Project 1

Financial Planning with APIs and Monte Carlo Simulations

Our Challenge:

- We are approached by a client who would like to invest $500,000 over the next 15 years in order to plan for her retirement. She is debating on whether to purchase a home in her local market (Seattle, Tacoma, Everett), stocks & bonds, or crypto. She would like us to analyze possible outcomes in order to maximize the return on her investment.

User Story & Acceptance Criteria:

- As an investor I want to invest $500k in either real estate, stocks and bonds, or crypto so that I can maximize on the return in 15 years.
- Given the analysis on real estate markets, stocks and bonds, and crypto, when the ROI is higher in one of the portfolio options then we would recommend the portfolio to the client.

Our Method

- We plan to analyze historical data of real estate by getting an annual appreciation rate for neighborhoods in Seattle, Tacoma, and Everett and then applying that to the initial investment. We will analyze crypto and stock/bonds with Monte Carlo simulations to propose different investment portfolios to the client in order to estimate possible outcomes.

Technologies:

- This project is written in **python**. The required libraries in order to use the application are:
- **import** os
- **import** requests
- **import** json
- **import** pandas **as** pd
- **from** dotenv **import** load_dotenv
- **import** alpaca_trade_api **as** tradeapi
- **from** MCForecastTools **import** MCSimulation
- **%matplotlib** inline
- **from** pathlib **import** Path
- **import** hvplot.pandas

Real Estate

```
# Using the read_csv function and Path module, create a DataFrame
# by importing the New Housing Data w MLS Area Codes.csv file from the Resources folder
sea_data_df = pd.read_csv(
    Path("Resources/New Housing Data w MLS Area Codes.csv"))

# Review the first and last five rows of the DataFrame
display(sea_data_df.head())
display(sea_data_df.tail())
```

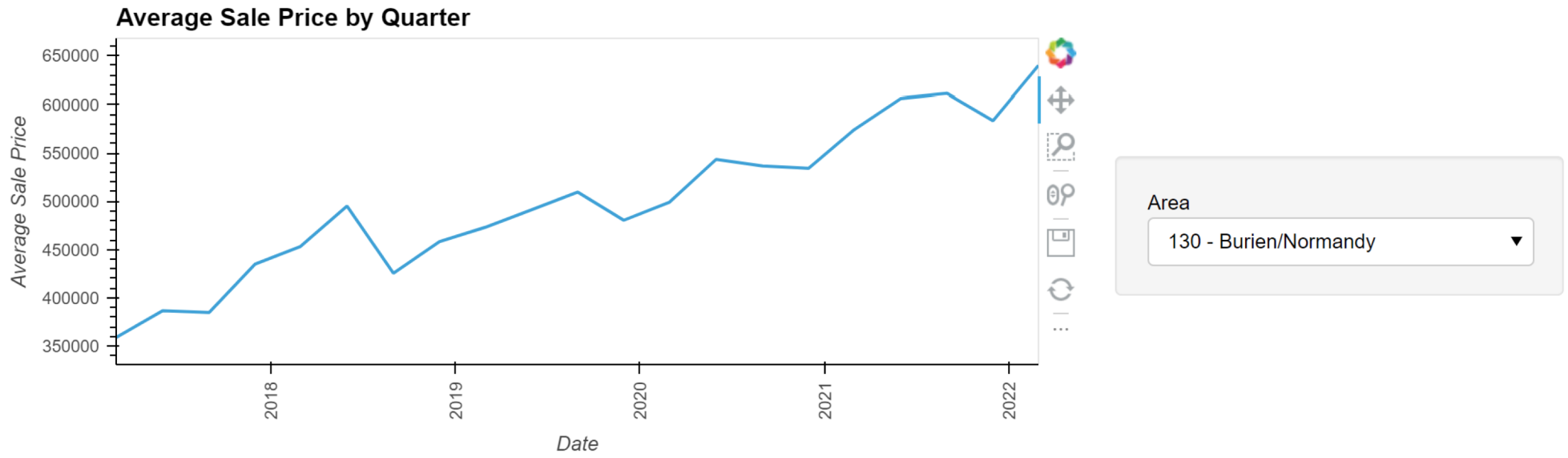
	Date	Area	Sales, $ Volume	Sales, Number of	Average Sale Price	Lat	Lon
0	2/28/17	130 - Burien/Normandy	5742587	16	358911.69	47.449212	-122.274963
1	2/28/17	140 - West Seattle	42521495	71	598894.30	47.515710	-122.371640
2	2/28/17	16 - North Tacoma	2533775	6	422295.83	47.277360	-122.496501
3	2/28/17	21 - North Tacoma	1970000	5	394000.00	47.268957	-122.464522
4	2/28/17	22 - North Tacoma	3023050	9	335894.44	47.262530	-122.470937

