python
with tab2:
    # ... (Sections for SMA, EMA, Parabolic SAR, ADX, RSI, Stochastic,
Bollinger Bands, MACD, CCI)
```

- Creating a tab for 'Technical Indicators'.
- Users can enable/disable various technical indicators such as SMA, EMA, Parabolic SAR, ADX, RSI, Stochastic Oscillator, Bollinger Bands, MACD, and CCI.
- For each indicator, users can set parameters and add them to the chart.

Retrieving Forex Data and Displaying Graph

```
python
symbol = f'{currencypair1}{currencypair2}=X'
forex_data, close_prices = simple_TA_lib.get_forex_data(symbol, start_date,
end_date, interval)
fig_candlestick = go.Figure(data=[go.Candlestick(x=forex_data.index,
open=forex_data['Open'], high=forex_data['High'], low=forex_data['Low'],
close=forex_data['Close'])])
```

- Constructing a symbol for the currency pair.

- Fetching forex data using the `get_forex_data` function from the `simple_TA_lib`.
- Creating a Candlestick chart using Plotly's `go.Figure`.

Calculating and Displaying Technical Indicators

```
python
# ... (For loops calculating and adding selected indicators to the charts)
```

- Iterating through selected moving averages, exponential moving averages, ADX, RSI, MACD, Stochastic Oscillator, Bollinger Bands, CCI, and other indicators.
- Calculating each indicator using functions from `simple_TA_lib`.
- Adding traces for each indicator to the respective figures.

Updating Layouts of Charts

```
python
# ... (Updating layout for each chart)
```

- Updating the layout of each chart with title, axis labels, and other relevant information.

Displaying Charts

```
python
st.plotly_chart(fig_candlestick, use_container_width=True)
# ... (Additional charts for ADX, RSI, Stochastic, MACD, CCI)
```

- Using Streamlit's `st.plotly_chart` to display the candlestick chart and other charts.
- Charts are displayed in the Streamlit app with an option to view the full graph.

Downloading Data

```
python
with tabb:
    df = forex_data
    def convert_df(df):
        return df.to_csv(index=False).encode('utf-8')

    csv = convert_df(df)

    st.download_button(
        "Download Data",
        csv,
        f"{interval}_{currencypair1}{currencypair2}.csv",
        "text/csv",
        key='download-csv'
    )
    st.table(forex_data.T[:-1].T)
```

- In the 'Raw Data' tab, providing a download button to export the data to a CSV file.
- Displaying the data in tabular form using `st.table`.

This breakdown covers the major sections of the code. Each section is responsible for a specific aspect of the analysis dashboard, including input parameters, technical indicators, data retrieval, chart creation, and data presentation.

Importing Libraries

```
python
import streamlit as st
import backtrader as bt
import yfinance as yf
from streamlit_ace import st_ace
import matplotlib
import pandas as pd

matplotlib.use('Agg')
```

- Importing necessary libraries for building the Streamlit app, backtesting with Backtrader, fetching financial data with Yahoo Finance, using the Streamlit Ace code editor, and handling data with Pandas.

Streamlit Page Configuration

```
python
st.set_page_config(
    page_title="Backtesting",
    page_icon="📈",
    layout="wide",  # Set the layout to wide
)
```

- Configuring Streamlit page with a title, icon, and wide layout.

Function for Fetching Forex Data

```
python
def get_forex_data(base_currency, quote_currency, start_date, end_date,
interval):
    # ... (fetching forex data using yfinance)
    return forex_data
```

- Defining a function `get_forex_data` to fetch forex data using Yahoo Finance based on provided parameters (base currency, quote currency, start date, end date, and interval).

Streamlit Sidebar for User Input

```
python
st.sidebar.header("Settings")
base_currency = st.sidebar.text_input("Base Currency", "USD")
quote_currency = st.sidebar.text_input("Quote Currency", "EUR")
start_date = st.sidebar.date_input("Start Date", pd.to_datetime("2023-01-
01"))
end_date = st.sidebar.date_input("End Date", pd.to_datetime("2024-01-01"))
```

```
selected_interval = st.sidebar.selectbox('Select Interval', interval_options,
index=0)
interval = io_decoder[selected_interval]
```

- Creating a sidebar in the Streamlit app for user input settings.
- Users can input base and quote currencies, select date range, and choose the data interval.

Streamlit Ace Code Editor

```
python
default_code = """
class MyStrategy(bt.Strategy):
    # Enter strategy code now
"""

code = st_ace(default_code, language='python')
if st.checkbox('Apply Code', True):
    exec(code)
```

- Using the Streamlit Ace code editor to allow users to input and edit Python code.
- Providing a default strategy code and the ability to apply the entered code.

Example Strategy Code Expanders

```
python
with st.expander("Example Strategy (Moving Average Crossover)"):
    st.code('''class MyStrategy(bt.Strategy):
    # ... (Example moving average crossover strategy code)
    ''')

with st.expander("Example Strategy (RSI)"):
    st.code('''class MyStrategy(bt.Strategy):
    # ... (Example RSI strategy code)
    ''')

with st.expander("Example Strategy (Mean Reversion Bollinger Bands)"):
    st.code('''class MyStrategy(bt.Strategy):
    # ... (Example mean reversion Bollinger Bands strategy code)
    ''')
```

- Providing expanders with example strategy code for Moving Average Crossover, RSI, and Mean Reversion Bollinger Bands.
- Users can expand each section to view and potentially modify the example strategy code.

Backtesting Section

```
python
balMax = st.sidebar.number_input('Enter You Maximum Balance', value=1000)
balance = st.sidebar.slider("Select Desired Balance", min_value=1,
max_value=balMax, value=int(balMax/2))
```

- Adding a slider in the sidebar for selecting the desired balance for backtesting.

Backtest Function

```
python
def backtest():
    # ... (function for running backtest and displaying results)
    pass

if st.button('Backtest'):
    backtest()
```

- Defining a function `backtest` for running the backtest using Backtrader and displaying the results.
- A button is provided in the Streamlit app to trigger the backtest when clicked.

Backtest Results Display

```
python
st.subheader("Backtest Results")
st.write(f"Starting Portfolio Value: ${cerebro.broker.startingcash:.2f}")
st.write(f"Ending Portfolio Value: ${cerebro.broker.getvalue():.2f}")

# Plot the backtest results
figure = cerebro.plot()[0][0]
st.pyplot(figure)
```

- Displaying backtest results such as starting and ending portfolio values.
- Plotting the backtest results using Matplotlib and displaying the plot in the Streamlit app.