Stock Market Portfolio Analysis and Optimization

Titan Financial Group

Team 4

Temi Oyefeso, Sheetal Padmanabhan, Ishani Parkar, David Pancardo, and Jung Won

Department of Information Systems & Decision Sciences, California State University Fullerton

ISDS 570: Business Data Transformation

Dr. Pawel Kalczynski

December 15, 2024

Table of Contents

Executive Summary 3
Project Overview 4
Database Architecture 6
ETL Implementation 9
Portfolio Construction 12
Performance Evaluation 14
Appendix A – SQL Code 17
Appendix B – R Code 30

Executive Summary

Titan Financial Group (TFG) is committed to building financial portfolios through sophisticated data transformation and processing techniques to achieve optimized analyses. Our client, Dr. K, Professor of Information Systems and Decision Sciences at California State University – Fullerton, contacted us with an urgent request. In response to Dr. K's request, TFG developed a portfolio to evaluate its projected performance against the S&P 500 Total Return Index (S&P 500 TR).

TFG analyzed a mean-variance (MV) portfolio comprising 15 stocks to facilitate this. This involved an extensive ETL process to collect and preprocess relevant data. The team utilized public daily data from three major stock exchanges—NASDAQ, NYSE, and AMEX—spanning 5 years (2016-2020) along with the first quarter of 2021. The primary objective of this initiative was to evaluate the portfolio's performance with the S&P 500 Index by training the dataset and conducting portfolio back testing for the first quarter of 2021 while also identifying opportunities for optimizing the portfolio crafted by TFG.

Our analysis indicated that the portfolio outperformed the S&P 500 TR by 11.81% for Q1 2021. We identified several critical factors contributing to this success, including sector exposure, stock selection, and effective risk management. Among the 15 stocks in the portfolio, the standout performers were Burlington Stores Inc. (BURL), Paylocity Holding Corporation (PCTY), and LHC Group (LCHG).

Project Overview

- *Purpose and Objectives*

Titan Financial Group maintains a stringent standard for creating a high-quality portfolio with appropriate variance risk ratios. The primary aim of this study is to construct and evaluate a 15-stock portfolio utilizing a Mean-Variance optimization approach. By analyzing historical data from 2016 to 2020, we seek to enhance the portfolio's risk-return profile and measure its performance against a relevant benchmark, the S&P 500 Total Return Index (S&P 500 TR), during the first quarter of 2021.

- *Data Range Selection*

The initial data collection consisted of daily figures from three major stock exchanges—NASDAQ, NYSE, and AMEX—from 2015 to 2020. We aimed to optimize the minimum acceptable return (MAR) for the SP500TR index, explicitly focusing on the timeframe from 2016 to 2020 while ensuring our portfolio complied with this constraint. To calculate the MAR and determine portfolio weights, we refined the data to include only trading days from January 1, 2016, to December 31, 2020. For portfolio back testing purposes, we obtained daily trading data for our selected stocks and the benchmark, utilizing a data range from January 1 to March 26, 2021. The data sources for this analysis are Yahoo! Finance (finance.yahoo.com) and the internal end-of-day (EOD) stock quote data from our client, Dr. K.

- *Portfolio Composition*

TFG's portfolio comprises 15 carefully selected stocks. A team of five evaluated three stocks each, ensuring no inaccuracies in the stock price listings and that no more than 1% of the complete trading data was missing for our chosen data range. The portfolio features a diverse array of sectors. Below is a table listing the stock ticker, company name, and corresponding industry.

Stock Ticker	Company Name	Sector
ENS	EnerSys	Industrials
PCTY	Paylocity Holding Corporation	Technology
FCNCA	First Citizens BancShares, Inc.	Financial Services
CBFV	CB Financial Services, Inc.	Financial Services
MPW	Medical Properties Trust, Inc.	Real Estate
ORAN	Orange S.A.	Technology

FWONK	Formula One Group	Communication Services
LHCG	LHC Group, Inc.	Healthcare
POR	Portland General Electric Company	Utilities
AGM	Federal Agricultural Mortgage Corporation	Financial Services
MIELY	Mitsubishi Electric Corporation	Industrials
BURL	Burlington Stores, Inc.	Consumer Cyclical
SMTOY	Sumitomo Electric Industries, Ltd.	Consumer Cyclical
ESLOY	Essilor Luxottica Société anonyme	Healthcare Cyclical
MLVF	Malvern Bancorp, Inc.	Banking Services

- *Data Engineering Framework*

  This analysis involves:

  - **Data Collection and Preparation:** Collect daily stock price data for the 15 chosen stocks and the S&P 500 TR index. Utilizing an ETL process to extract daily returns for each selected stock and developing a custom calendar that accounts for all stock trading holidays to accurately determine the number of trading days in the year.
  - **Portfolio Optimization:** Employing mean-variance optimization to determine the optimal allocation weights for each stock, subject to a minimum return constraint.
  - **Back testing:** Evaluating the performance of the optimized portfolio on out-of-sample data (2021).
  - **Performance Analysis:** Assessing the portfolio's performance metrics, including annualized return, annualized variance, and annualized Sharpe ratio.

Database Architecture

Schema Design

The schema design of our database stockmarket_GP comprises five tables and views. Our database was designed with the principles of a relational database that ensures data normalization, data integrity controls, and access management. Each table in our database was created with a unique purpose, focusing on reducing redundancy, improving efficiency, and maintaining consistency. Constraints like primary key and not null were often applied to columns to ensure logical consistency and data integrity.

- ***Tables***
  - **stock_mkt**: The table stores the list of stock markets (e.g., NASDAQ, NYSE, AMEX) to provide a unique central repository for the stock market names.
  - **company_list**: The table contains information about companies, including but not limited to their company name, market cap, and IPO year. It establishes a relationship between companies and their respective stock markets, ensuring data integrity and efficiency.
  - **eod_quotes**: This table records end-of-the-day trading data for stocks, such as price, and provides an essential starting foundation for financial data analysis.
  - **custom_calendar**: This table maintains a custom calendar with NYSE trading days, holidays, and metadata such as end-of-month and previous trading days. custom_calendar ensures alignment of date data from the eod_quotes table with valid trading days, enabling accurate analysis that involves time and reducing errors due to non-trading days.
  - **exclusions_2016_2021**: This table tracks records of trading days from 2016-2021 that were excluded from analysis due to data quality issues. This table helps maintain the reliability of analysis by recording and managing data with quality issues.
- ***Views***
  - **v_company_list**: This view simplifies the company_list table by removing and excluding any irrelevant attributes. This view reduces complexity in querying while maintaining access to essential company information.
  - **v_eod_quotes_2016_2021**: This view filters the eod_quotes table to focus on relevant columns from records from 2016 to 2021. This allows for data access optimization as irrelevant information is excluded from queries.
  - **Materialized Views**:
    - **mv_eod_2016_2021**: This materialized view filters quotes data using unique tickers from the exclusions_2016_2021 table.

- **mv_ret_2016_2021**: This materialized view stores daily returns for stocks to improve the performance of calculations.

The tables are linked to each other to maintain referential integrity. Below is a list of those relationships:

- **stock_mkt** and **company_list**: The stock_mkt table is related through a one-to-many relationship with company_list. Each name in stock_mkt can be associated with multiple company names in company_list.
- **company_list** and **eod_quotes**: The company_list table is linked to the eod_quotes table through the symbol column in a one-to-many relationship. This relationship ensures that all trading data in the eod_quotes table corresponds with companies in the company_list table.
- **eod_quotes** and **custom_calendar**: The custom_calendar table aligns the eod_quotes data with valid trading days through their respective date columns. This relationship ensures that only relevant dates are used in any analysis using the date field.

