Data Analytics

The Description and Analysis of the Historical Performance of Berkshire Hathaway

Yuchen LU

June 5, 2023

Table of content

Introduction

Warren Buffett is one of the most successful and renowned investors in the world. He was born on August 30, 1930, in Omaha, Nebraska, United States. Buffett is the chairman and CEO of Berkshire Hathaway, a multinational conglomerate holding company based in Omaha.

Buffett's investment philosophy is characterized by value investing and long-term thinking. He is known for his ability to identify undervalued companies with strong fundamentals and holding them for extended periods. Buffett has achieved remarkable success through his disciplined approach to investing, which has earned him the nickname "Oracle of Omaha."

Berkshire Hathaway, under Buffett's leadership, has become a major player in various industries, including insurance, energy, manufacturing, retail, transportation, and more. The company holds a diversified portfolio of subsidiary companies, including GEICO, Dairy Queen, Duracell, BNSF Railway, and many others. Berkshire Hathaway also owns significant stakes in well-known public companies, such as Apple, Coca-Cola, and Bank of America.

In this project we will use a dataset composed of five categories: the data of the historical stock price of Berkshire Hathaway (BRK_A), the significant stakes that Berkshire Hathaway held in 45 companies, the stock price of the companies since the time that Buffet has had holdings, the index fund prices of SnP500 and Nasdaq100 (as represented by QQQ), and complementary information about the companies in hold by BRK.

The primary goal is to describe the historical performance and the portfolio of Berkshire Hathaway, based on which we will undertake some exploratory analysis. Additionally, the project aims to develop a Machine Learning model to predict the future price tendency of BRK (settled in various time point in the past).

Plan for the Project

1- Jira planification of the different steps needed to complete the project

2- Export the dataset as csv files from the website

3- Use python to clean the dataset and prepare it for the exploratory analysis

4- Use python to build plots also for exploratory analysis purposes

5- Export the clean dataset to MySQL and build a database

6- Build an ERD for the different tables in the database

7- Use MySQL to run queries for exploratory analysis purposes

8- Encoding the dataset to prepare for modeling

9- Build a Machine Learning model and measure its prediction outcomes

Data source, Data collection, and Metadata

The project exploits multiple data source, which was exported as csv file. Most of the data files we used could be found in the following link: https://www.kaggle.com/datasets/tomasmantero/warren-buffett-us-stock-companies?resource=download

Entitled "Warren Buffett US Stock Companies", this dataset is published by Tomas Mantero 3 years ago. Among the files uploaded, what I have exported for my project include: 45 files contain the U.S. stocks owned by Berkshire Hathaway, 1 file contains the historical data of Berkshire Hathaway, Class A stock (BRK_A), 1 file contains the major holdings of BRK on October 18, 2020, and 1 file contains the list of all the companies with additional information.

To make the dataset more complete, I exported from Yahoo Finance as well as another Kaggle link on Apple stock (https://www.kaggle.com/datasets/tarunpaparaju/apple-aapl-historical-stock-data) : more ancient historical data of BRK_A, the historical data of two indexes, i.e. SnP500 and Nasdaq100 (QQQ).

Bellow you can find several tables describing the attributes of the different datafiles in this dataset, from columns names and types to row number.

BRK_A Stock Price (8 columns, 10887 rows)

Symbol	Date	Open	High	Low	Close	Adj Close	Volume
The abbreviation of the company name. Along with "Date", this column serves as the composite primary key once the file is imported into MySQL	All the working days of the American financial market from March 17, 1980, to May 19, 2023	Open stock price of the day	Highest stock price of the day	Lowest stock price of the day	Close stock price of the day, used in this project as the default price of the day	Adjusted closing price refers to the price of the stock after paying off the dividends	The exchange volume of the stock through the day

The Holdings and Stakes of Berkshire Hathaway in 45 companies (7 columns, 45 rows)

Date	Name	Symbol	Holdings	Market Price	Value	Stake
There is a single value, i.e. October 18, 2020. Along with "Symbol", this column serves as the composite primary key in MySQL	The official names of the companies in which BRK has holdings	The abbreviation of the company name	The number of stock shares that BRK possessed of this company	The market price per share of the company	Holdings * Market Price	The percentage of BRK's ownership in the company

The stock price of the companies since the time that Buffet has had holdings (8 columns, 129590 rows)

Symbol	Date	High	Low	Open	Close	Volume	Adj Close
The abbreviation of 45 company names. Along with "Date", this column serves as the composite primary key once the file is imported into MySQL	All the working days of the American financial market since the date that Buffet has had holdings	Highest stock price of the day	Lowest stock price of the day	Open stock price of the day	Close stock price of the day, used in this project as the default price of the day	The exchange volume of the stock through the day	Adjusted closing price refers to the price of the stock after paying off the dividends

The index fund prices of SnP500 and Nasdaq100 (3 columns, 8864 rows)

Symbol	Date	Close				
The abbreviation of 2 indexes. Along with "Date", this column serves as the composite primary key once the file is imported into MySQL	All the working days of the American financial market from October 18, 2005, to May 25, 2023	Close stock price of the day, used in this project as the default price of the day				

Complementary information about the companies in hold by BRK (5 columns, 45 rows)

Name	Symbol	Sector	Industry	Num_Employee		
The names of the 45 companies	**The abbreviation of the company name, which serves as the primary key in MySQL**	Which sector the company belongs to	Which industry (of the sector) the company belongs to	Number of employees in the company		

Data Cleaning

There are three main tasks for Data cleaning. The first is to get rid of unnecessary symbols in order to change the type of the values.

