Stock Market Portfolio Analysis and Optimization

Titan Quantitative Strategies

Team 2

Mohith Bandi, Venkata Sai Manish Gupta Bontala, Pranavi Chinthireddy

Nandini Koppunuru and Sai Srujana Ravipati

Department of Information Systems & Decision Sciences, California State University

Fullerton ISDS 570: Business Data Transformation

Dr. Pawel Kalczynski

May 11, 2025

Table of Contents

Executive Summary

Titan Quantitative Strategies (TQS) is a data-driven investment analysis firm that specializes in constructing optimized portfolios using rigorous statistical and financial modeling techniques. For this project, our client, Dr. Pawel Kalczynski from California State University Fullerton, requested an analytical evaluation of a custom stock portfolio designed using Mean-Variance (MV) optimization principles. The goal was to assess its performance relative to the S&P 500 Total Return Index (SP500TR) under both historical and forward-looking market conditions.

TQS curated a portfolio of 15 stocks across multiple sectors, using historical daily data from 2016 to 2020 for training. Our approach incorporated a robust ETL process, ensuring data quality and completeness. The optimization was performed using the MAR (Minimum Acceptable Return) derived from the SP500TR, and portfolio weights were determined accordingly. The custom portfolio was then tested out-of-sample against SP500TR for Q1 2021.

The results revealed that the optimized portfolio underperformed the benchmark, with an annualized return of -10.16% compared to 29.87% for the SP500TR. Despite demonstrating slightly lower volatility (13.97% vs. 16.23%), the portfolio's negative Sharpe ratio of -0.7271 indicated poor risk-adjusted returns. These findings highlight the limitations of static optimization under shifting market conditions and reinforce the need for adaptive, dynamic portfolio strategies in real-world investing.

Project Overview

- *Purpose and Objectives*

Titan Quantitative Strategies upholds a rigorous standard in designing portfolios that balance return potential with controlled variance risk. The goal of this project is to build and evaluate a 15-stock portfolio utilizing the Mean-Variance (MV) optimization model. Utilizing historical data from 2016 to 2020, we seek to enhance the portfolio's risk-adjusted return and compare its performance to the S&P 500 Total Return Index (SP500TR) for the first quarter of 2021.

- *Data Range Selection*

The preliminary data gathering incorporated daily statistics from three prominent stock exchanges—NASDAQ, NYSE, and AMEX—spanning the years 2015 to 2020. Our optimization concentrated on conforming to the minimum acceptable return (MAR) for the SP500TR index, particularly during the period from 2016 to 2020, while guaranteeing our portfolio complied with this requirement. To calculate the MAR and determine portfolio weights, we adjusted the dataset to feature only trading days from January 1, 2016, to December 31, 2020. For the purpose of backtesting, we obtained daily trading information for the chosen stocks and the benchmark during the timeframe from January 1 to March 26, 2021. The information utilized for this analysis was obtained from Yahoo! Finance (finance.yahoo.com) and internal end-of-day (EOD) stock quote files supplied by our project sponsor, Dr. Kalczynski.

- *Composition of Portfolio*

TQS's portfolio consists of 15 carefully selected stocks. Each of the five team members evaluated three stocks, verifying data integrity and ensuring that no more than 1% of the trading data was missing within the selected date range. The final portfolio represents a broad mix of sectors, supporting diversification. A summary of the selected stocks, including their tickers, company names, and respective industries, is provided in the table below.