Data Integrity Controls

Our database stockmarket_GP employs data cleaning, constraints, and exclusion tables to ensure data accuracy, consistency, and completeness. Data integrity was key to ensuring our final analysis was error-free.

- ***Constraints*:**
  - **Primary Keys**: The primary key constraint ensures each table has unique and identifiable rows like unique stock market names.
  - **NOT NULL Constraints**: The NOT NULL constraint ensures that there isn't any missing data by enforcing non-null requirements for essential columns like stock market names or company symbols.
- ***Data Cleaning & Transformation:***
  - **Trimming**: Trimming of various columns ensures uniformity by removing unnecessary whitespace from strings such as stock market names and company names.
  - **Type Conversion**: Transformation of data into appropriate data type.
- ***Exclusions Table**:*
  - As the saying goes, garbage in, garbage out. This table excludes garbage from our analysis. Excluding records with issues enhances the accuracy and reliability of data used in the analysis.

Access Management

The stockmarket_GP database was built with a robust access management strategy to ensure secure and controlled access. We created a read-only user, stockmarketreadergp, to limit permissions and accessibility and prevent security breaches and unintentional user errors while allowing access for analysis and reporting. We aim to restrict access to what is necessary for the user's role, minimizing security risks and user oversight.

ETL Implementation

Data Acquisition Process

The stock market data was acquired from two different sources: Yahoo Finance and Dr.K:

- Yahoo Finance: Yahoo Finance is a widely used platform for financial data. Yahoo Finance provides historical data for indices like SP500.
- Dr. K: The professor's dataset included financial data, date, company name, and symbols.

The data from Yahoo Finance was accessed via their website. The platform is free to use and easily accessed without requiring a login. The data was filtered to show daily historical prices within the specified date range, as shown below. The data was then copied from the table on the website and pasted into an Excel file. After adjustments, the file was imported into a PostgreSQL database using the Import feature.

S&P 500 (TR) (^SP500TR) ☆ Follow

13,276.56 +1.32 +(0.01%)

At close: December 13 at 4:48:12 PM EST

Jan 01, 2016 - Dec 15, 2024 | Historical Prices | Daily

Currency in USD | Download

Date	Open	High	Low	Close ⓘ	Adj Close ⓘ	Volume
Dec 13, 2024	13,314.03	13,336.85	13,242.94	13,276.56	13,276.56	-
Dec 12, 2024	13,325.78	13,337.61	13,275.24	13,275.24	13,275.24	-
Dec 11, 2024	13,294.78	13,365.82	13,294.78	13,347.51	13,347.51	-
Dec 10, 2024	13,288.96	13,305.69	13,228.19	13,239.21	13,239.21	-
Dec 9, 2024	13,344.45	13,356.37	13,269.32	13,278.29	13,278.29	-
Dec 6, 2024	13,339.00	13,379.76	13,336.22	13,358.50	13,358.50	-
Dec 5, 2024	13,354.28	13,366.27	13,319.85	13,323.74	13,323.74	-

The data from Dr. K had a limited date range, covering dates up to 2021-03-26. As a result, SP500 data was gathered to match this date range. Dr. K's data was split into three CSV files, each containing financial data for stocks within the specified date range. The files were organized by exchange: AMEX, NASDAQ, and NYSE. These CSV files were then imported into a PostgreSQL database using the Import feature, like the process for Yahoo Finance data. The SP500 data was incorporated into Dr. K's dataset by appending it. Additionally, a missing row for SP500TR on 2015-12-31 was manually added to ensure continuity.

The data was stored in a PostgreSQL database and accessed via R using the RPostgres package. A connection to the PostgreSQL database was made using the dbConnect function from the DBI package. As mentioned in the Access Management section, the stockmarketreadergp user, with limited accessibility, was used to ensure security and prevent user oversight. SQL queries were

constructed to extract relevant symbols and trading data, further filtered to retrieve a date range from 2015-12-31 to 2021-03-26, and only include relevant tickers.

	ticker [PK] character varying (16)	date [PK] date	adj_open real	adj_high real	adj_low real	adj_close real	adj_volume numeric
1	SMTOY	2020-01-29	14.41	14.41	14.41	14.41	0
2	SMTOY	2020-01-27	14.41	14.41	14.41	14.41	0
3	SMTOY	2020-02-06	14.41	14.41	14.41	14.41	0
4	SMTOY	2020-01-30	14.41	14.41	14.41	14.41	0
5	SMTOY	2020-02-03	14.41	14.41	14.41	14.41	0
6	SMTOY	2020-01-28	14.41	14.41	14.41	14.41	0
7	SMTOY	2020-06-11	12.38	12.38	12.38	12.38	0
8	SMTOY	2020-06-23	11.84	11.84	11.84	11.84	0
9	SMTOY	2020-02-14	13.44	13.44	13.44	13.44	0
10	SMTOY	2020-01-31	14.41	14.41	14.41	14.41	0

Quality Control Measures

We took various measures to ensure quality control. Our first measure involved refining our SQL query. Date ranges were defined to limit the scope of trading periods, thereby avoiding the inclusion of any unwanted data in our dataset. The query was also constrained to retrieving only relevant tickers. Per the instructions, we focused on the tickers assigned in HW2. By defining the tickers, we reduced the risk of adding irrelevant data to our dataset.

Next, the data was explored to ensure its structure, size, and content met expectations. Functions like head(), tail(), and nrow() were used to review the dataset. This inspection was further supported with additional steps like explicitly declaring data types during transformations and verifying dataset dimensions and structure after every major operation.

Furthermore, imputation for missing data step was implemented using the zoo package. The “last observation carried forward” method was applied to fill in missing values. This constraint was implemented so that the policy would only be applied to rows with gaps of no more than three consecutive days.

Lastly, the percentage of completeness was checked for each symbol using the table() function. Only symbols with at least 99% completeness were selected for processing. This step filtered out stocks that did not meet the required trading frequency and identified symbols appearing in the dataset at least 99% of the time. This ensured data completeness and removed noisy symbols that failed to meet our frequency requirement.

Trading Calendar Integration

A custom trading calendar was created to align stock market data with actual trading days. The calendar was based on the NYSE stock exchange's calendar as a reference. Using Excel, It was then filtered to include only trading days, limiting the dataset to valid trading periods and ensuring the exclusion of any irrelevant dates that could skew the analysis.