	Name	Symbol	Holdings	Market Price	Value	Stake
0	Amazon.com, Inc.	AMZN	5.333000e+03	$3,272.71	$1,745,336,243	0.1%
1	American Express Company	AXP	1.516107e+08	$104.91	$15,905,478,537	18.8%
2	Apple Inc.	AAPL	1.003466e+09	$119.02	$119,432,554,741	5.9%
3	Axalta Coating Systems Ltd	AXTA	2.353504e+07	$25.64	$603,438,451	10.0%
4	Bank of America Corp	BAC	1.032852e+09	$24.24	$25,036,332,625	11.9%
5	Bank of New York Mellon Corp	BK	7.434686e+07	$38.02	$2,826,667,769	8.4%
6	Barrick Gold Corp	GOLD	2.091870e+07	$27.57	$576,728,587	1.2%
7	Biogen Inc	BIIB	6.430220e+05	$280.01	$180,052,590	0.4%
8	Charter Communications Inc	CHTR	5.213461e+06	$633.92	$3,304,917,197	2.5%
9	Coca-Cola Co	KO	4.000000e+08	$50.03	$20,012,000,000	9.3%
10	Costco Wholesale Corporation	COST	4.333363e+06	$381.54	$1,653,351,319	1.0%

As shown in the above screenshot, the values of the columns “Market Price” and “Value” contain “$” and “,” which prevent the shift of the data type from objects to float. We used the code below to solve the problem.

```
wb_nototal['Market Price'] = wb_nototal['Market Price'].str.replace(',', '')
```

```
wb_nototal['Market Price'] = wb_nototal['Market Price'].str.replace('$', '', regex=True)
```

```
wb_nototal['Market Price'] = wb_nototal['Market Price'].astype(float)
```

The second task is to find out the hidden non-SCII characters which prevent the export of the files to MySQL before replacing them with SCII counterparts. The two screenshots below demonstrate our solutions to do the research and replacement. We introduced a function to find the non-SCII characters / to check whether all non-SCII character have been removed, and then a loop to do the replacement job.

```python
def check_dataframe_for_non_ascii(df):
    non_ascii = []
    for column in df.columns:
        for value in df[column]:
            #if has_non_ascii(str(value)):
            if (str(value).isascii()==False):
                non_ascii.append(value)
    return non_ascii


# Usage example

contains_non_ascii = check_dataframe_for_non_ascii(PROFILE)
print(f"The DataFrame contains non-ASCII characters: {contains_non_ascii}")
```

```
The DataFrame contains non-ASCII characters: ['Banks–Diversified', 'Drug Manufacturers–G
eneral', 'Beverages–Non-Alcoholic', 'Insurance–Life', 'Drug Manufacturers–General', 'Ban
ks–Diversified', 'Banks–Regional', 'Banks–Regional', 'Software–Application', 'Software–A
pplication', 'REIT–Diversified', 'Drug Manufacturers–Specialty & Generic', 'Banks–Region
al', 'Software–Infrastructure', 'Banks–Diversified']
```

```python
txt = 'Banks-Diversified'

x = txt.isascii()

print(x)
```

```
True
```

```
## The Non-SCII character is:'–'
```

```python
for index, row in PROFILE.iterrows():
    for column in PROFILE.columns:
        if isinstance(row[column], str):
            #if '–' in value:
            new_value=row[column].replace('–','-')
            PROFILE.at[index, column]=new_value
```

```python
def check_dataframe_for_non_ascii(df):
    non_ascii = []
    for column in df.columns:
        for value in df[column]:
            #if has_non_ascii(str(value)):
            if (str(value).isascii()==False):
                non_ascii.append(value)
    return non_ascii


# Check if I succeed or not

contains_non_ascii = check_dataframe_for_non_ascii(PROFILE)
print(f"The DataFrame contains non-ASCII characters: {contains_non_ascii}")
```

```
The DataFrame contains non-ASCII characters: []
```

The third task is to impute the missing values. The original data is very clean overall. The only important missing values are situated in the file containing complementary information of the companies, as shown below.

```
PROFILE.isna().sum()
```

```
Name            0
Symbol          0
Sector          2
Industry        2
Num_Employees   7
dtype: int64
```

The missing values in the categorical columns 'Sector' and 'Industry' exist for indexes in which BRK has invested. We chose to simply fill in “Non” to replace the NaN. For the missing values in 'Num_Employee', we filled them with the mean of the column (round to integers).

```
PROFILE.isna().sum()
```

```
Name            0
Symbol          0
Sector          2
Industry        2
Num_Employees   7
dtype: int64
```

Feature Engineering

For the feature engineering, there is one major challenge: how to combine the stock price files of the 45 companies? Our first plan is to merge the 45 dataframes horizontally, which expanded the column number to 265, as shown below.

Date	High_AMZN	Low_AMZN	Open_AMZN	Close_AMZN	Volume_AMZN	Adj Close_AMZN	High_AXP	Low_AXP	Open_AXP	Close_AXP
2007-01-03	39.060001373291000	38.04999923706060	38.68000030517580	38.70000076293950	12405100	38.70000076293950	61.900001525878900	60.04999923706060	61.18000030517580	60.36000
2007-01-04	39.13999938964840	38.2599983215332	38.59000015258790	38.900001525878900	6318400	38.900001525878900	60.56999969482420	59.790000915527300	60.22999954223630	59.91999
2007-01-05	38.790000915527300	37.59999847412110	38.720001220703100	38.369998931884800	6619700	38.369998931884800	59.869998931884800	58.900001525878900	59.650001525878900	59.130001

This solution seems to work well until we began to create the ERD. Then, it became clear that the above data structure will cause significant problems in connecting different entities. Following the advice of Thomas, we pivoted the dataframe, creating a key column of "Symbol" as the first column. Along with the column "Date", they will serve as composite primary keys which facilitate the join operations in MySQL. The pivoted dataframe and the code to realize it are shown below.

Symbol	Date	High	Low	Open	Close	Volume	Adj Close
AMZN	2007-01-03	39.060001373291000	38.04999923706060	38.68000030517580	38.70000076293950	12405100.0	38.70000076293950
AMZN	2007-01-04	39.13999938964840	38.2599983215332	38.59000015258790	38.900001525878900	6318400.0	38.900001525878900
AMZN	2007-01-05	38.790000915527300	37.59999847412110	38.720001220703100	38.369998931884800	6619700.0	38.369998931884800
AMZN	2007-01-08	38.310001373291000	37.16999816894530	38.220001220703100	37.5	6783000.0	37.5
AMZN	2007-01-09	38.060001373291000	37.34000015258790	37.59999847412110	37.77999877929690	5703000.0	37.77999877929690

