**Create a Web Application for an ETF Analyzer**

In this Challenge assignment, you’ll build a financial database and web application by using SQL, Python, and the Voilà library to analyze the performance of a hypothetical fintech ETF.

Instructions:

Use this notebook to complete your analysis of a fintech ETF that consists of four stocks: GOST, GS, PYPL, and SQ. Each stock has its own table in the etf.db database, which the Starter\_Code folder also contains.

Analyze the daily returns of the ETF stocks both individually and as a whole. Then deploy the visualizations to a web application by using the Voilà library.

The detailed instructions are divided into the following parts:

* Analyze a single asset in the ETF
* Optimize data access with Advanced SQL queries
* Analyze the ETF portfolio
* Deploy the notebook as a web application

**Analyze a Single Asset in the ETF**

For this part of the assignment, you’ll use SQL queries with Python, Pandas, and hvPlot to analyze the performance of a single asset from the ETF.

Complete the following steps:

1. Write a SQL SELECT statement by using an f-string that reads all the PYPL data from the database. Using the SQL SELECT statement, execute a query that reads the PYPL data from the database into a Pandas DataFrame.
2. Use the head and tail functions to review the first five and the last five rows of the DataFrame. Make a note of the beginning and end dates that are available from this dataset. You’ll use this information to complete your analysis.
3. Using hvPlot, create an interactive visualization for the PYPL daily returns. Reflect the “time” column of the DataFrame on the x-axis. Make sure that you professionally style and format your visualization to enhance its readability.
4. Using hvPlot, create an interactive visualization for the PYPL cumulative returns. Reflect the “time” column of the DataFrame on the x-axis. Make sure that you professionally style and format your visualization to enhance its readability.

**Optimize Data Access with Advanced SQL Queries**

For this part of the assignment, you’ll continue to analyze a single asset (PYPL) from the ETF. You’ll use advanced SQL queries to optimize the efficiency of accessing data from the database.

Complete the following steps:

1. Access the closing prices for PYPL that are greater than 200 by completing the following steps:
   * Write a SQL SELECT statement to select the dates where the PYPL closing price was higher than 200.0.
   * Using the SQL statement, read the data from the database into a Pandas DataFrame, and then review the resulting DataFrame.
   * Select the “time” and “close” columns for those dates where the closing price was higher than 200.0.
2. Find the top 10 daily returns for PYPL by completing the following steps:
   * Write a SQL statement to find the top 10 PYPL daily returns. Make sure to do the following:
     + Use SELECT to select only the “time” and “daily\_returns” columns.
     + Use ORDER to sort the results in descending order by the “daily\_returns” column.
     + Use LIMIT to limit the results to the top 10 daily return values.
   * Using the SQL statement, read the data from the database into a Pandas DataFrame, and then review the resulting DataFrame.

**Analyze the ETF Portfolio**

For this part of the assignment, you’ll build the entire ETF portfolio and then evaluate its performance. To do so, you’ll build the ETF portfolio by using SQL joins to combine all the data for each asset.

Complete the following steps:

1. Write a SQL query to join each table in the portfolio into a single DataFrame. To do so, complete the following steps:
   * Use a SQL inner join to join each table on the “time” column. Access the “time” column in the GDOT table via the GDOT.time syntax. Access the “time” columns from the other tables via similar syntax.
   * Using the SQL query, read the data from the database into a Pandas DataFrame. Review the resulting DataFrame.
2. Create a DataFrame that averages the “daily\_returns” columns for all four assets. Review the resulting DataFrame.

**Hint** Assuming that this ETF contains equally weighted returns, you can average the returns for each asset to get the average returns of the portfolio. You can then use the average returns of the portfolio to calculate the annualized returns and the cumulative returns. For the calculation to get the average daily returns for the portfolio, use the following code:

etf\_portfolio\_returns **=** etf\_portfolio['daily\_returns']**.**mean(axis**=**1)

You can use the average daily returns of the portfolio the same way that you used the daily returns of a single asset.

1. Use the average daily returns in the etf\_portfolio\_returns DataFrame to calculate the annualized returns for the portfolio. Display the annualized return value of the ETF portfolio.

**Hint** To calculate the annualized returns, multiply the mean of the etf\_portfolio\_returns values by 252.

To convert the decimal values to percentages, multiply the results by 100.