Stock Ticker	Company Name	Sector
LYV	Live Nation Entertainment Inc.	Communication Services
KUBTY	Kubota Corporation	Industrials
RDY	Dr. Reddy's Laboratories Ltd.	Healthcare
IMKTA	Ingles Markets, Inc.	Consumer Defensive
SFBS	ServisFirst Bancshares Inc.	Financial Services
MWA	Mueller Water Products Inc.	Industrials
NWBI	Northwest Bancshares Inc.	Financial Services
DOC	Healthpeak Properties Inc.	Real Estate
DLR	Digital Realty Trust Inc.	Real Estate
CIZN	Citizens Holding Company	Financial Services
GS	The Goldman Sachs Group, Inc.	Financial Services
MASI	Masimo Corporation	Healthcare
NFG	National Fuel Gas Co.	Utilities
UPLD	Upland Software Inc.	Technology
MYRG	MYR Group Inc.	Industrials

Framework for Data Engineering

- **Data Collection and Data Preparation:** The S&P 500 Total Return Index (SP500TR) and the daily stock price data for the 15 selected equities were collected. To ensure data integrity and correctness, an ETL process was established to extract daily results. A customized trading calendar was made to take into account market holidays and accurately match stock information with days that are allowed for trade.
- **Portfolio Optimization:** The Mean-Variance (MV) optimization framework was used to derive optimal stock weights, constrained by a Minimum Acceptable Return (MAR) benchmarked to SP500TR. The resulting portfolio included a combination of positive and negative weights, reflecting the model's mathematical solution for return-risk tradeoffs.
- **Back testing:** The optimized portfolio was evaluated on out-of-sample data spanning January to March 2021. This stage tested how well the theoretically optimized portfolio would perform under real market conditions.
- **Performance Analysis:** The portfolio's performance was assessed using key metrics—annualized return, standard deviation, and Sharpe ratio. While the portfolio exhibited lower volatility than SP500TR, it delivered negative returns and a negative Sharpe ratio, indicating weaker risk-adjusted performance during the test window.

Architecture of Database

Design of Schema

The design of our PostgreSQL database stockmarket consists of five well-structured tables and several supporting views, built following relational database principles to ensure normalization, consistency, and integrity. Each table was tailored for a specific purpose ranging from storing company details and end-of-day quotes to maintaining a custom trading calendar and exclusions for low-quality data. Constraints such as PRIMARY KEY and NOT NULL were applied to enforce data validity and uniqueness. These integrity controls were especially crucial, as the accuracy of financial data directly impacted return computations, volatility assessments, and the reliability of performance metrics like the Sharpe ratio. The database structure supported efficient querying and secure access for downstream portfolio analysis and modeling.

- ***Tables***
  - **stock_mkt**: This table serves as a special central repository for the names of stock markets by storing the list of stock markets (NASDAQ, NYSE, AMEX, etc.).
  - **company_list**: This table includes details about firms, such as their name, market capitalization, and year of initial public offering. It ensures data efficiency and integrity by establishing a connection between businesses and their individual stock exchanges.
  - **eod_quotes**: This table provides a crucial initial basis for financial data analysis by capturing end-of-day trading information for equities, including price.
  - **custom_calendar**: This table keeps track of a trading calendar that is based on NYSE activity. It includes holidays and indicators like prior trading days and the end of the month. By doing this, distortions brought on by non-trading times are avoided and price data is guaranteed to be in line with legitimate trading days.
  - **exclusions_2016_2021**: This table records and oversees trade days that were not included in the analysis because of issues with data quality. This table removes inaccurate or partial records to ensure the accuracy of calculations made later.
- ***Views***
  - **v_company_list**: This view is a condensed form of the company_list table that does not include characteristics that are not necessary. This view preserves access to important company information while streamlining query performance.
  - **v_eod_quotes_2016_2021**: This view improves efficiency and concentrates analysis on the specified study period by filtering the eod_quotes table to contain only pertinent columns and records from 2016 to 2021.
  - **Materialized Views:**
    - **mv_eod_2016_2021**: A materialized view that stores filtered quote data for selected tickers, as determined by the exclusions_2016_2021 table. This

structure supports faster access and consistent preprocessing for time-series analysis.

- **mv_ret_2016_2021**: This materialized view greatly improves the performance of return-based computations during portfolio optimization and analysis by storing precomputed daily returns for the chosen stocks.