……

AMZN	2020-11-18	3140.0	3105.10009765625	3134.0	3105.4599609375	2916800.0	3105.4599609375
AMZN	2020-11-19	3125.0	3080.919921875	3105.31005859375	3117.02001953125	3010300.0	3117.02001953125
AMZN	2020-11-20	3132.889892578130	3098.050048828130	3117.02001953125	3099.39990234375	3374400.0	3099.39990234375
AXP	2007-01-03	61.900001525878900	60.04999923706060	61.18000030517580	60.36000061035160	6142500.0	48.15082931518560
AXP	2007-01-04	60.56999969482420	59.790000915527300	60.22999954223630	59.91999816894530	5671200.0	47.7998161315918
AXP	2007-01-05	59.869998931884800	58.900001525878900	59.650001525878900	59.130001068115200	6768100.0	47.16963958740230
AXP	2007-01-08	59.7599983215332	58.34999847412110	59.02999877929690	59.68999862670900	5000200.0	47.61634826660160

```python
company_abbre_lst =
['AMZN',
 'AXP',
 'AAPL',
 'AXTA',
 'BAC',
 'BK',
 'GOLD',
 'BIIB',
 'CHTR',
 'KO',
 'COST',
 'DVA',
 'GM',
 'GL',
 'JNJ',
 'JPM',
 'KHC',
 'KR',
 'LBTYA',
 'LBTYK',
 'LILA',
 'LILAK',
 'LSXMA',
 'LSXMK',
 'MTB',
 'MA',
 'MDLZ',
 'MCO',
 'PNC',
 'PG',
 'RH',
 'SIRI',
 'SNOW',
 'SPY',
 'STNE',
 'STOR',
 'SU',
 'SYF',
 'TEVA',
 'USB',
 'UPS',
 'VOO',
 'VRSN',
 'V',
 'WFC']
```

```python
company_catenated = pd.concat(company_df_lst, keys=company_abbre_lst)
```

```python
company_catenated
```

		Date	High	Low	Open	Close	Volume	Adj Close
AMZN	**0**	2007-01-03	39.060001	38.049999	38.680000	38.700001	12405100.0	38.700001
	1	2007-01-04	39.139999	38.259998	38.590000	38.900002	6318400.0	38.900002
	2	2007-01-05	38.790001	37.599998	38.720001	38.369999	6619700.0	38.369999
	3	2007-01-08	38.310001	37.169998	38.220001	37.500000	6783000.0	37.500000
	4	2007-01-09	38.060001	37.340000	37.599998	37.779999	5703000.0	37.779999
...	**...**	...	...	...	...	...	...	...
WFC	**3493**	2020-11-16	25.030001	24.389999	24.990000	24.900000	36954800.0	24.900000
	3494	2020-11-17	25.150000	24.520000	24.530001	25.040001	29490400.0	25.040001
	3495	2020-11-18	25.950001	25.059999	25.150000	25.059999	44318300.0	25.059999
	3496	2020-11-19	26.219999	24.969999	25.190001	26.160000	44560000.0	26.160000
	3497	2020-11-20	26.230000	25.459999	26.040001	25.480000	38425200.0	25.480000

129590 rows × 7 columns

This solution combined pivoting and concatenating, making the final data file more adaptable to the working logic of MySQL.

Exploratory Data Analysis and Visualization

We chose to do the EDA and visualization in Python. Both Matplotlib and Seaborn are applied. Let's start from discussing the pie plot which shows the top 10 holding companies in BRK's Portfolio (on October 18, 2020) as well as the corresponding code.

```
brk_company_lst2 = brk_company_lst.sort_values('Value', ascending=False)
```

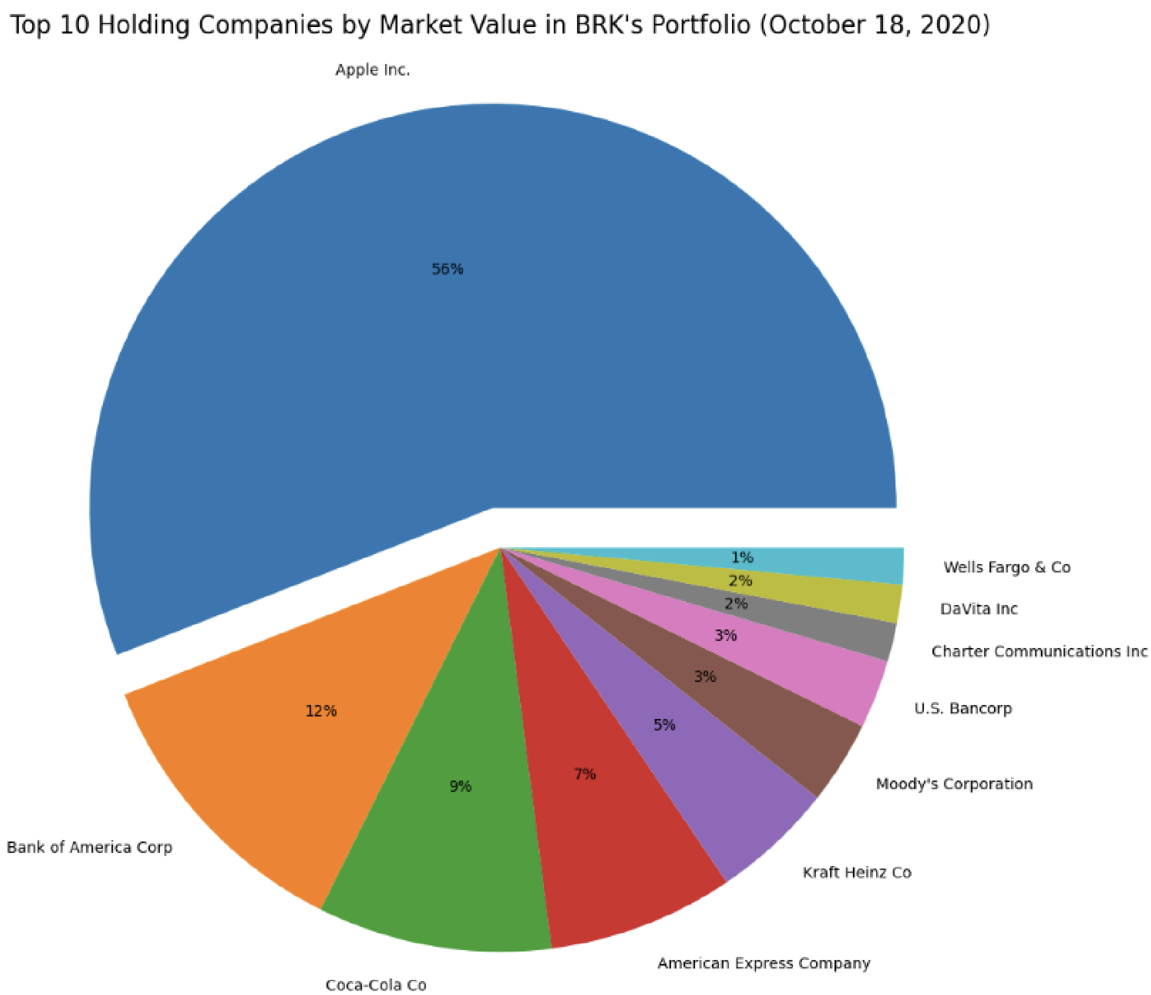
```
labels = list(brk_company_lst2['Name'][:10])
labels
```

```
['Apple Inc.',
 'Bank of America Corp',
 'Coca-Cola Co',
 'American Express Company',
 'Kraft Heinz Co',
 "Moody's Corporation",
 'U.S. Bancorp',
 'Charter Communications Inc',
 'DaVita Inc',
 'Wells Fargo & Co']
```

```
sizes = list(brk_company_lst2['Value'][:10])
sizes
```

```
[119432554741.0,
 25036332625.0,
 20012000000.0,
 15905478537.0,
 10472415747.0,
 7131046029.0,
 5854963364.0,
 3304917197.0,
 3276755845.0,
 3116751785.0]
```