Portfolio Construction

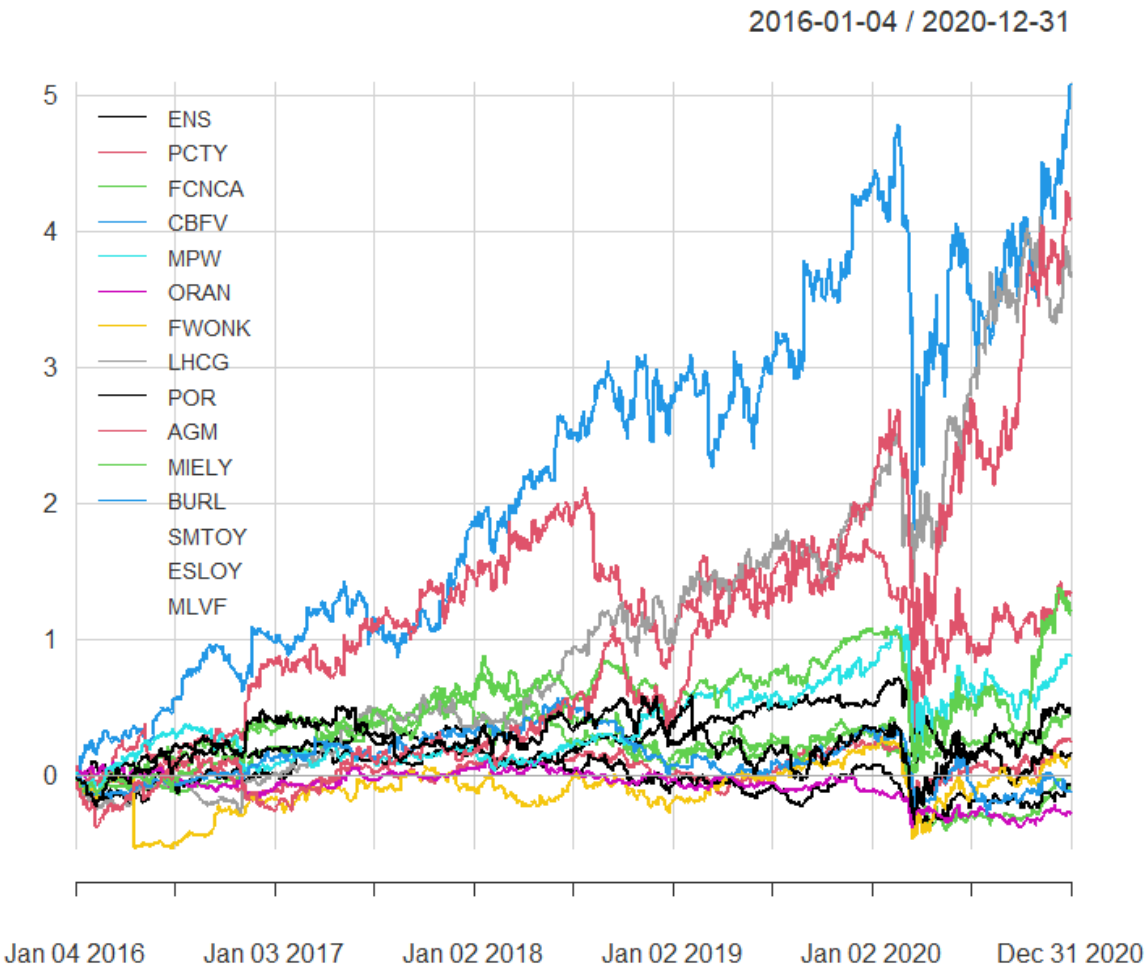
Stock Performance Metrics

Using the cleaned data and removing all exclusions, we analyzed stock performance using the historical end-of-day stock data from 2016 to 2021. Key metrics included adjusted open, high, low, and close prices and volume. This comprehensive dataset allowed for a detailed examination of each stock's historical behavior and volatility.

Return Calculations

Daily and cumulative returns were computed using the. We utilized SQL queries to calculate percentage changes in adjusted closing prices, enabling the assessment of both short-term price movements and long-term performance trends for each stock.

The cumulative returns chart for the stocks selected for 2016-2020 is shown below.



Market Correlation Study

We conducted a market correlation study to determine how closely individual stock returns align with the broader market, represented by the SP500TR index. Data for the SP500TR index from 2016 to 2021 was used.

Weight Distribution

We determined the weight of each stock based on market capitalization data stored. This approach provides a starting point for portfolio allocation, reflecting each company's relative size in the market. This analysis was facilitated by combining company details with numeric market cap values. The individual weights for the tickers from the optimized portfolio for Range 1 (2016 – 2020) are shown in the table below (Rounded to 4 decimal places). Also, it was verified that the sum of these weights equated to 1.

Tickers	**Optimized Weights**
ENS	-0.1494
PCTY	0.0407
FCNCA	0.0261
CBFV	0.1606
MPW	-0.0515
ORAN	0.0848
FWONK	-0.0328
LHCG	0.1892
POR	0.1328
AGM	0.0540
MIELY	0.1606
BURL	0.1113
SMTOY	0.1180
ESLOY	0.1268
MLVF	0.0288

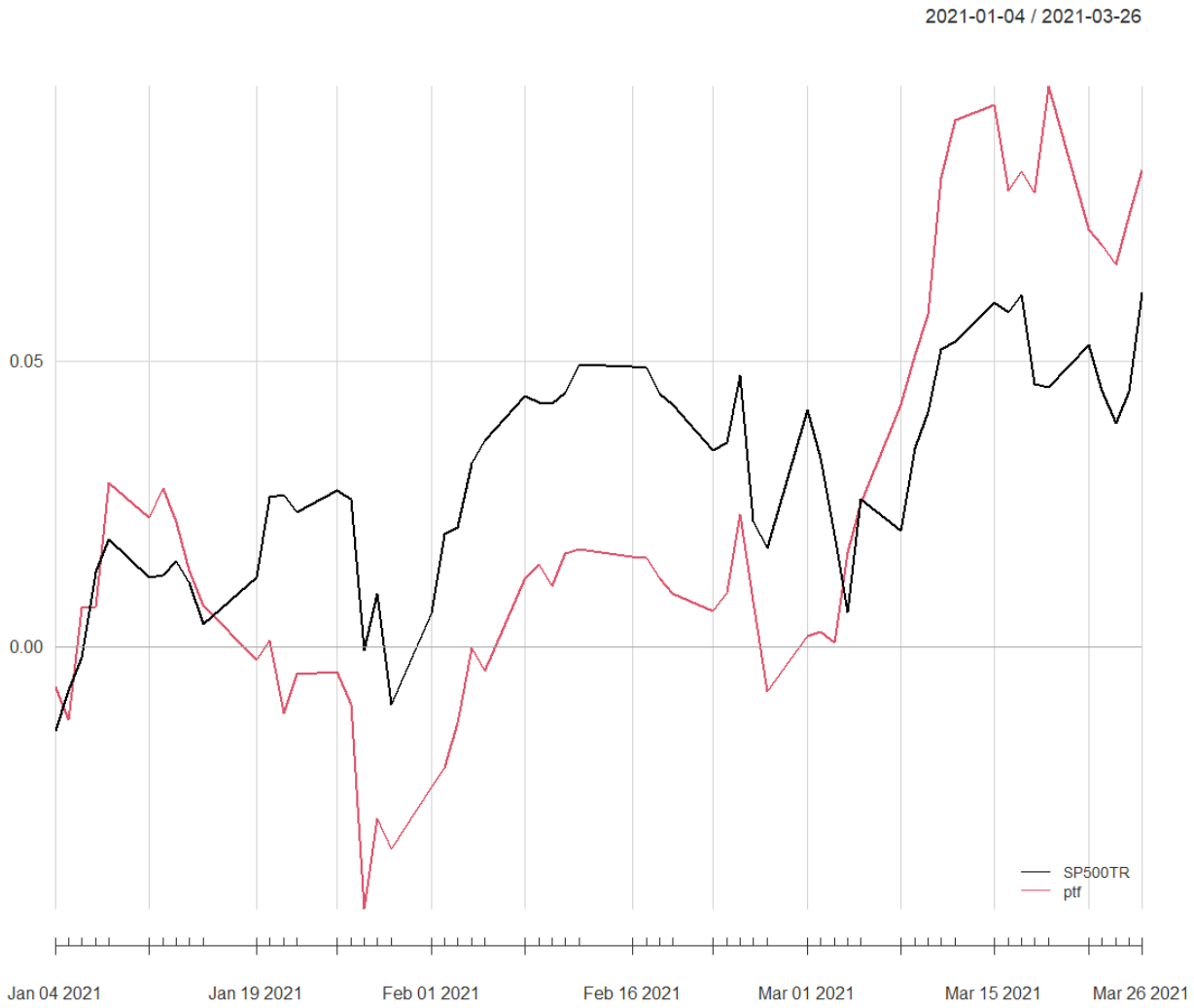
Performance Evaluation

Out-of-Sample Performance

We evaluated out-of-sample performance by splitting our data into training (2016-2020) and testing (Jan 2021 – Mar 2021) periods. This approach helps assess the robustness of our investment strategy across different market conditions.