To preserve referential integrity, the tables are connected to one another. A list of those relationships can be seen below:

- **Company_list and stock_mkt:** There is a one-to-many relationship between the aforementioned tables. There may be more than one firm name in company_list linked to each name in stock_mkt.
- **Company_list and eod_quotes:** The symbol column establishes a one-to-many relationship between the company_list and eod_quotes tables. All trading information in the eod_quotes table is guaranteed to match firms in the company_list table thanks to this relationship.
- **eod_quotes** and **custom_calendar**: Using their corresponding date columns, the custom_calendar table aligns the eod_quotes data with legitimate trading days. Only pertinent dates are considered in any study utilizing the date field because to this relationship.

Data Quality Checks

To guarantee data accuracy, consistency, and completeness, our database stockmarket uses exclusion tables, constraints, and data cleansing. Data integrity was essential to guaranteeing our ultimate analysis was flawless.

- ***Constraints*:**
  - **Primary Keys**: The primary key constraint guarantees that every table contains distinct and recognizable rows, such as names of distinct stock markets.
  - **NOT NULL Constraints**: By imposing non-null constraints for necessary columns, such as stock market names or corporate symbols, the NOT NULL constraint makes sure that no data is missing.
- ***Data Cleaning & Transformation:***
  - **Trimming:** By eliminating extraneous whitespace from strings like stock market names and corporate names, trimming different columns guarantees uniformity.
    - **Type Conversion**: Converting data into the proper format.

- ***Exclusions Table****:*
    - As they say, "garbage in, garbage out." Garbage is not included in our study in this table. The correctness and dependability of the data utilized in the study are improved by removing records that have problems.

Management of Access

To guarantee safe and role-appropriate data usage, a strong access control mechanism was incorporated into the development of the stockmarket database. A dedicated read-only user, stockmarketreadergp, was created to restrict modification privileges while enabling full access for analysis and reporting. This approach minimized the risk of security breaches and unintentional data alterations, aligning with best practices for controlled data environments in academic and professional settings.