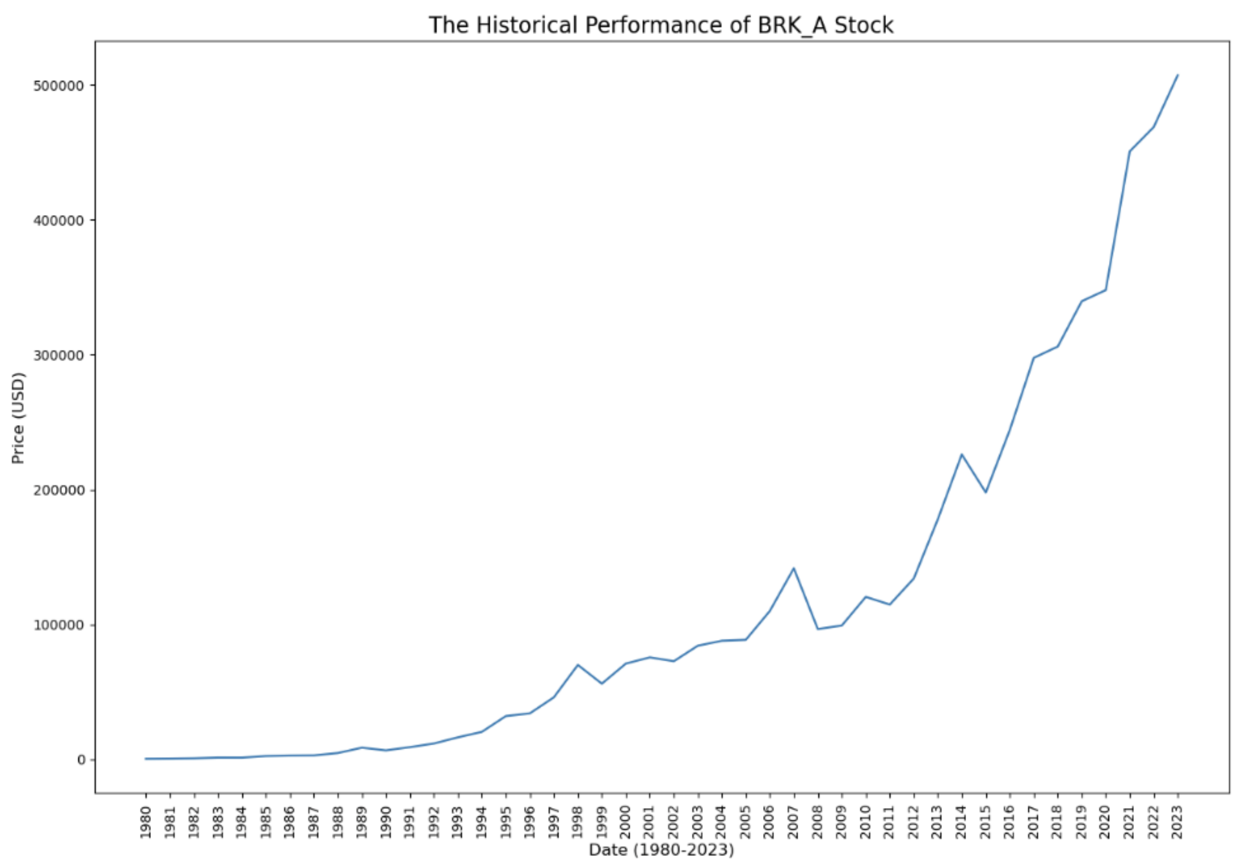
```
explode = (0.10,0,0,0,0,0,0,0,0,0)

fig, ax = plt.subplots(figsize=(16, 12))

ax.set_title('Title', fontsize=12)

plt.pie(sizes, labels=labels, explode=explode, autopct='%1.0f%%')

ax.set_title("Top 10 Holding Companies by Market Value in BRK's Portfolio (October 18, 2020)", fontsize=16)

plt.show()
```



About the technical aspect, we used “explode” to draw the reader’s attention to the importance of Apple. We can see that BRK invested extremely heavily on Apple, which alone occupied more than half of its total market value (as to the top 10 holdings). The boom of Apple’s stock price in the 2010s must have greatly benefited the accumulation of Buffet’s wealth.

The next comes the line plot showing the historical performance of BRK_A stock (1980-2023).