SP500TR Benchmark Comparison

Cumulative returns for the portfolio and the SP500TR benchmark were calculated and compared over the entire period (Jan 2021 – Mar 2021). This comparison illustrates how our active management strategy performed relative to a passive index investment. Below is the cumulative returns chart for the optimized portfolio and the SP500TR benchmark over Jan 2021 – Mar 2021.



Risk-Adjusted Returns

We calculated risk-adjusted performance metrics for both our portfolio and the SP500TR benchmark. This provides insight into how well the portfolio performed on a risk-adjusted basis during the testing period. Below are the results to help you understand portfolio performance.

Parameters / Symbol	**SP500TR**	**Portfolio**
Annualized Return	0.2987	0.4168
Annualized Standard Deviation	0.1623	0.1757
Annualized Sharpe (Rf = 0%)	1.8399	2.3721

This table presents annualized performance metrics for the SP500TR (S&P 500 Total Return Index) and our custom portfolio. Here's a detailed explanation of each metric:

- Annualized Return:
  - SP500TR: 29.87% per year
  - Custom portfolio: 41.68% per year
    The custom portfolio significantly outperformed the S&P 500 benchmark, generating an additional 11.81% annualized returns.
- Annualized Standard Deviation:
  - SP500TR: 16.23%
  - Custom portfolio: 17.57%
    This measure represents the volatility of returns. The custom portfolio shows slightly higher volatility (1.34% more) than the benchmark, indicating marginally increased risk.
- Annualized Sharpe Ratio (assuming 0% risk-free rate):
  - SP500TR: 1.8399
  - Custom portfolio: 2.3721
    The Sharpe ratio measures risk-adjusted returns. A higher ratio indicates better risk-adjusted performance. The custom portfolio's Sharpe ratio is 0.5322 higher than the benchmark, suggesting it provided superior returns per unit of risk taken.

Overall, the custom portfolio achieved significantly higher returns with only a slight increase in volatility. This resulted in a notably better risk-adjusted performance than the S&P 500 Total Return index. The higher Sharpe ratio for the custom portfolio indicates that it provided more excess return per unit of risk, making it a more attractive investment from a risk-adjusted perspective.