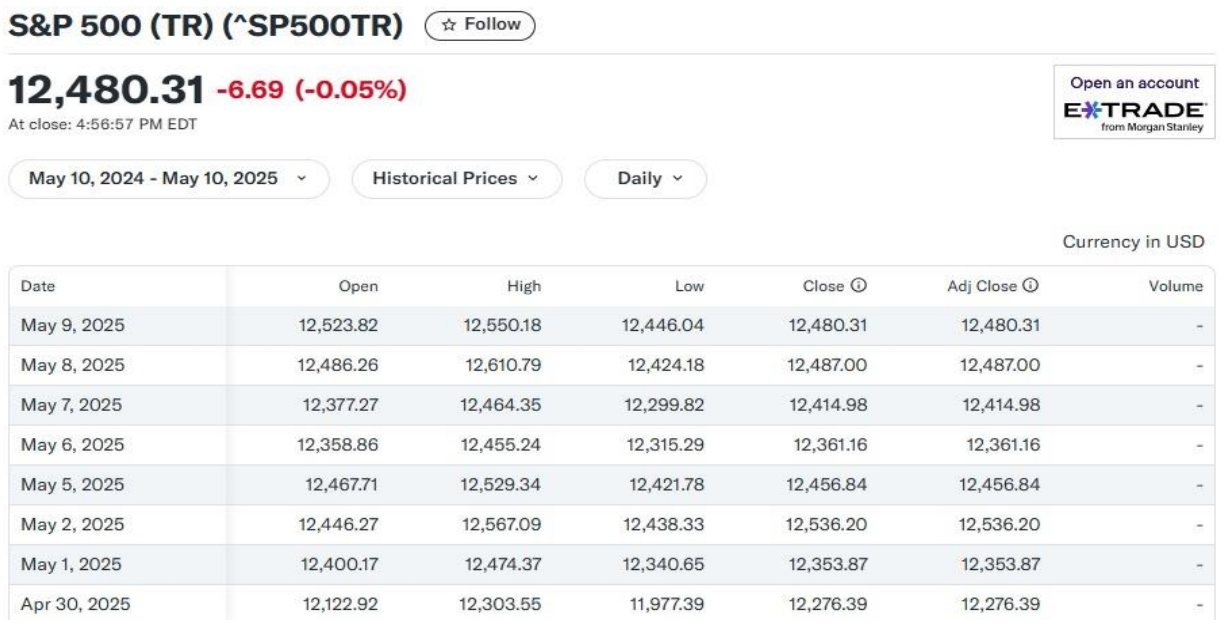
ETL Implementation

Data Collection Process

Stock market data was obtained from two sources: Yahoo Finance and a dataset provided by Dr. Kalczynski.

- **Yahoo Finance**: A mostly used platform for accessing historical financial data, including indices such as the SP500TR. The data was retrieved directly from the website, which is freely accessible without requiring login credentials.
- **Dr. K's Dataset**: This dataset included essential attributes such as financial data, trading dates, company names, and ticker symbols.

The data was filtered from Yahoo Finance to show daily historical prices within the specified time frame. It was then copied into an Excel file, cleaned and formatted as needed, and subsequently imported into the PostgreSQL database using the platform's Import feature.



S&P 500 (TR) (^SP500TR) ☆ Follow

12,480.31 -6.69 (-0.05%)

At close: 4:56:57 PM EDT

Open an account E*TRADE from Morgan Stanley

May 10, 2024 - May 10, 2025 | Historical Prices | Daily

Currency in USD

Date	Open	High	Low	Close ⓘ	Adj Close ⓘ	Volume
May 9, 2025	12,523.82	12,550.18	12,446.04	12,480.31	12,480.31	-
May 8, 2025	12,486.26	12,610.79	12,424.18	12,487.00	12,487.00	-
May 7, 2025	12,377.27	12,464.35	12,299.82	12,414.98	12,414.98	-
May 6, 2025	12,358.86	12,455.24	12,315.29	12,361.16	12,361.16	-
May 5, 2025	12,467.71	12,529.34	12,421.78	12,456.84	12,456.84	-
May 2, 2025	12,446.27	12,567.09	12,438.33	12,536.20	12,536.20	-
May 1, 2025	12,400.17	12,474.37	12,340.65	12,353.87	12,353.87	-
Apr 30, 2025	12,122.92	12,303.55	11,977.39	12,276.39	12,276.39	-

Dr. Kalczynski's dataset had a limited date range, with records available up to March 26, 2021. To ensure alignment, SP500TR data was gathered separately to match this range. The dataset, which included financial information for equities listed on the AMEX, NASDAQ, and NYSE exchanges, was made available in three CSV files. The identical procedure used for the Yahoo Finance data was used to import these files into the PostgreSQL database using the Import function. For consistency, the SP500TR index data was appended to Dr. K's dataset, and to preserve chronological continuity, a missing record for December 31, 2015, was manually added.

The RPostgres package was used to access the consolidated dataset, which was kept in the PostgreSQL environment. The DBI package's dbConnect function was used to create a connection. The stockmarketreadergp user was used to limit write access and provide safe, read-only data activities in accordance with our access management protocol. Only pertinent tickers were included in the SQL queries that were created to extract pertinent symbols and trading data. These searches were then further filtered to retrieve a date span from 2015-12-31 to 2021-03-26.

Quality Assurance Measures

Multiple steps were taken to ensure the accuracy, completeness, and reliability of the dataset. The initial quality control measure involved refining SQL queries to limit the data scope strictly to the specified trading period and the relevant tickers assigned in HW2. By explicitly filtering the data at the query level, the risk of including irrelevant or extraneous records was minimized.

Following data extraction, the dataset was examined to confirm its structural integrity and expected size. Functions such as head(), tail(), and nrow() were employed in R for preliminary inspection. These were supplemented with explicit data type declarations and dimensional checks after key transformation stages to ensure consistency.