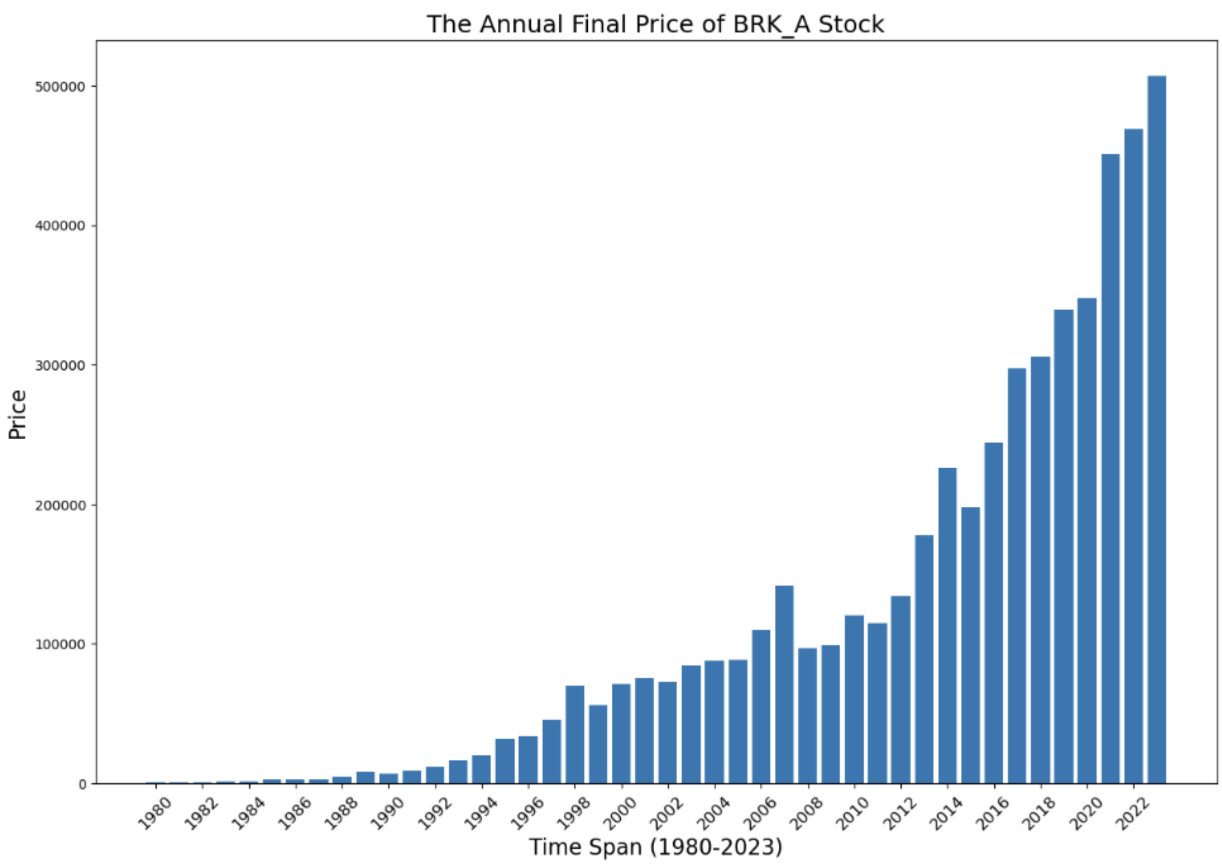
```
# Seaborn uses Matplotlib as its underlying plotting library, so you can modify the plot
# size using Matplotlib's functions.
fig, ax = plt.subplots(figsize=(15,10))

# Plot your data using Seaborn
sns.lineplot(x, y)

# Set the plot title
plt.title("The Historical Performance of BRK_A Stock", fontsize=16)

# Set the x-axis label
plt.xlabel("Date (1980-2023)", fontsize=12)
plt.xticks(rotation=90)
# Set the y-axis label
plt.ylabel("Price (USD)", fontsize=12)

# Show the plot
plt.show()
```



By doing “plt.xticks(rotation=90)”, we make the year labels much more visible. The stock price chart impresses us of the exceptional return of Buffet’s investment through his career from 1980. There are several significant setback at a yearly scale, especially during the financial crisis in 2008. In the last decade or so since 2012, the performance of BRK is particularly strong, which corresponded to the monetary easing cycle.

The third bar plot basically informs us about similar information. What needs to be highlighted is a technical improvement, using “[el*2 for el in range(len(labels[::2]))], labels[::2]” in plt.xticks, in order to show less labels (one for every two year). This is a very useful skill in improving the visibility of timeline plotting.

```
x = BRK_A_annual_value3['Date']
y = BRK_A_annual_value3['Close']

fig, ax = plt.subplots(figsize=(15,10))
plt.title("The Annual Final Price of BRK_A Stock", fontsize=18)
plt.xlabel("Time Span (1980-2023)", fontsize=16)
plt.ylabel("Price", fontsize=16)

#plt.xticks(range(len(labels)),labels, rotation=45, fontsize=12)
plt.xticks([el*2 for el in range(len(labels[::2]))],labels[::2], rotation=45, fontsize=12)

plt.bar(x,y)
plt.show()
```



The fourth is a box plot based on the calculation of the monthly standard deviation of BRK's return. The code below illustrates the process to get the monthly std. The main barrier to overcome is to find the "resample('M')" operator to divide the price series into monthly blocks.

```
BRK_A['Return'] = BRK_A['Close'].pct_change()
```

```
BRK_A_std = BRK_A.loc[1:]
```

```
BRK_A_std
```

	Symbol	Date	Open	High	Low	Close	Adj Close	Volume	Return
1	BRK	1980-03-18	290.0	290.0	290.0	290.0	290.0	0	0.000000
2	BRK	1980-03-19	290.0	310.0	290.0	290.0	290.0	20000	0.000000
3	BRK	1980-03-20	290.0	290.0	290.0	290.0	290.0	0	0.000000
4	BRK	1980-03-21	290.0	290.0	290.0	290.0	290.0	0	0.000000
5	BRK	1980-03-24	290.0	290.0	270.0	270.0	270.0	10000	-0.068966
...	...	...	...	...	...	...	...	...	...
10882	BRK	2023-05-15	492844.0	497580.0	489001.0	495900.0	495900.0	7300	0.009605
10883	BRK	2023-05-16	493583.0	499999.0	490756.0	498620.0	498620.0	6600	0.005485
10884	BRK	2023-05-17	498176.0	503000.0	494787.0	500600.0	500600.0	6000	0.003971
10885	BRK	2023-05-18	502450.0	505420.0	498125.0	504360.0	504360.0	5900	0.007511
10886	BRK	2023-05-19	505890.0	511335.0	503905.0	507161.0	507161.0	6400	0.005554

10886 rows × 9 columns

```
Monthly_Std = BRK_A_std['Close'].resample('M').std()
```

