Report:

Introduction

This report details a Python script designed to analyze financial data from a PostgreSQL database, focusing on Nifty tick data and option chain tick data. The script performs various operations, including fetching data, cleaning it, generating statistics, filtering based on criteria, and visualizing results.

Objectives

- Fetch and analyze financial data from a PostgreSQL database.
- Perform descriptive statistics on Nifty and option chain data.
- Clean the dataset by removing inconsistencies.
- Visualize key metrics and identify potential trading strategies.

Approach

1. Setting Up the Environment

- **Libraries**: The script uses several libraries, including:
  - pandas: For data manipulation and analysis.
  - matplotlib.pyplot: For plotting graphs.
  - sqlalchemy: For database connections.
  - bcsdbconfig: A custom module for database configuration.
- **Database Connection**: The script establishes a connection to a PostgreSQL database using SQLAlchemy.

```
engine = create_engine(f'postgresql+psycopg2://{bcsdbconfig.user}:{bcsdbconfig.password}@{bcsdbconfig.host}:{bcsdbconfig.port}/{bcsdbconfig.database}')
```

2. Data Fetching

- Data is fetched from two tables: nifty_tick_data and option_chain_tick_data using SQL queries.

```
nifty_data = pd.read_sql_query('SELECT * FROM nifty_tick_data', engine)

option_chain_data = pd.read_sql_query('SELECT * FROM option_chain_tick_data', engine)
```

3. Descriptive Statistics

- The script calculates descriptive statistics for relevant columns in both datasets, providing insights into their distributions.

```
nifty_stats = nifty_data[['Open', 'Close', 'Volume']].describe()

option_stats = option_chain_data[['Strike Price', 'Volume', 'Open Interest']].describe()
```

4. Volume Calculation

- The total volume of options is calculated by grouping the data by Option Type.

```
total_volume = option_chain_data.groupby('Option Type')['Volume'].sum().reset_index()
```

5. Data Cleaning

- Rows with missing values are removed from the option_chain_data.

```
option_chain_data_cleaned = option_chain_data.dropna()
```

6. Date Formatting

- The Timestamp column is converted to a datetime format and sorted.

```
option_chain_data_cleaned['Timestamp'] =
pd.to_datetime(option_chain_data_cleaned['Timestamp'])

option_chain_data_cleaned = option_chain_data_cleaned.sort_values(by='Timestamp')
```

7. Filtering Data

- Options with a Strike Price greater than 15000 and a Volume greater than 1000 are filtered.

```
filtered_options = option_chain_data_cleaned[(option_chain_data_cleaned['Strike Price'] > 15000) &
(option_chain_data_cleaned['Volume'] > 1000)]
```

8. Data Visualization

- A plot of the bid price for a specified option is generated if the relevant column exists.

```
plt.plot(specific_option_data['Timestamp'], specific_option_data['Bid Price'], marker='o')
```

9. Backtesting Trading Strategy

- The script implements a backtesting strategy for buying Call options based on bid price movement.

```
if (current_row['Option Type'] == 'Call' and

   current_row['Bid Price'] >= previous_row['Bid Price'] * 1.05 and

   (current_row['Timestamp'] - previous_row['Timestamp']).seconds <= 600):
```

10. Performance Metrics Calculation

- It calculates and plots cumulative profit or loss from the identified trades.

```
trade_performance['Cumulative Profit'] = trade_performance['Profit/Loss'].cumsum()
```

11. Cleanup

- The database connection is properly disposed of at the end of the script to free resources.

```
engine.dispose()
```