Imputation was done using the "last observation carried forward" (LOCF) method and the zoo package in R to deal with missing values. To preserve the validity of the imputed data, this was limited to intervals of no more than three trading days in a row.

Finally, the table() function was used to assess data completeness at the symbol level. For analysis, only symbols with a data completeness of at least 99% were kept. By excluding records that were noisy or untrustworthy, this threshold made guaranteed that the final dataset only contained high-frequency trading symbols.

Trading Calendar Merging

A custom trading calendar was developed to align all stock data with official trading days on the NYSE. This calendar was constructed using Excel and filtered to exclude non-trading dates such as weekends and market holidays. Integrating this reference calendar into the dataset helped ensure temporal consistency and eliminated the risk of analytical distortion caused by invalid or missing dates.

Portfolio Construction

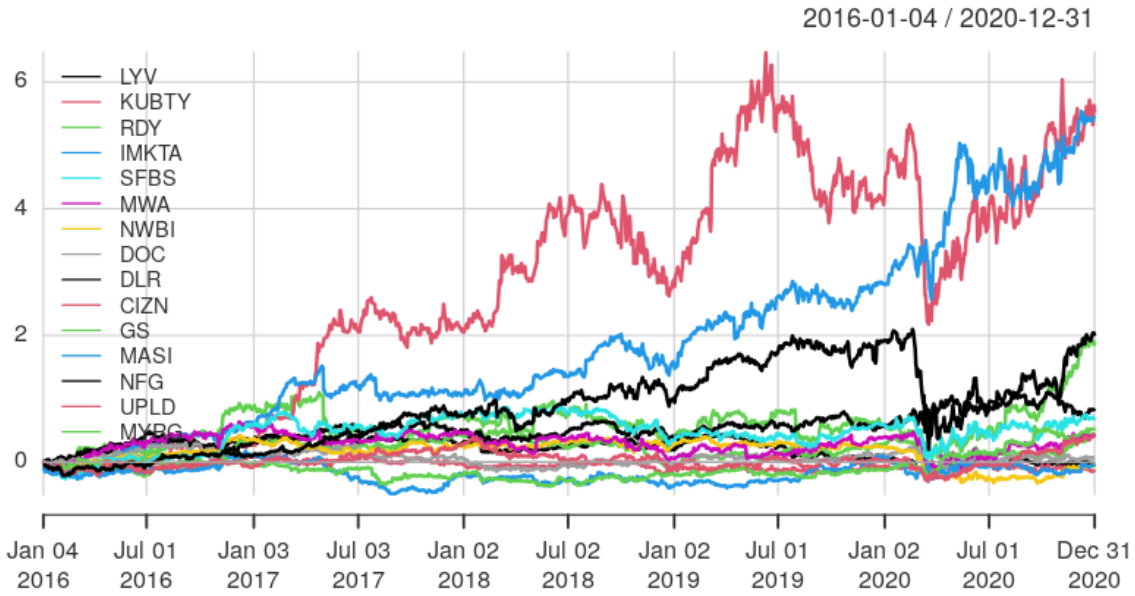
Stock Evaluation Criteria

We performed a thorough analysis of stock performance using historical end-of-day trading data from 2016 to 2021 using the cleaned and preprocessed dataset, with all exclusions applied. Adjusted open, high, low, and close prices as well as trade volume were important factors. We were able to assess the volatility, trends, and past performance of every stock chosen for the portfolio thanks to this extensive dataset.

Return Calculations

Daily and cumulative returns were computed to assess both short-term fluctuations and long-term growth trajectories. SQL queries were used to calculate percentage changes in adjusted closing prices for each trading day. These return calculations formed the foundation for comparative performance analysis and informed the weight allocation in the optimization model.

The figure below presents the cumulative return trends of all 15 selected stocks from January 2016 through December 2020.



The line plot shows the cumulative return (normalized) of each stock over the period from January 4, 2016, to December 31, 2020.