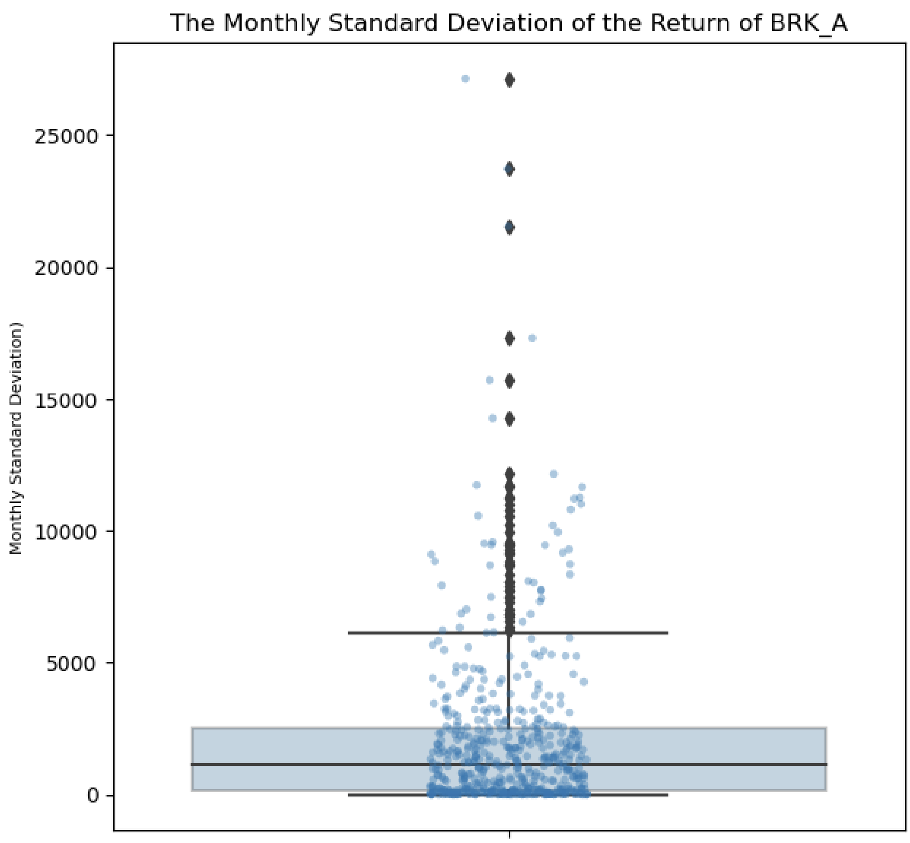
```
Monthly_Std = Monthly_Std.to_frame()
```

```
Monthly_Std.head()
```

Date	Close
1980-03-31	11.595018
1980-04-30	9.430195
1980-05-31	15.979898
1980-06-30	5.140873
1980-07-31	12.844811

Now we can introduce the box plot and the corresponding code. Each point in the box plot corresponds to a value of standard deviation.

```
f, ax = plt.subplots(figsize=(7,7))
ax = sns.boxplot(data=Monthly_Std, y='Close')
for patch in ax.patches:
    patch.set_alpha(0.3)
sns.stripplot(data=Monthly_Std, y='Close', alpha=0.4, size=4)

plt.title("The Monthly Standard Deviation of the Return of BRK_A", fontsize=12)
plt.ylabel("Monthly Standard Deviation)", fontsize=8)

plt.show()
```



We can see that the higher the Y axis goes, the fewer the points are. Within the box, more points are situated near its bottom than in its upper half. By further checking the dataframe of monthly standard deviation, we see that the std value becomes steadily larger over the course of roughly four decades. It is understandable since the absolute value of BRK stock kept growing and became a financial giant. If we wish to dig deeper into BRK's volatility difference in Buffet's earlier career and recent years, it may be necessary to introduce other indicators.

Data Base Type Selection

An essential question in undertaking data analytics is to choose a proper data base type. Between SQL and NoSQL database, the main differences can be included as follows:

1) Data Model: SQL databases follow a structured data model known as the relational model. Data is organized into tables with predefined schemas, and relationships between tables are established using keys. NoSQL databases employ various data models, including key-value, document, columnar, and graph models. They provide more flexibility in handling unstructured or semi-structured data.

2) Schema: SQL databases have a rigid, predefined schema that determines the structure and data types of the tables. Schema modifications often require altering table structures. NoSQL databases are schema-less or have flexible schemas. They allow dynamic schema changes, enabling the addition or removal of fields without strict adherence to a predefined structure.

3) Scalability: SQL databases typically follow a vertical scaling approach, where hardware resources (CPU, RAM) are increased to handle increased data and traffic. NoSQL databases excel at horizontal scalability. They can distribute data across multiple servers or clusters, providing better scalability and performance for large-scale applications.

4) Data Consistency: SQL databases prioritize data consistency and adhere to the ACID (Atomicity, Consistency, Isolation, Durability) properties. They ensure that data remains in a consistent state during transactions. NoSQL databases often prioritize other properties like availability and partition tolerance, following the CAP (Consistency, Availability, Partition Tolerance) theorem. They may sacrifice immediate consistency for increased scalability and fault tolerance.

5) Query Language: SQL databases use the SQL language for defining and manipulating data. SQL provides a standardized syntax for querying and managing

relational databases. NoSQL databases employ various query languages specific to their data models. Examples include MongoDB's query language for document databases and Cassandra Query Language (CQL) for columnar databases.