Appendix A – SQL Code

```
-------------------------------------------------------------
-- First, created a database "stockmarket_GP" under "Databases"
-------------------------------------------------------------

-------------------------------------------------------------
-- Let us create the actual tables to transfer data there
-------------------------------------------------------------

-- DROP TABLE public.stock_mkt;
CREATE TABLE public.stock_mkt
(
    stock_mkt_name character varying(16) COLLATE pg_catalog."default" NOT NULL,
    CONSTRAINT stock_mkt_pkey PRIMARY KEY (stock_mkt_name)
)
WITH (
    OIDS = FALSE
)
TABLESPACE pg_default;

ALTER TABLE public.stock_mkt
    OWNER to postgres;

-- DROP TABLE public.company_list;
CREATE TABLE public.company_list
(
    symbol character varying(16) COLLATE pg_catalog."default" NOT NULL,
    stock_mkt_name character varying(16) COLLATE pg_catalog."default" NOT NULL,
    company_name character varying(255) COLLATE pg_catalog."default",
    market_cap double precision,
       country character varying(255) COLLATE pg_catalog."default",
       ipo_year integer,
       sector character varying(255) COLLATE pg_catalog."default",
    industry character varying(255) COLLATE pg_catalog."default",
    CONSTRAINT company_list_pkey PRIMARY KEY (symbol, stock_mkt_name)
)
WITH (
    OIDS = FALSE
)
TABLESPACE pg_default;

ALTER TABLE public.company_list
    OWNER to postgres;

-- We have PK and other entity integrity constraints, now let us set the referential
integrity (FK) constraints
ALTER TABLE public.company_list
    ADD CONSTRAINT company_list_fkey FOREIGN KEY (stock_mkt_name)
    REFERENCES public.stock_mkt (stock_mkt_name) MATCH SIMPLE
    ON UPDATE NO ACTION
    ON DELETE NO ACTION
    NOT VALID;
```

```
CREATE INDEX fki_company_list_fkey
    ON public.company_list(stock_mkt_name);



-----------------------------------------------
-- Populate the final tables with data
-----------------------------------------------

DELETE FROM stock_mkt;

-- Insert NASDAQ data with proper casting (type conversion)
-- Create a temporary table and import data
-- DROP TABLE public._temp_company_list;
CREATE TABLE public._temp_company_list
(
    symbol character varying(255) COLLATE pg_catalog."default",
    name character varying(255) COLLATE pg_catalog."default",
    last_sale character varying(255) COLLATE pg_catalog."default",
    net_change character varying(255) COLLATE pg_catalog."default",
       pct_change character varying(255) COLLATE pg_catalog."default",
       market_cap character varying(255) COLLATE pg_catalog."default",
       country character varying(255) COLLATE pg_catalog."default",
    ipo_year character varying(255) COLLATE pg_catalog."default",
    volume character varying(255) COLLATE pg_catalog."default",
    sector character varying(255) COLLATE pg_catalog."default",
    industry character varying(255) COLLATE pg_catalog."default"
)
WITH (
    OIDS = FALSE
)
TABLESPACE pg_default;

ALTER TABLE public._temp_company_list
    OWNER to postgres;

-- Once created import data from csv companylist_nasdaq.csv

-- Check if we have data in the correct format
SELECT * FROM _temp_company_list LIMIT 10;
SELECT COUNT(*) from _temp_company_list;

-- Insert Market Name in table
INSERT INTO stock_mkt (stock_mkt_name) VALUES ('NASDAQ');

-- Check!
SELECT * FROM stock_mkt;

-- We will load the company_list with data stored in _temp_company_list
INSERT INTO company_list
SELECT symbol, 'NASDAQ' AS stock_mkt_name, name company_name, market_cap::double
precision ,country, ipo_year::integer, sector,industry
FROM _temp_company_list;

-- Insert NYSE data with proper casting (type conversion)
```

```
-- Create a temporary table and import data
DROP TABLE public._temp_company_list;
CREATE TABLE public._temp_company_list
(
    symbol character varying(255) COLLATE pg_catalog."default",
    name character varying(255) COLLATE pg_catalog."default",
    last_sale character varying(255) COLLATE pg_catalog."default",
    net_change character varying(255) COLLATE pg_catalog."default",
       pct_change character varying(255) COLLATE pg_catalog."default",
       market_cap character varying(255) COLLATE pg_catalog."default",
       country character varying(255) COLLATE pg_catalog."default",
    ipo_year character varying(255) COLLATE pg_catalog."default",
    volume character varying(255) COLLATE pg_catalog."default",
    sector character varying(255) COLLATE pg_catalog."default",
    industry character varying(255) COLLATE pg_catalog."default"
)
WITH (
    OIDS = FALSE
)
TABLESPACE pg_default;

ALTER TABLE public._temp_company_list
    OWNER to postgres;

-- Once created import data from csv companylist_nyse.csv

-- Check if we have data in the correct format
SELECT * FROM _temp_company_list LIMIT 10;
SELECT COUNT(*) from _temp_company_list;

-- Insert Market Name in table
INSERT INTO stock_mkt (stock_mkt_name) VALUES ('NYSE');

-- Check!
SELECT * FROM stock_mkt;

-- We will load the company_list with data stored in _temp_company_list
INSERT INTO company_list
SELECT symbol, 'NYSE' AS stock_mkt_name, name company_name, market_cap::double
precision ,country, ipo_year::integer, sector,industry
FROM _temp_company_list;

-- Insert AMEX data with proper casting (type conversion)
-- Create a temporary table and import data
DROP TABLE public._temp_company_list;
CREATE TABLE public._temp_company_list
(
    symbol character varying(255) COLLATE pg_catalog."default",
    name character varying(255) COLLATE pg_catalog."default",
    last_sale character varying(255) COLLATE pg_catalog."default",
    net_change character varying(255) COLLATE pg_catalog."default",
       pct_change character varying(255) COLLATE pg_catalog."default",
       market_cap character varying(255) COLLATE pg_catalog."default",
       country character varying(255) COLLATE pg_catalog."default",
    ipo_year character varying(255) COLLATE pg_catalog."default",
```

```
    volume character varying(255) COLLATE pg_catalog."default",
    sector character varying(255) COLLATE pg_catalog."default",
    industry character varying(255) COLLATE pg_catalog."default"
)
WITH (
    OIDS = FALSE
)
TABLESPACE pg_default;

ALTER TABLE public._temp_company_list
    OWNER to postgres;

-- Once created import data from csv companylist_amex.csv

-- Check if we have data in the correct format
SELECT * FROM _temp_company_list LIMIT 10;
SELECT COUNT(*) from _temp_company_list;

-- Insert Market Name in table
INSERT INTO stock_mkt (stock_mkt_name) VALUES ('AMEX');

-- Check!
SELECT * FROM stock_mkt;

-- We will load the company_list with data stored in _temp_company_list
INSERT INTO company_list
SELECT symbol, 'AMEX' AS stock_mkt_name, name company_name, market_cap::double
precision ,country, ipo_year::integer, sector,industry
FROM _temp_company_list;

-- Check
SELECT * FROM company_list LIMIT 10;
SELECT COUNT(*) FROM company_list;



-----------------------------------------------------------------
-- Dealing with unwanted values  and leading/trailing blanks ----
-----------------------------------------------------------------

SELECT * FROM company_list order by market_cap LIMIT 100;
UPDATE company_list SET market_cap=NULL WHERE market_cap=0;
SELECT * FROM company_list order by market_cap LIMIT 100;

UPDATE stock_mkt SET stock_mkt_name=TRIM(stock_mkt_name);

UPDATE company_list SET
       stock_mkt_name=TRIM(stock_mkt_name)
       ,company_name=TRIM(company_name)
       ,country=TRIM(country)
       ,sector=TRIM(sector)
       ,industry=TRIM(industry);

SELECT * FROM company_list LIMIT 10;
```

```