Observations:

- MASI (Masimo Corp) and RDY (Dr. Reddy's Labs) showed significant growth, reaching 5–6 times their original value.
- IMKTA and KUBTY also performed quite well, crossing a value of 2.
- Some stocks like GS, NWBI, and DLR had steady moderate gains.
- A few like SFBS, MYRG, and GS underperformed or remained flat.

Market Correlation Study

A market correlation analysis was performed to assess how closely the returns of each individual stock aligned with the broader market, represented by the S&P 500 Total Return Index (SP500TR). Using return data from 2016 to 2021, we measured the degree of co-movement between each stock and the benchmark index. This analysis provided insight into each asset's sensitivity to overall market trends and informed diversification decisions within the portfolio.

Weight Distribution

Initial stock weights were informed by market capitalization data, serving as a baseline for allocation by reflecting each company's relative size. Company-level financial data was merged with market cap values to calculate proportional weights. These baseline weights were further optimized through the Mean-Variance framework to derive final portfolio allocations. The optimized weights and sum of these weights for the 15 selected stocks over the 2016–2020 training period are listed below (rounded to four decimal places). The weights were validated to ensure that they sum to one.

Sum of weights: [1] 0.9999

Tickers	Optimized Weights
LYV	0.0060
KUBTY	0.1431
RDY	0.1406
IMKTA	0.0642
SFBS	-0.0655
MWA	-0.0228
NWBI	0.1265

DOC	0.0143
DLR	0.1409
CIZN	0.1830
GS	-0.0496
MASI	0.2007
NFG	0.0771
UPLD	0.0593
MYRG	-0.0179

This final table shows the optimized allocation weights based on your return and risk model (likely from a Mean-Variance Optimization approach). The goal here is to maximize return for a given level of risk.

Key insights:

- MASI (20.07%), CIZN (18.3%), and KUBTY (14.31%) received the highest weights, indicating they are expected to contribute the most to a risk-optimized portfolio.
- Negative weights (e.g., SFBS: -6.55%, GS: -4.96%) suggest a short position, meaning the model expects these stocks to reduce overall portfolio risk or drag on performance.
- LYV (0.6%), DOC (1.43%), and UPLD (5.93%) received minor weights, indicating limited but positive contribution to the portfolio.