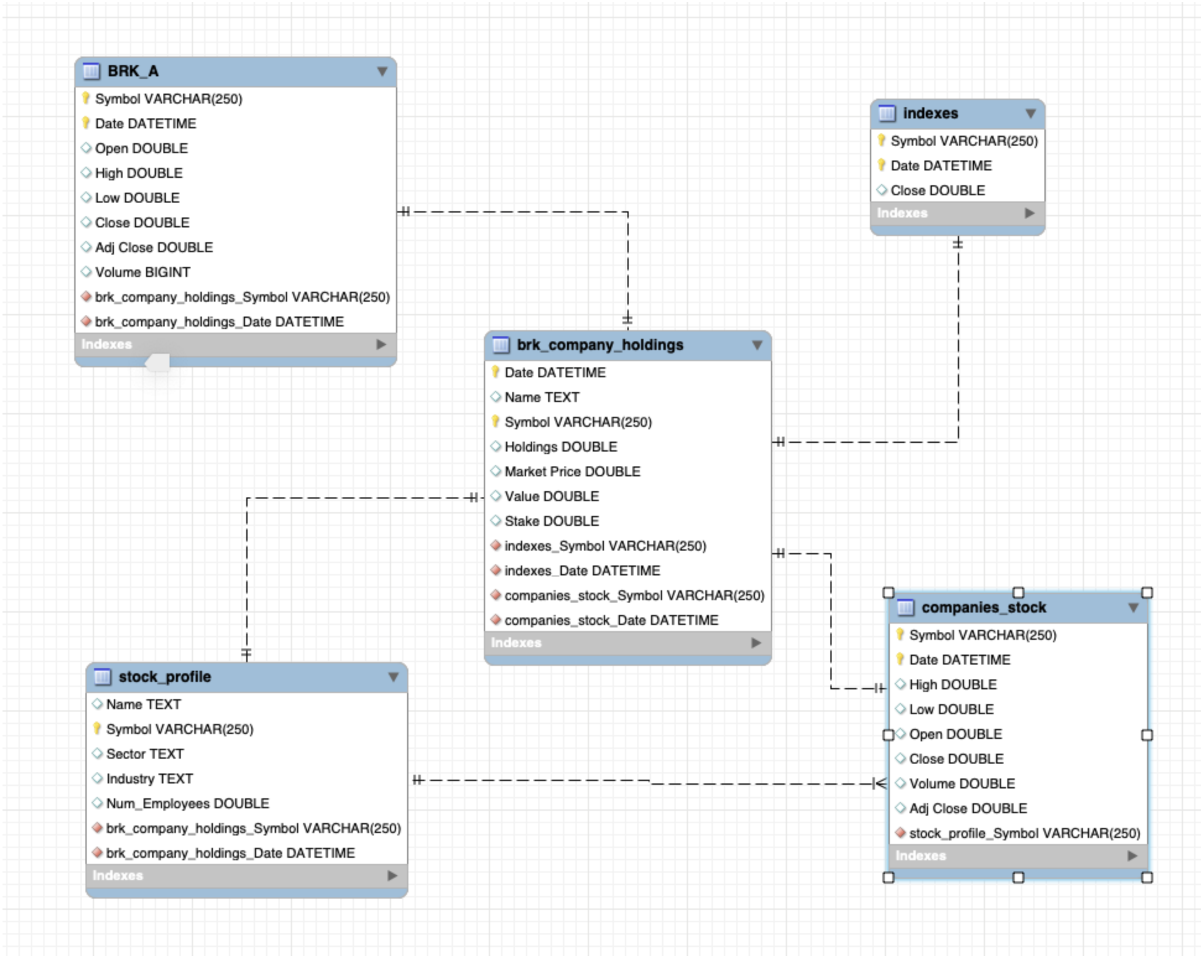
Given that our data consists of structured tables with a predefined schema, it would be appropriate to use SQL. It seems appropriate since I have structured tables with a predefined schema to use SQL. Furthermore, SQL would be optimal to add more data or other sources, which allow us to update the current dataset, bringing in the stock price information of BRK in the following years.

However, choosing SQL as our database type also has its disadvantages. The rigidity of the SQL schema prevents us from easily modifier the structure and data types of the tables. During our first exporting attempts, we encountered problems in connecting various entities. To solve the problem, we went back and forth between Jupyter Notebook and MySQL Workbench several rounds, which costed considerable time.

Entity Relationship Diagram (ERD)

To create an entity relationship diagram, we first the “sqlalchemy” library to import the cleaned data files into newly created database called “final_project” in MySQL. The code and the ERD are shown below.

```
### import pymysql.cursors
from sqlalchemy import create_engine, MetaData
from sqlalchemy.schema import CreateTable
from sqlalchemy import text

#prompt user to enter MySQL root password
import getpass

sql_pass = getpass.getpass()
#create connection tring and angine to connect to MySQL database
connection_string = 'mysql+pymysql://root:' + sql_pass + '@localhost:3306/final_project'
engine = create_engine(connection_string)

# export the tables to MySQL
brk_holdings_dated.to_sql('brk_company_holdings',engine, 'final_project', if_exists='replace', index=False)
```



As presented previously, the main difficulty in creating the ERD is to modify dataframes in Pandas in order to get the proper primary and foreign keys. Once this job is done, the ERD is ready to be produced. From the ERD above, we can see that "brk_company_holdings" is the central entity which connect all the other four entities together. Among them, "BRK_A" and "companies_stock" contain the most extensive data whereas "stock_profile" offers some categorical data which could be valuable complements. For all the five entities, there are two columns serving as keys, i.e. "Symbol" and "Date." For some of them, the two are combined into a composite key.

MySQL Queries

Our first query deals with the moving average, which is one of most useful indicators in finance. The screenshot below shows how we calculated three common moving averages, i.e. the 10-day MA, the monthly MA, and the quarterly MA. As we review through more rows, it becomes clear that the quarterly MA changes more smoothly, which makes it less sensible to temporary ups and downs but more revealing in terms of long-term tendencies.

```sql
-- Query1: Listing the three Moving Average Columns in the same table
SELECT Date, Close,
        AVG(Close) OVER (ORDER BY Date ROWS BETWEEN 9 PRECEDING AND CURRENT ROW) AS Moving_Average_10,
        AVG(Close) OVER (ORDER BY Date ROWS BETWEEN 23 PRECEDING AND CURRENT ROW) AS Moving_Average_monthly,
        AVG(Close) OVER (ORDER BY Date ROWS BETWEEN 71 PRECEDING AND CURRENT ROW) AS Moving_Average_quarterly
FROM final_project.BRK_A;
```

130% 20:45 1 error found

Result Grid Filter Rows: Search Export:

Date	Close	Moving_Average_10	Moving_Average_monthly	Moving_Average_quarterly
1980-03-17 00:00:00	290	290	290	290
1980-03-18 00:00:00	290	290	290	290
1980-03-19 00:00:00	290	290	290	290
1980-03-20 00:00:00	290	290	290	290
1980-03-21 00:00:00	290	290	290	290
1980-03-24 00:00:00	270	286.6666666666667	286.6666666666667	286.6666666666667
1980-03-25 00:00:00	270	284.2857142857143	284.2857142857143	284.2857142857143
1980-03-26 00:00:00	270	282.5	282.5	282.5
1980-03-27 00:00:00	270	281.1111111111111	281.1111111111111	281.1111111111111
1980-03-28 00:00:00	270	280	280	280
1980-03-31 00:00:00	260	277	278.1818181818182	278.1818181818182
1980-04-01 00:00:00	260	274	276.6666666666667	276.6666666666667

Our second query turns to the categorical data. Each rows correspond to one distinct industry of a sector. The last columns shows the average number of employees, which can be interesting to certain stakeholders.

```sql
-- Query2: Calculating Categorical Data
select * from final_project.stock_profile;
select count(distinct Sector) from final_project.stock_profile;
select count(distinct Industry) from final_project.stock_profile;
select Sector, Industry, avg(Num_Employees) from final_project.stock_profile
group by Sector, Industry
order by avg(Num_Employees) desc;
select Sector, count(distinct Industry) from final_project.stock_profile
group by Sector;
```