1. Use the average daily returns in the etf\_portfolio\_returns DataFrame to calculate the cumulative returns of the ETF portfolio.
2. Using hvPlot, create an interactive line plot that visualizes the cumulative return values of the ETF portfolio. Reflect the “time” column of the DataFrame on the x-axis. Make sure that you professionally style and format your visualization to enhance its readability.

**Deploy the Notebook as a Web Application**

For this part of the assignment, complete the following steps:

1. Use the Voilà library to deploy your notebook as a web application. You can deploy the web application locally on your computer.
2. Take a screen recording or screenshots to show how the web application appears when using Voilà. Include the recording or screenshots in the README.md file for your GitHub repository.

**Review the following code which imports the required libraries, initiates your SQLite database, popluates the database with records from the etf.db seed file that was included in your Starter\_Code folder, creates the database engine, and confirms that data tables that it now contains.**

['GDOT', 'GS', 'PYPL', 'SQ']

**Analyze a single asset in the FinTech ETF**

For this part of the assignment, you’ll use SQL queries with Python, Pandas, and hvPlot to analyze the performance of a single asset from the ETF.

Complete the following steps:

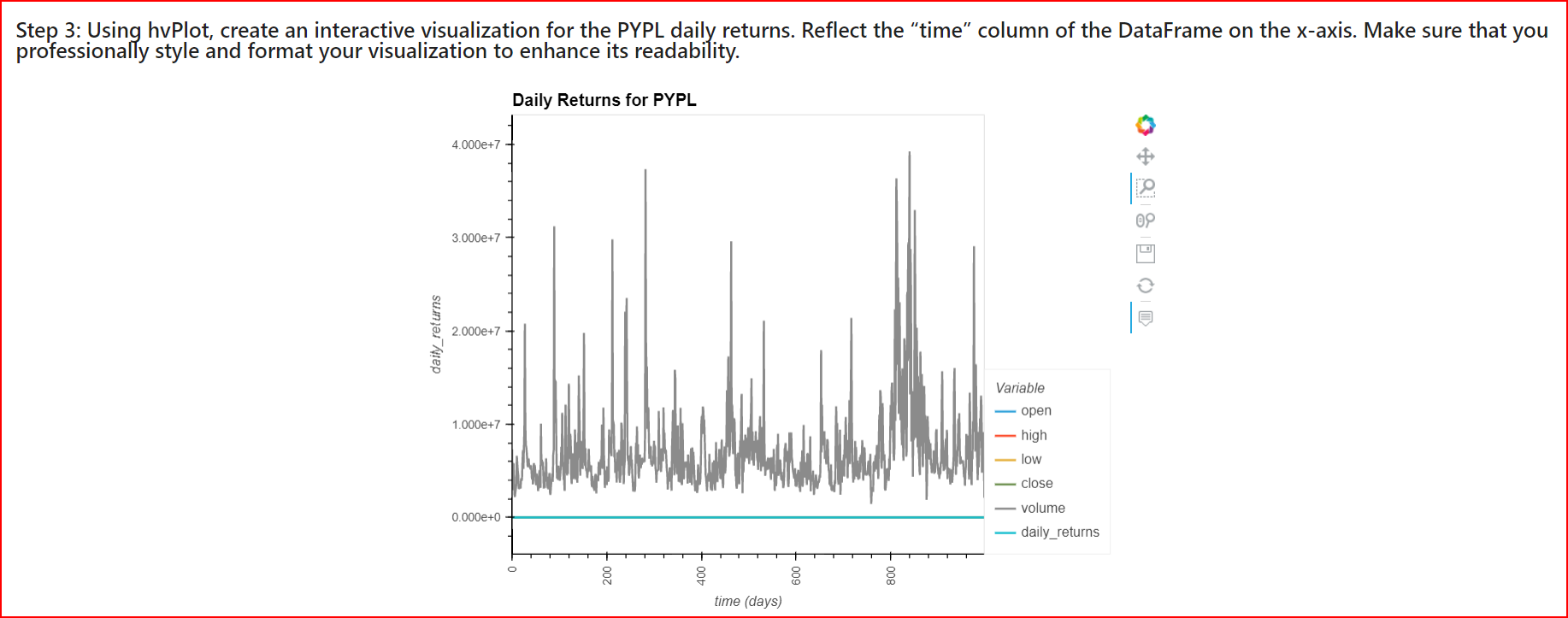
1. Write a SQL SELECT statement by using an f-string that reads all the PYPL data from the database. Using the SQL SELECT statement, execute a query that reads the PYPL data from the database into a Pandas DataFrame.
2. Use the head and tail functions to review the first five and the last five rows of the DataFrame. Make a note of the beginning and end dates that are available from this dataset. You’ll use this information to complete your analysis.
3. Using hvPlot, create an interactive visualization for the PYPL daily returns. Reflect the “time” column of the DataFrame on the x-axis. Make sure that you professionally style and format your visualization to enhance its readability.
4. Using hvPlot, create an interactive visualization for the PYPL cumulative returns. Reflect the “time” column of the DataFrame on the x-axis. Make sure that you professionally style and format your visualization to enhance its readability.