-----------------------------------------------
----------------- Create a View --------------
-----------------------------------------------

-- Let us create a view v_company_list using the select statement with numeric market
cap
CREATE OR REPLACE VIEW public.v_company_list AS
 SELECT company_list.symbol,
    company_list.stock_mkt_name,
    company_list.company_name,
    company_list.market_cap,
    company_list.country,
    company_list.sector,
    company_list.industry
   FROM company_list;

ALTER TABLE public.v_company_list
    OWNER TO postgres;

-- Check!
SELECT * FROM v_company_list;



-----------------------------------------------------------
----------------- Import EOD (End of Day) Quotes ---------
-----------------------------------------------------------
-- Create table eod_quotes
-- NOTE: ticker and date will be the PK; volume numeric, and other numbers real (4
bytes)
-- NOTE: double precision and bigint will result in an import error on Windows
-- DROP TABLE public.eod_quotes;
CREATE TABLE public.eod_quotes
(
    ticker character varying(16) COLLATE pg_catalog."default" NOT NULL,
    date date NOT NULL,
    adj_open real,
    adj_high real,
    adj_low real,
    adj_close real,
    adj_volume numeric,
    CONSTRAINT eod_quotes_pkey PRIMARY KEY (ticker, date)
)
WITH (
    OIDS = FALSE
)
TABLESPACE pg_default;

ALTER TABLE public.eod_quotes
    OWNER to postgres;

-- Import eod.csv to the table - it will take some time (approx. 17 million rows)
```

```
-- Check!
SELECT * FROM eod_quotes LIMIT 10;
SELECT COUNT(*) FROM eod_quotes; -- 16,891,814


-- And let us join the view with the table, extract the "NULL" sector in NASDAQ,
store the results in a separate table
SELECT
ticker,date,company_name,market_cap,country,adj_open,adj_high,adj_low,adj_close,adj_v
olume
INTO eod_quotes_nasdaq_null_sector
FROM v_company_list C INNER JOIN eod_quotes Q ON C.symbol=Q.ticker
WHERE C.sector IS NULL AND C.stock_mkt_name='NASDAQ';

-- And let us join the view with the table, extract the "NULL" sector in NYSE, store
the results in a separate table
SELECT
ticker,date,company_name,market_cap,country,adj_open,adj_high,adj_low,adj_close,adj_v
olume
INTO eod_quotes_nyse_null_sector
FROM v_company_list C INNER JOIN eod_quotes Q ON C.symbol=Q.ticker
WHERE C.sector IS NULL AND C.stock_mkt_name='NYSE';

-- And let us join the view with the table, extract the "NULL" sector in AMEX, store
the results in a separate table
SELECT
ticker,date,company_name,market_cap,country,adj_open,adj_high,adj_low,adj_close,adj_v
olume
INTO eod_quotes_amex_null_sector
FROM v_company_list C INNER JOIN eod_quotes Q ON C.symbol=Q.ticker
WHERE C.sector IS NULL AND C.stock_mkt_name='AMEX';

-- Check
SELECT * FROM eod_quotes_nasdaq_null_sector;
SELECT * FROM eod_quotes_nyse_null_sector;
SELECT * FROM eod_quotes_amex_null_sector;

-- Adjust the PK by adding a constraint - properties will not work!
ALTER TABLE public.eod_quotes_nasdaq_null_sector
    ADD CONSTRAINT eod_quotes_nasdaq_null_sector_pkey PRIMARY KEY (ticker, date);

ALTER TABLE public.eod_quotes_nyse_null_sector
    ADD CONSTRAINT eod_quotes_nyse_null_sector_pkey PRIMARY KEY (ticker, date);

ALTER TABLE public.eod_quotes_amex_null_sector
    ADD CONSTRAINT eod_quotes_amex_null_sector_pkey PRIMARY KEY (ticker, date);



------------------------------------------------------------
-- Let us check the stock market data we have --------
------------------------------------------------------------

-- What is the date range?
SELECT min(date),max(date) FROM eod_quotes;
```

```
-- Really? How many companies have full data in each year?
SELECT date_part('year',date), COUNT(*)/252 FROM eod_quotes GROUP BY
date_part('year',date);

-- Let's decide on some practical time range (e.g. 2016-2021)
SELECT ticker, date, adj_close FROM eod_quotes WHERE date BETWEEN '2016-01-01' AND
'2021-03-26';

-- And create a (simple version of) view v_eod_quotes_2016_2021
-- DROP VIEW public.v_eod_quotes_2016_2021;
CREATE OR REPLACE VIEW public.v_eod_quotes_2016_2021 AS
 SELECT eod_quotes.ticker,
    eod_quotes.date,
    eod_quotes.adj_close
   FROM eod_quotes
  WHERE eod_quotes.date >= '2016-01-01'::date AND eod_quotes.date <= '2021-03-
26'::date;

ALTER TABLE public.v_eod_quotes_2016_2021
    OWNER TO postgres;

-- Check
SELECT min(date),max(date) FROM v_eod_quotes_2016_2021;

-- Let's download 2016-2021 of SP500TR from Yahoo
https://finance.yahoo.com/quote/%5ESP500TR/history?p=^SP500TR

-- An analysis of the CSV indicated that to make it compatible with eod
-- - all unusual formatting has to be removed
-- - a "ticker" column with the value SP500TR need to be added
-- - the volume column has to be updated (zeros are fine)
-- Import the (modified) CSV to a (new) data table eod_indices which reflects the
original file's structure
-- DROP TABLE public.eod_indices;
CREATE TABLE public.eod_indices
(
    symbol character varying(16) COLLATE pg_catalog."default" NOT NULL,
    date date NOT NULL,
    open real,
    high real,
    low real,
    close real,
    adj_close real,
    volume double precision,
    CONSTRAINT eod_indices_pkey PRIMARY KEY (symbol, date)
)
WITH (
    OIDS = FALSE
)
TABLESPACE pg_default;

ALTER TABLE public.eod_indices
    OWNER to postgres;
```

```
-- Import the csv SP500TR.csv

-- Check
SELECT * FROM eod_indices LIMIT 10;

-- Create a view analogous to our quotes view: v_eod_indices_2016_2021
-- DROP VIEW public.v_eod_indices_2016_2020;
CREATE OR REPLACE VIEW public.v_eod_indices_2016_2021 AS
 SELECT eod_indices.symbol,
    eod_indices.date,
    eod_indices.adj_close
   FROM eod_indices
   WHERE eod_indices.date >= '2016-01-01'::date AND eod_indices.date <= '2021-03-
26'::date;

ALTER TABLE public.v_eod_indices_2016_2021
    OWNER TO postgres;

-- CHECK
SELECT MIN(date),MAX(date) FROM v_eod_indices_2016_2021;

-- We can combine the two views using UNION which help us later (this will take a
while)
SELECT * FROM v_eod_quotes_2016_2021
UNION
SELECT * FROM v_eod_indices_2016_2021;



-------------------------------------------------------------------------
-- Next, let's prepare a custom calendar (using a spreadsheet) --------
-------------------------------------------------------------------------
-- We need a stock market calendar to check our data for completeness
-- https://www.nyse.com/markets/hours-calendars
-- Because it is faster, we will use Excel (we need market holidays to do that)
-- We will use NETWORKDAYS.INTL function
-- date, y,m,d,dow,trading (format date and dow!)
-- Save as custom_calendar.csv and import to a new table
-- DROP TABLE public.custom_calendar;
CREATE TABLE public.custom_calendar
(
    date date NOT NULL,
    y integer,
    m integer,
    d integer,
    dow character varying(3) COLLATE pg_catalog."default",
    trading smallint,
    CONSTRAINT custom_calendar_pkey PRIMARY KEY (date)
)
WITH (
    OIDS = FALSE
)
TABLESPACE pg_default;

ALTER TABLE public.custom_calendar
```

```
    OWNER to postgres;

-- Import the csv custom_calendar.csv

-- CHECK
SELECT * FROM custom_calendar LIMIT 10;

-- Let's add some columns to be used later: eom (end-of-month) and prev_trading_day
ALTER TABLE public.custom_calendar
    ADD COLUMN eom smallint;

ALTER TABLE public.custom_calendar
    ADD COLUMN prev_trading_day date;

-- CHECK
SELECT * FROM custom_calendar LIMIT 10;

-- Now let's populate these columns
-- Identify trading days
SELECT * FROM custom_calendar WHERE trading=1;

-- Identify previous trading days via a nested query
-- Update the table with new data
UPDATE custom_calendar
SET prev_trading_day = PTD.ptd
FROM (SELECT date, (SELECT MAX(CC.date) FROM custom_calendar CC WHERE CC.trading=1