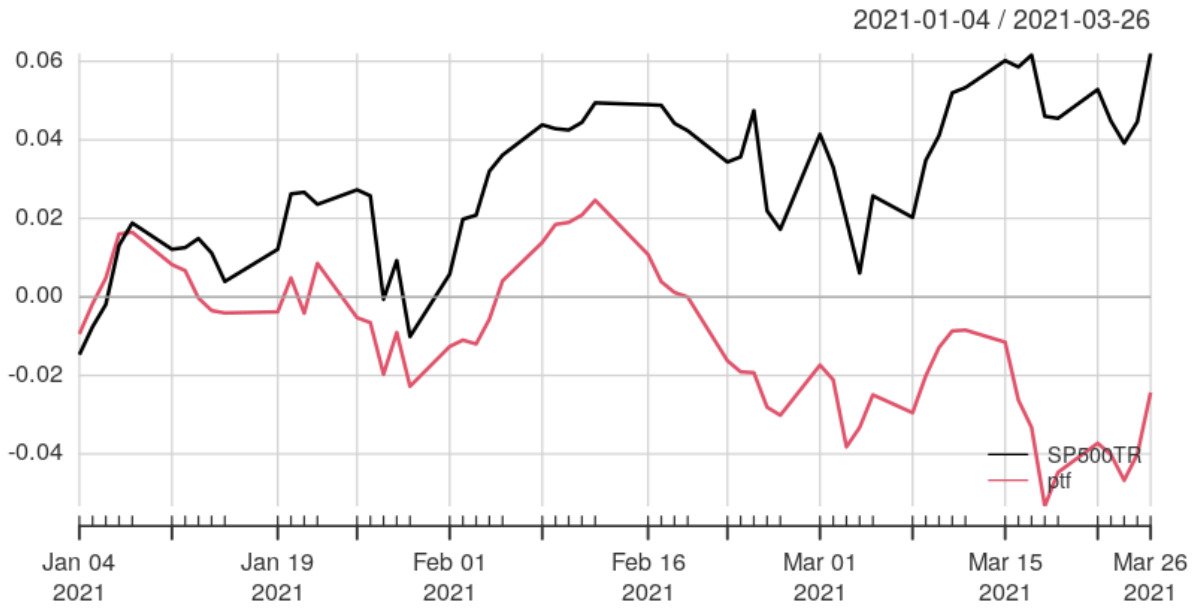
Performance Evaluation

Out-of-Sample Performance

To evaluate the robustness of our investment strategy, we divided the data into a training period (2016–2020) and a testing period (January to March 2021). This out-of-sample testing approach allowed us to assess how well the optimized portfolio performed under new, unseen market conditions.

SP500TR Benchmark Comparison

Over the whole out-of-sample period, which ran from January 4, 2021, to March 26, 2021, cumulative returns for the optimized portfolio and the SP500TR benchmark were computed and compared.This comparison provides insight into the performance of our actively managed strategy relative to a passive index investment. As visualized in the chart below, the SP500TR outperformed the optimized portfolio, underscoring the limitations of the static optimization approach during this particular market phase.



Risk-Adjusted Returns

To evaluate the portfolio's performance on a risk-adjusted basis, we calculated key annualized metrics for both the optimized portfolio and the SP500TR benchmark over the out-of-sample testing period (January to March 2021). These metrics included annualized return, standard deviation, and the Sharpe ratio, assuming a 0% risk-free rate.

Parameters / Symbol	**SP500TR**	**Portfolio**
Annualized Return	0.2987	-0.1016
Annualized Standard Deviation	0.1623	0.1397
Annualized Sharpe (Rf = 0%)	1.8399	-0.7271

The annualized performance measures for our custom portfolio and the SP500TR (S&P 500 Total Return Index) are shown in this table. Below is a thorough breakdown of every metric:

- **Annualized Return**: The SP500TR delivered a positive annualized return of 29.87%, while the optimized portfolio yielded a negative return of -10.16%, indicating a substantial underperformance during the testing window.
- **Annualized Standard Deviation**: The portfolio exhibited slightly lower volatility (13.97%) compared to the benchmark (16.23%), suggesting a more stable return profile despite the decline.
- **Annualized Sharpe Ratio (assuming 0% risk-free rate)**: The Sharpe ratio for the SP500TR was 1.8399, indicating strong risk-adjusted returns. In contrast, the portfolio's negative Sharpe ratio of -0.7271 reflects poor return performance relative to its risk, highlighting a misalignment between the optimization output and market conditions.

Performance of Portfolio Against Index

Although the optimized portfolio demonstrated lower volatility compared to the SP500TR benchmark, its negative return of -10.16% and a Sharpe ratio of -0.7271 indicate underperformance on a risk-adjusted basis. In contrast, the SP500TR delivered a strong 29.87% return with a Sharpe ratio of 1.8399, underscoring the portfolio's failure to translate theoretical stability into real-world gains. This outcome reveals the limitations of static, historically based models like Mean-Variance Optimization, which may falter when market conditions shift. To enhance future performance, strategies such as dynamic rebalancing, inclusion of alternative risk factors, or regime-switching models should be considered to better align with evolving market dynamics.

Appendix A – SQL Code

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

--  First, created a database "stockmarket" 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
('LYV','KUBTY','RDY','IMKTA','SFBS','MWA','NWBI','DOC','DLR','CIZN','GS','MASI','NFG',
'UPLD','MYRG','SP500TR')"

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('LYV','KUBTY','RDY','IMKTA','SFBS','MWA','NWBI','DOC','DLR','CIZN','G
S','MASI','NFG','UPLD','MYRG','SP500TR'),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')
```