**Step 1: Write a SQL SELECT statement by using an f-string that reads all the PYPL data from the database. Using the SQL SELECT statement, execute a query that reads the PYPL data from the database into a Pandas DataFrame.**

**Step 2: Use the head and tail functions to review the first five and the last five rows of the DataFrame. Make a note of the beginning and end dates that are available from this dataset. You’ll use this information to complete your analysis.**

|  | **time** | **open** | **high** | **low** | **close** | **volume** | **daily\_returns** |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **0** | 2016-12-16 00:00:00.000000 | 39.90 | 39.90 | 39.12 | 39.32 | 7298861 | -0.005564 |
| **1** | 2016-12-19 00:00:00.000000 | 39.40 | 39.80 | 39.11 | 39.45 | 3436478 | 0.003306 |
| **2** | 2016-12-20 00:00:00.000000 | 39.61 | 39.74 | 39.26 | 39.74 | 2940991 | 0.007351 |
| **3** | 2016-12-21 00:00:00.000000 | 39.84 | 40.74 | 39.82 | 40.09 | 5826704 | 0.008807 |
| **4** | 2016-12-22 00:00:00.000000 | 40.04 | 40.09 | 39.54 | 39.68 | 4338385 | -0.010227 |
|  | **time** | **open** | **high** | **low** | **close** | **volume** | **daily\_returns** |
| **994** | 2020-11-30 00:00:00.000000 | 212.51 | 215.83 | 207.0900 | 214.200 | 8992681 | 0.013629 |
| **995** | 2020-12-01 00:00:00.000000 | 217.15 | 220.57 | 214.3401 | 216.520 | 9148174 | 0.010831 |
| **996** | 2020-12-02 00:00:00.000000 | 215.60 | 215.75 | 210.5000 | 212.660 | 6414746 | -0.017827 |
| **997** | 2020-12-03 00:00:00.000000 | 213.33 | 216.93 | 213.1100 | 214.680 | 6463339 | 0.009499 |
| **998** | 2020-12-04 00:00:00.000000 | 214.88 | 217.28 | 213.0100 | 217.235 | 2118319 | 0.011901 |

0.00195722193656932



### Step 4: Using hvPlot, create an interactive visualization for the PYPL cumulative returns. Reflect the “time” column of the DataFrame on the x-axis. Make sure that you professionally style and format your visualization to enhance its readability.

0 0.994436

1 0.997724

2 1.005058

3 1.013910

4 1.003541

...

994 5.417299

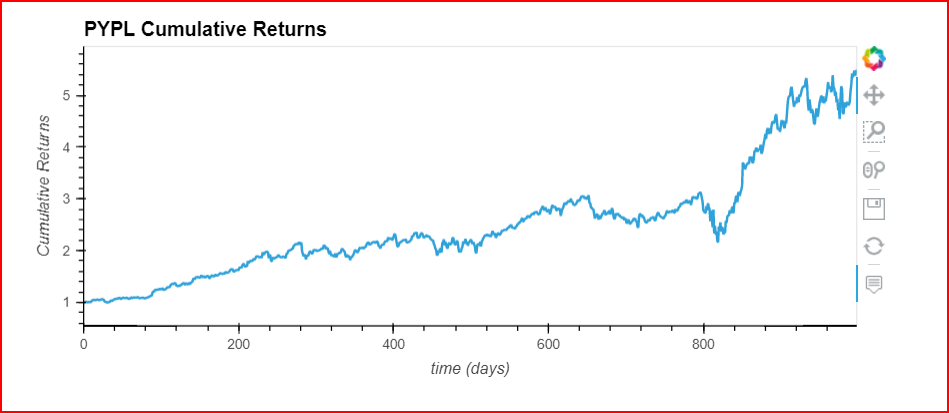
995 5.475974

996 5.378351

997 5.429439

998 5.494057

Name: daily\_returns, Length: 999, dtype: float64



**Optimize the SQL Queries**

For this part of the assignment, you’ll continue to analyze a single asset (PYPL) from the ETF. You’ll use advanced SQL queries to optimize the efficiency of accessing data from the database.