AND CC.date<custom_calendar.date) ptd FROM custom_calendar) PTD
WHERE custom_calendar.date = PTD.date;

-- CHECK
SELECT * FROM custom_calendar ORDER BY date;

-- Identify the end of the month
-- Update the table with new data
UPDATE custom_calendar
SET eom = EOMI.endofm
FROM (SELECT CC.date,CASE WHEN EOM.y IS NULL THEN 0 ELSE 1 END endofm FROM
custom_calendar CC LEFT JOIN
(SELECT y,m,MAX(d) lastd FROM custom_calendar WHERE trading=1 GROUP by y,m) EOM
ON CC.y=EOM.y AND CC.m=EOM.m AND CC.d=EOM.lastd) EOMI
WHERE custom_calendar.date = EOMI.date;

-- CHECK
SELECT * FROM custom_calendar ORDER BY date;



-------------------------------------------------------------------
-- Determine the completeness of price or index data ------------
-------------------------------------------------------------------

-- Incompleteness may be due to when the stock was listed/delisted or due to errors
-- First, let's see how many trading days were there between 2016 and 2021
SELECT COUNT(*)
FROM custom_calendar
```

```
WHERE trading=1 AND date BETWEEN '2016-01-01' AND '2021-03-26';

-- Now, let us check how many price items we have for each stock in the same date
range
SELECT ticker,min(date) as min_date, max(date) as max_date, count(*) as price_count
FROM v_eod_quotes_2016_2021
GROUP BY ticker
ORDER BY price_count DESC;

-- Let's calculate the percentage of complete trading day prices for each stock and
identify 99%+ complete
-- Let's store the excluded tickers (less than 99% complete in a table)
SELECT ticker, 'More than 1% missing' as reason
INTO exclusions_2016_2021
FROM v_eod_quotes_2016_2021
GROUP BY ticker
HAVING count(*)::real/(SELECT COUNT(*) FROM custom_calendar WHERE trading=1 AND date
BETWEEN '2016-01-01' AND '2021-03-26')::real<0.99;

-- Also define the PK constraint for exclusions_2016_2021
ALTER TABLE public.exclusions_2016_2021
    ADD CONSTRAINT exclusions_2016_2021_pkey PRIMARY KEY (ticker);

-- Apply the same procedure for the indices and store exclusions (if any) in the same
table: exclusions_2016_2021
INSERT INTO exclusions_2016_2021
SELECT symbol, 'More than 1% missing' as reason
FROM v_eod_indices_2016_2021
GROUP BY symbol
HAVING count(*)::real/(SELECT COUNT(*) FROM custom_calendar WHERE trading=1 AND date
BETWEEN '2016-01-01' AND '2021-03-26')::real<0.99;

-- CHECK
SELECT * FROM exclusions_2016_2021;

-- Let combine everything we have (it will take some time to execute)
SELECT * FROM v_eod_indices_2016_2021 WHERE symbol NOT IN  (SELECT DISTINCT ticker
FROM exclusions_2016_2021)
UNION
SELECT * FROM v_eod_quotes_2016_2021 WHERE ticker NOT IN  (SELECT DISTINCT ticker
FROM exclusions_2016_2021);

-- Let's create a materialized view mv_eod_2015_2020
-- DROP MATERIALIZED VIEW public.mv_eod_2016_2021;
CREATE MATERIALIZED VIEW public.mv_eod_2016_2021
TABLESPACE pg_default
AS
 SELECT v_eod_indices_2016_2021.symbol,
    v_eod_indices_2016_2021.date,
    v_eod_indices_2016_2021.adj_close
   FROM v_eod_indices_2016_2021
  WHERE NOT (v_eod_indices_2016_2021.symbol::text IN ( SELECT DISTINCT
exclusions_2016_2021.ticker
           FROM exclusions_2016_2021))
UNION
```

```
 SELECT v_eod_quotes_2016_2021.ticker AS symbol,
    v_eod_quotes_2016_2021.date,
    v_eod_quotes_2016_2021.adj_close
   FROM v_eod_quotes_2016_2021
  WHERE NOT (v_eod_quotes_2016_2021.ticker::text IN ( SELECT DISTINCT
exclusions_2016_2021.ticker
           FROM exclusions_2016_2021))
WITH NO DATA;

ALTER TABLE public.mv_eod_2016_2021
    OWNER TO postgres;

-- We must refresh it (it will take time but it is one-time or infrequent)
REFRESH MATERIALIZED VIEW mv_eod_2016_2021 WITH DATA;

-- CHECK
SELECT * FROM mv_eod_2016_2021 LIMIT 10; -- faster
SELECT DISTINCT symbol FROM mv_eod_2016_2021; -- fast



-------------------------------------------------------
-- Calculate daily returns or changes -----------------
-------------------------------------------------------

-- We will assume the following definition R_1=(P_1-P_0)/P_0=P_1/P_0-1.0 (P:price,
i.e., adj_close)
-- First let us join the calendar with the prices (and indices)
SELECT EOD.*, CC.*
FROM mv_eod_2016_2021 EOD INNER JOIN custom_calendar CC ON EOD.date=CC.date;

-- Let's make another materialized view - this time with the returns
-- DROP MATERIALIZED VIEW public.mv_ret_2016_2021;
CREATE MATERIALIZED VIEW public.mv_ret_2016_2021
TABLESPACE pg_default
AS
 SELECT eod.symbol,
    eod.date,
    eod.adj_close / prev_eod.adj_close - 1.0::double precision AS ret
   FROM mv_eod_2016_2021 eod
     JOIN custom_calendar cc ON eod.date = cc.date
     JOIN mv_eod_2016_2021 prev_eod ON prev_eod.symbol::text = eod.symbol::text AND
prev_eod.date = cc.prev_trading_day
WITH NO DATA;

ALTER TABLE public.mv_ret_2016_2021
    OWNER TO postgres;

-- We must refresh it (it will take time but it is one-time or infrequent)
REFRESH MATERIALIZED VIEW mv_ret_2016_2021 WITH DATA;

-- CHECK
SELECT * FROM mv_ret_2016_2021 LIMIT 10;
```

```
-----------------------------------------------------------------
-- Identify potential errors and expand the exlusions list --------
-----------------------------------------------------------------

-- Let's explore first
SELECT min(ret),avg(ret),max(ret) from mv_ret_2016_2021;

-- Make an arbitrary decision how much daily return is too much (e.g. 100%), identify
such symbols
-- and add them to exclusions_2016_2021
INSERT INTO exclusions_2016_2021
SELECT DISTINCT symbol, 'Return higher than 100%' as reason FROM mv_ret_2016_2021
WHERE ret>1.0;

-- IMPORTANT: we have stored (materialized) views, we need to refresh them IN A
SEQUENCE!
REFRESH MATERIALIZED VIEW mv_eod_2016_2021 WITH DATA;
REFRESH MATERIALIZED VIEW mv_ret_2016_2021 WITH DATA;

-- We can continue adding exclusions for various reasons - remember to refresh the
stored views



-------------------------------------------------------------------------
-- Format price and return data for export to the analytical tool  --------
-------------------------------------------------------------------------

-- In order to export all data we will left-join custom_calendar with materialized
views
-- This way we will not miss a trading day even if there is not a single record
available
-- It is very important when data is updated daily
-- Daily prices export
SELECT PR.*
INTO export_daily_prices_2016_2021
FROM custom_calendar CC LEFT JOIN mv_eod_2016_2021 PR ON CC.date=PR.date
WHERE CC.trading=1;

-- Daily returns export
SELECT PR.*
INTO export_daily_returns_2016_2021
FROM custom_calendar CC LEFT JOIN mv_ret_2016_2021 PR ON CC.date=PR.date
WHERE CC.trading=1;

-- Export the csv daily_prices_2016_2021.csv and daily_returns_2016_2021.csv

-- Remove temporary (export_) tables because they are not refreshed
DROP TABLE export_daily_prices_2016_2021;
DROP TABLE export_daily_returns_2016_2021;



-----------------------------------------
```

```
-- Create a role for the database  --------
-----------------------------------------
-- rolename: stockmarketreadergp
-- password: read123
-- REVOKE ALL PRIVILEGES ON ALL TABLES IN SCHEMA public FROM stockmarketreader;
-- DROP USER stockmarketreadergp;
CREATE USER stockmarketreadergp WITH
       LOGIN
       NOSUPERUSER
       NOCREATEDB
       NOCREATEROLE
       INHERIT
       NOREPLICATION
       CONNECTION LIMIT -1
       PASSWORD 'read123';

-- Grant read rights (on existing tables and views)
GRANT SELECT ON ALL TABLES IN SCHEMA public TO stockmarketreadergp;

-- Grant read rights (for future tables and views)
ALTER DEFAULT PRIVILEGES IN SCHEMA public
   GRANT SELECT ON TABLES TO stockmarketreadergp;


-- Returns and portfolio will be analysed in R --
```