130% | 23:55 | 1 error found

Result Grid | Filter Rows: Search | Export:

Sector	Industry	avg(Num_Employee...
Consumer Cyclical	Internet Retail	1125300
Consumer Defensive	Grocery Stores	500000
Industrials	Integrated Freight & Logistics	260280
Financial Services	Banks-Diversified	247419.33333333334
Consumer Cyclical	Auto Manufacturers	164000
Consumer Defensive	Discount Stores	156000
Technology	Consumer Electronics	147000
Basic Materials	Gold	108792
None	None	108792
Consumer Defensive	Household & Personal Products	99000
Consumer Defensive	Beverages-Non-Alcoholic	86200
Consumer Defensive	Confectioners	80000

Result Grid
Form Editor
Field Types

Our third query studies the capitalization of the companies in which Buffet has invested. The companies are listed in a descending order, giving the larger companies more visibility.

```sql
-- Query3: Calculating the Total Capitalization of the Companies

select * from final_project.brk_company_holdings;
select Name, Symbol, Value/Stake as Total_Capitalization from final_project.brk_company_holdings
order by Total_Capitalization desc;
```

1:68 | 1 error found

Result Grid | Filter Rows: Search | Export:

Name	Symbol	Total_Capitalization
Apple Inc.	AAPL	2024280588830.5085
Amazon.com, Inc.	AMZN	1745336243000
Visa Inc	V	400017748000
JPMorgan Chase & Co.	JPM	322053917857.14746
Mastercard Inc	MA	309855637200
Coca-Cola Co	KO	215182795698.92474
Bank of America Corp	BAC	210389349789.916
Costco Wholesale Corporation	COST	165335131900
Charter Communications Inc	CHTR	132196687880
Wells Fargo & Co	WFC	94447023787.87878
American Express Company	AXP	84603609239.3617
Snowflake Inc	SNOW	67523917636.36364
U.S. Bancorp	USB	59141044080.808075
Moody's Corporation	MCO	54435465870.229004
Barrick Gold Corp	GOLD	48060715583.333336
General Motors Company	GM	48039989423.07693
PNC Financial Services Group Inc	PNC	45883332846.15385
Biogen Inc	BIIB	45013147500
Kraft Heinz Co	KHC	39369984011.27819
Bank of New York Mellon Corp	BK	33650806773.80952
Kroger Co	KR	26696374714.28581
Sirius XM Holdings Inc	SIRI	24333333333.333332
Verisign, Inc.	VRSN	23724903562.50002
StoneCo Ltd	STNE	17831624108.695652
Suncor Energy Inc.	SU	17547239769.23077
Synchrony Financial	SYF	17392960000
M&T Bank Corporation	MTB	12974753685.714285
DaVita Inc	DVA	11457188269.23077
Teva Pharmaceutical Industries Ltd	TEVA	10324288871.794872

Our fourth query gives us an overview of the total market value of Buffet's investment in every major sectors, also arranged in descending order.

```sql
-- Query4: Calculating the Total Capitalization of Each Sector

select count(Symbol) from final_project.brk_company_holdings;
select count(Symbol) from final_project.stock_profile;
select Sector, count(Sector) as Company_Number, sum(Value) as Total_Value
from final_project.stock_profile
inner join final_project.brk_company_holdings using (Symbol)
group by Sector
order by Total_Value desc;
```

Sector	Company_Number	Total_Value
Technology	4	124395524837
Financial Services	13	67845829041
Consumer Defensive	6	32964312824
Communication Services	8	6180102001
Consumer Cyclical	3	4901642177
Healthcare	4	3907899211
Basic Materials	2	1180167038
Real Estate	1	662627660
Energy	1	228114117
None	2	27399796
Industrials	1	10361736

Our fifth query did the most complex calculation in order to trace the monthly performance of BRK_A. We first created a temporary table (using the "where" clause) to filter and select only the rows of the last days of the months. Then we did subtraction between the end-of-month stock price and that of the previous month to get our target column, i.e. "monthly_gain". This can be a valuable reference to the investors, both current and potential, of BRK for their future financial decision.

```sql
-- Query5: Calculating the Monthly Performance of BRK_A

SELECT LAST_DAY(Date) AS LastDayOfMonth, Close
from final_project.BRK_A where (Date) in (LAST_DAY(Date));

# Create a Temporary Table istead of Using Subquery
create temporary table Monthly_Performance SELECT LAST_DAY(Date) AS Last_Day_Month, Close
from final_project.BRK_A where (Date) in (LAST_DAY(Date));

select * from final_project.Monthly_Performance;

SELECT
  Last_Day_Month,
  Close AS Stock_Price,
  Close - LAG(Close) OVER (ORDER BY Close) AS Monthly_Gain
FROM Monthly_Performance;
```

12:97 1 error found

Result Grid Filter Rows: Search Export:

Last_Day_Month	Stock_Price	Monthly_Gain
1980-03-31	260	NULL
1980-04-30	275	15
1980-06-30	305	30
1980-07-31	340	35
1980-09-30	385	45
1980-10-31	420	35
1980-12-31	425	5
1981-09-30	460	35
1981-03-31	480	20
1982-03-31	480	0
1981-07-31	485	5
1982-08-31	485	0

Conclusion

The exploratory data analysis and the SQL queries, as well as the experiences in manipulating the original dataset, allow us to get some essential information about the exceptional investment performance of Berkshire Hathaway and its founder Warren Buffett.

We see first that Buffet's investment gained steady success, outperforming not only most of other investors but also the major indexes such as SnP500 and Nasdaq100. The constant success, shown by the annual performance data, convinced us of the consistency and power of Buffet's long-term holding strategy. If we look closer, it is not difficult to discover several radical drops of BRK's stock value, especially during the 2008 financial crisis. These downturns can be detected and measured by indicators including the standard deviation.

Another important insight comes from the categorical data. We see that Buffet, on the one hand, has diversified his investment in most of the major sectors and industries, on the other, invested heavily on only a limited number of companies in each sector. Apple for the technology and the Bank of America for the finance are both remarkable examples. While the diversified investment guaranteed steady gains through economic and financial circles, the selection of specific companies demonstrated the profound business insight of Buffet.

The next step of the project involves developing a Machine Learning model that will utilize the various features and extensive data to predict the "future" performance of BRK (settled in various time point in the past). Both the historical data of BRK's stock price and external factors like the indexes will be exploited to produce prediction values, which will be compared with the actual values to estimate its accuracy.