Complete the following steps:

1. Access the closing prices for PYPL that are greater than 200 by completing the following steps:
2. Access the closing prices for PYPL that are greater than 200 by completing the following steps:
   * Write a SQL SELECT statement to select the dates where the PYPL closing price was higher than 200.0.
   * Select the “time” and “close” columns for those dates where the closing price was higher than 200.0.
   * Using the SQL statement, read the data from the database into a Pandas DataFrame, and then review the resulting DataFrame.
3. Find the top 10 daily returns for PYPL by completing the following steps:
   * Write a SQL statement to find the top 10 PYPL daily returns. Make sure to do the following:
     + Use SELECT to select only the “time” and “daily\_returns” columns.
     + Use ORDER to sort the results in descending order by the “daily\_returns” column.
     + Use LIMIT to limit the results to the top 10 daily return values.
   * Using the SQL statement, read the data from the database into a Pandas DataFrame, and then review the resulting DataFrame.

**Step 1: Access the closing prices for PYPL that are greater than 200 by completing the following steps:**

- Write a SQL `SELECT` statement to select the dates where the PYPL closing price was higher than 200.0.

- Select the “time” and “close” columns for those dates where the closing price was higher than 200.0.

- Using the SQL statement, read the data from the database into a Pandas DataFrame, and then review the resulting DataFrame.

|  | **time** | **close** |
| --- | --- | --- |
| **0** | 2020-08-05 00:00:00.000000 | 202.92 |
| **1** | 2020-08-06 00:00:00.000000 | 204.09 |
| **2** | 2020-08-25 00:00:00.000000 | 201.71 |
| **3** | 2020-08-26 00:00:00.000000 | 203.53 |
| **4** | 2020-08-27 00:00:00.000000 | 204.34 |
| **5** | 2020-08-28 00:00:00.000000 | 204.48 |
| **6** | 2020-08-31 00:00:00.000000 | 203.95 |
| **7** | 2020-09-01 00:00:00.000000 | 208.92 |
| **8** | 2020-09-02 00:00:00.000000 | 210.82 |
| **9** | 2020-09-03 00:00:00.000000 | 205.07 |

**Step 2: Find the top 10 daily returns for PYPL by completing the following steps:**

- Write a SQL statement to find the top 10 PYPL daily returns. Make sure to do the following:

\* Use `SELECT` to select only the “time” and “daily\_returns” columns.

\* Use `ORDER` to sort the results in descending order by the “daily\_returns” column.

\* Use `LIMIT` to limit the results to the top 10 daily return values.

- Using the SQL statement, read the data from the database into a Pandas DataFrame, and then review the resulting DataFrame.

|  | **time** | **daily\_returns** |
| --- | --- | --- |
| **0** | 2020-03-24 00:00:00.000000 | 0.140981 |
| **1** | 2020-05-07 00:00:00.000000 | 0.140318 |
| **2** | 2020-03-13 00:00:00.000000 | 0.138700 |
| **3** | 2020-04-06 00:00:00.000000 | 0.100877 |
| **4** | 2018-10-19 00:00:00.000000 | 0.093371 |
| **5** | 2019-10-24 00:00:00.000000 | 0.085912 |
| **6** | 2020-11-04 00:00:00.000000 | 0.080986 |
| **7** | 2020-03-10 00:00:00.000000 | 0.080863 |
| **8** | 2020-04-22 00:00:00.000000 | 0.075321 |
| **9** | 2018-12-26 00:00:00.000000 | 0.074656 |

**Analyze the Fintech ETF Portfolio**

For this part of the assignment, you’ll build the entire ETF portfolio and then evaluate its performance. To do so, you’ll build the ETF portfolio by using SQL joins to combine all the data for each asset.

Complete the following steps:

1. Write a SQL query to join each table in the portfolio into a single DataFrame. To do so, complete the following steps:
   * Use a SQL inner join to join each table on the “time” column. Access the “time” column in the GDOT table via the GDOT.time syntax. Access the “time” columns from the other tables via similar syntax.
   * Using the SQL query, read the data from the database into a Pandas DataFrame. Review the resulting DataFrame.
2. Create a DataFrame that averages the “daily\_returns” columns for all four assets. Review the resulting DataFrame.