Appendix B – R Code

```
# Stock Market Case in R
rm(list=ls(all=T))

# We are going to perform most of the transformation tasks in R


# Connect to PostgreSQL ------------------------------------------------
require(RPostgres) # did you install this package?
require(DBI)
conn <- dbConnect(RPostgres::Postgres()
                  ,user="stockmarketreadergp"
                  ,password="read123"
                  ,host="localhost"
                  ,port=5432
                  ,dbname="stockmarket_GP"
)

# Custom calendar
qry<-"SELECT * FROM custom_calendar WHERE date BETWEEN '2015-12-31' AND '2021-03-26'
ORDER by date"
ccal<-dbGetQuery(conn,qry)

# Eod prices and indices
qry1="SELECT symbol,date,adj_close FROM eod_indices WHERE date BETWEEN '2015-12-31'
AND '2021-03-26'"
qry2="SELECT ticker,date,adj_close FROM eod_quotes WHERE date BETWEEN '2015-12-31'
AND '2021-03-26'
AND ticker IN ('ENS', 'PCTY', 'FCNCA', 'CBFV', 'MPW', 'ORAN', 'FWONK', 'LHCG', 'POR',
'AGM', 'MIELY', 'BURL', 'SMTOY', 'ESLOY', 'MLVF')"
eod<-dbGetQuery(conn,paste(qry1,'UNION',qry2))
dbDisconnect(conn)
rm(conn)

# Check
# Explore
head(ccal)
tail(ccal)
nrow(ccal)

head(eod)
tail(eod)
nrow(eod)

head(eod[which(eod$symbol=='SP500TR'),])

#We may need one more data item (for 2015-12-31)
eod_row<-data.frame(symbol='SP500TR',date=as.Date('2015-12-31'),adj_close=3821.60)
eod<-rbind(eod,eod_row)
tail(eod)
```

```
# Use Calendar ----------------------------------------------------
tdays<-ccal[which(ccal$trading==1),,drop=F]
head(tdays)
nrow(tdays)-1 # Trading days between 2016 and 2021

# Completeness ----------------------------------------------------
# Percentage of completeness
pct<-table(eod$symbol)/(nrow(tdays)-1)
selected_symbols_daily<-names(pct)[which(pct>=0.99)]
eod_complete<-eod[which(eod$symbol %in% selected_symbols_daily),,drop=F]

# Check
head(eod_complete)
tail(eod_complete)
nrow(eod_complete)


# Transform (Pivot) -----------------------------------------------
require(reshape2)
eod_pvt<-dcast(eod_complete, date ~ symbol,value.var='adj_close',fun.aggregate =
mean, fill=NULL)

# Check
eod_pvt[1:10,] # First 10 rows and all columns
ncol(eod_pvt) # Column count
nrow(eod_pvt)


# Merge with Calendar ---------------------------------------------
eod_pvt_complete<-
merge.data.frame(x=tdays[,'date',drop=F],y=eod_pvt,by='date',all.x=T)

# Check
eod_pvt_complete[1:10,] # First 10 rows and all columns
ncol(eod_pvt_complete)
nrow(eod_pvt_complete)

# Use dates as row names and remove the date column
rownames(eod_pvt_complete)<-eod_pvt_complete$date
eod_pvt_complete$date<-NULL # Remove the "date" column

# Re-check
eod_pvt_complete[1:10] # First 10 rows and all columns
ncol(eod_pvt_complete)
nrow(eod_pvt_complete)


# Missing Data Imputation -----------------------------------------
# We can replace a few missing (NA or NaN) data items with previous data
# Let's say no more than 3 in a row...
require(zoo)
eod_pvt_complete<-na.locf(eod_pvt_complete,na.rm=F,fromLast=F,maxgap=3)

# Re-check
eod_pvt_complete[1:10,] # First 10 rows and all columns
```

```
ncol(eod_pvt_complete)
nrow(eod_pvt_complete)


# Calculating Returns ----------------------------------------------------
require(PerformanceAnalytics)
eod_ret<-CalculateReturns(eod_pvt_complete)

# Check
eod_ret[1:10,] # First 10 rows and all columns
ncol(eod_ret)
nrow(eod_ret)

# Remove the first row
eod_ret<-tail(eod_ret,-1) # Use tail with a negative value

# Check
eod_ret[1:10,] # First 10 rows and all columns
ncol(eod_ret)
nrow(eod_ret)


# Check for extreme returns ----------------------------------------
# There is colSums, colMeans but no colMax so we need to create it
colMax <- function(data) sapply(data, max, na.rm = TRUE)

# Apply it
max_daily_ret<-colMax(eod_ret)
max_daily_ret #first 10 max returns

# And proceed just like we did with percentage (completeness)
selected_symbols_daily<-names(max_daily_ret)[which(max_daily_ret<=1.00)]
length(selected_symbols_daily)

# Subset eod_ret
eod_ret<-eod_ret[,which(colnames(eod_ret) %in% selected_symbols_daily),drop=F]

# Check
eod_ret[1:10,] #first 10 rows and all columns
ncol(eod_ret)
nrow(eod_ret)


# Tabular Return Data Analytics ------------------------------------------

# We will select 'SP500TR' and selected 15 tickers assigned to group members
# We need to convert data frames to xts (extensible time series)
Ra<-as.xts(eod_ret[,c('ENS', 'PCTY', 'FCNCA', 'CBFV', 'MPW', 'ORAN', 'FWONK', 'LHCG',
'POR', 'AGM', 'MIELY', 'BURL', 'SMTOY', 'ESLOY', 'MLVF'),drop=F])
Rb<-as.xts(eod_ret[,'SP500TR',drop=F]) #benchmark

head(Ra)
tail(Ra)
head(Rb)
```

```
# And now we can use the analytical package...

# Stats
table.Stats(Ra)

# Distributions
table.Distributions(Ra)

# Returns
table.AnnualizedReturns(cbind(Rb,Ra),scale=252)

# Annualized Sharpe - Risk adjusted return (higher the better) (higher return lower
risk)

# Accumulate Returns
acc_Ra<-Return.cumulative(Ra);acc_Ra
acc_Rb<-Return.cumulative(Rb);acc_Rb

# Capital Assets Pricing Model
table.CAPM(Ra,Rb)

# Beta (measure of risk - slope parameter of SLR between asset and market), alpha is
intercept


# Graphical Return Data Analytics ----------------------------------------

# Cumulative returns chart
chart.CumReturns(Ra,legend.loc = 'topleft')
chart.CumReturns(Rb,legend.loc = 'topleft')

# Box plots
chart.Boxplot(cbind(Rb,Ra))
chart.Drawdown(Ra,legend.loc = 'bottomleft')


# MV Portfolio Optimization ----------------------------------------------

# Withhold the last 58 trading days (all of 2021 data)
Ra_training<-head(Ra,-58)
Rb_training<-head(Rb,-58)

# Cummulative returns for Range 1
acc_Ra_training<-Return.cumulative(Ra_training);acc_Ra_training
chart.CumReturns(Ra_training,legend.loc = 'topleft')

# Use the last 58 trading days for testing (all of 2021 data)
Ra_testing<-tail(Ra,58)
Rb_testing<-tail(Rb,58)

# Optimize the MV (Markowitz 1950s) portfolio weights based on training
table.AnnualizedReturns(Rb_training)
mar<-mean(Rb_training) #we need daily minimum acceptable return
```

```
require(PortfolioAnalytics)
require(ROI) # make sure to install it
require(ROI.plugin.quadprog)  # make sure to install it
pspec<-portfolio.spec(assets=colnames(Ra_training))
pspec<-add.objective(portfolio=pspec,type="risk",name='StdDev')
pspec<-add.constraint(portfolio=pspec,type="full_investment")
pspec<-add.constraint(portfolio=pspec,type="return",return_target=mar)
#pspec<-add.constraint(portfolio=pspec,type="long_only")

# Optimize portfolio
opt_p<-optimize.portfolio(R=Ra_training,portfolio=pspec,optimize_method = 'ROI')

# Extract weights (negative weights means shorting)
opt_w<-opt_p$weights

# Weights for Range 1 (2016-2020)
round(opt_w, 4)

# Sum of weights for Range 1 (2016-2020)
sum(round(opt_w, 4))

# Apply weights to test returns
Rp<-Rb_testing

# Define new column that is the dot product of the two vectors
Rp$ptf<-Ra_testing %*% opt_w

# Check
head(Rp)
tail(Rp)

# Compare basic metrics
table.AnnualizedReturns(Rp)

#std dev is sqrt(var) that corresponds to risk, less variance less risk

# Chart Hypothetical Portfolio Returns -----------------------------------
chart.CumReturns(Rp,legend.loc = 'bottomright')
```