**Hint** Assuming that this ETF contains equally weighted returns, you can average the returns for each asset to get the average returns of the portfolio. You can then use the average returns of the portfolio to calculate the annualized returns and the cumulative returns. For the calculation to get the average daily returns for the portfolio, use the following code:

etf\_portfolio\_returns **=** etf\_portfolio['daily\_returns']**.**mean(axis**=**1)

You can use the average daily returns of the portfolio the same way that you used the daily returns of a single asset.

1. Use the average daily returns in the etf\_portfolio\_returns DataFrame to calculate the annualized returns for the portfolio. Display the annualized return value of the ETF portfolio.

**Hint** To calculate the annualized returns, multiply the mean of the etf\_portfolio\_returns values by 252.

To convert the decimal values to percentages, multiply the results by 100.

1. Use the average daily returns in the etf\_portfolio\_returns DataFrame to calculate the cumulative returns of the ETF portfolio.
2. Using hvPlot, create an interactive line plot that visualizes the cumulative return values of the ETF portfolio. Reflect the “time” column of the DataFrame on the x-axis. Make sure that you professionally style and format your visualization to enhance its readability.

**Step 1: Write a SQL query to join each table in the portfolio into a single DataFrame. To do so, complete the following steps:**

- Use a SQL inner join to join each table on the “time” column. Access the “time” column in the `GDOT` table via the `GDOT.time` syntax. Access the “time” columns from the other tables via similar syntax.

- Using the SQL query, read the data from the database into a Pandas DataFrame. Review the resulting DataFrame.

|  | **time** | **"GDOT\_RETURNS"** | **"PYPL\_RETURNS"** | **"GS\_RETURNS"** | **"SQ\_RETURNS"** |
| --- | --- | --- | --- | --- | --- |
| **0** | 2016-12-16 00:00:00.000000 | -0.023218 | -0.005564 | -0.016708 | 0.017339 |
| **1** | 2016-12-19 00:00:00.000000 | -0.007923 | 0.003306 | 0.000795 | -0.001043 |
| **2** | 2016-12-20 00:00:00.000000 | 0.001261 | 0.007351 | 0.016602 | 0.009053 |
| **3** | 2016-12-21 00:00:00.000000 | 0.001679 | 0.008807 | -0.006911 | -0.007591 |
| **4** | 2016-12-22 00:00:00.000000 | 0.006077 | -0.010227 | -0.005178 | -0.023644 |
|  | **time** | **"GDOT\_RETURNS"** | **"PYPL\_RETURNS"** | **"GS\_RETURNS"** | **"SQ\_RETURNS"** |
| **994** | 2020-11-30 00:00:00.000000 | -0.043750 | 0.013629 | -0.021266 | -0.007153 |
| **995** | 2020-12-01 00:00:00.000000 | 0.004482 | 0.010831 | 0.006549 | -0.037823 |
| **996** | 2020-12-02 00:00:00.000000 | -0.027328 | -0.017827 | 0.024387 | -0.004384 |
| **997** | 2020-12-03 00:00:00.000000 | 0.027523 | 0.009499 | -0.008959 | 0.016921 |
| **998** | 2020-12-04 00:00:00.000000 | 0.001860 | 0.011901 | 0.012520 | 0.010151 |

**Step 2: Create a DataFrame that averages the “daily\_returns” columns for all four assets. Review the resulting DataFrame.**

"GDOT\_RETURNS" 0.001504

"PYPL\_RETURNS" 0.001957

"GS\_RETURNS" 0.000196

"SQ\_RETURNS" 0.003300

dtype: float64

**Step 3: Use the average daily returns in the etf\_portfolio\_returns DataFrame to calculate the annualized returns for the portfolio. Display the annualized return value of the ETF portfolio.**

"GDOT\_RETURNS" 0.378933

"PYPL\_RETURNS" 0.493220

"GS\_RETURNS" 0.049281

"SQ\_RETURNS" 0.831657

dtype: float64

**Step 4: Use the average daily returns in the etf\_portfolio\_returns DataFrame to calculate the cumulative returns of the ETF portfolio.**

"GDOT\_RETURNS" 1.001504

"PYPL\_RETURNS" 1.003464

"GS\_RETURNS" 1.003660

"SQ\_RETURNS" 1.006972

dtype: float64

**Step 5: Using hvPlot, create an interactive line plot that visualizes the cumulative return values of the ETF portfolio. Reflect the “time” column of the DataFrame on the x-axis. Make sure that you professionally style and format your visualization to enhance its readability.**