	Date	Area	Sales, $ Volume	Sales, Number of	Average Sale Price	Lat	Lon
624	2/28/22	705 - Ballard/Greenlak	149374743	145	1030170.64	47.686183	-122.329708
625	2/28/22	710 - North Seattle	69714429	56	1244900.52	47.675515	-122.306207
626	2/28/22	740 - Everett/Mukilteo	61250158	82	746953.15	47.870178	-122.307482
627	2/28/22	94 - Browns Point	8879999	13	683076.85	47.298577	-122.396309
628	2/28/22	99 - Spanaway	5267500	9	585277.78	47.055303	-122.364094

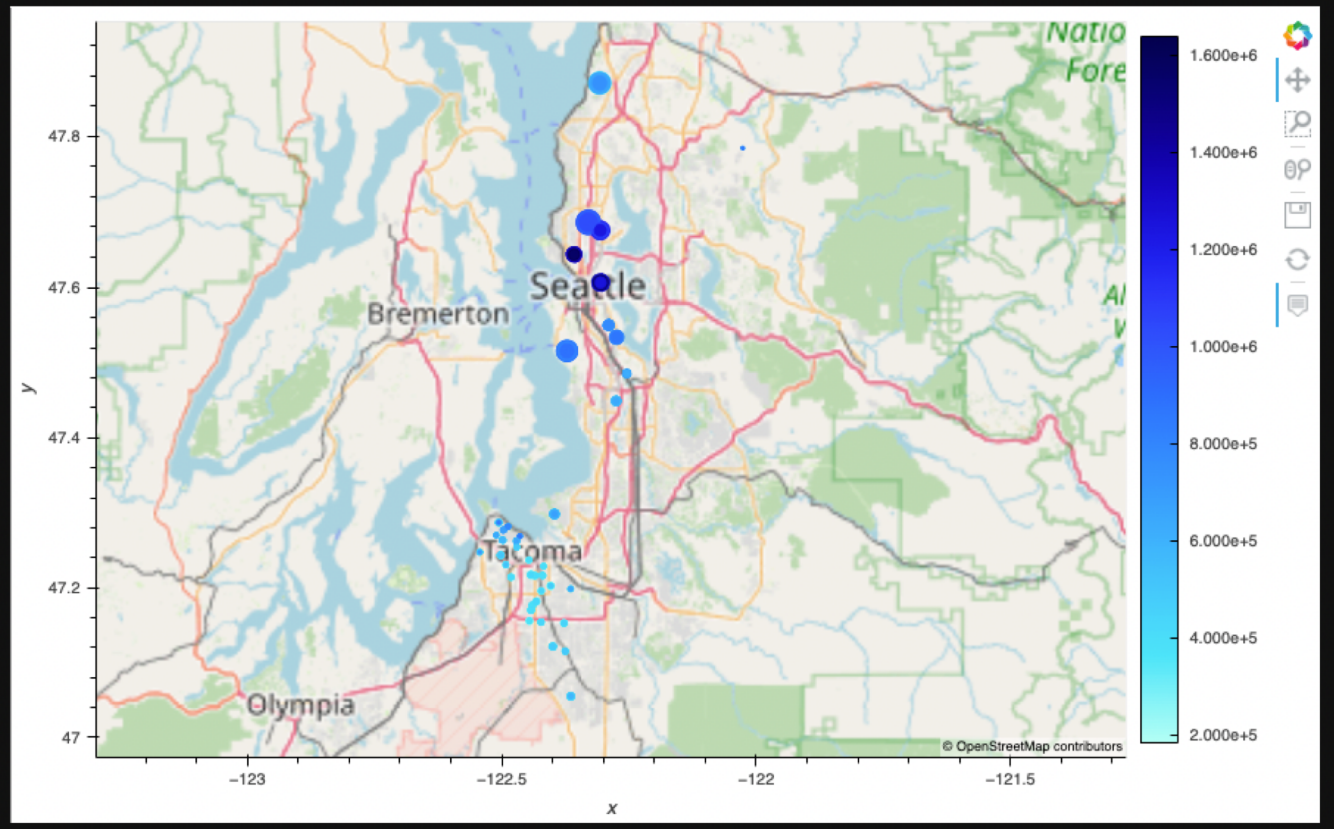
```
#drop NaN values
sea_data_df.dropna()
```

```
#Remove all '$' and ','

sea_data_df = sea_data_df.replace({'$':'', ',':''}, regex=True)
sea_data_df
```

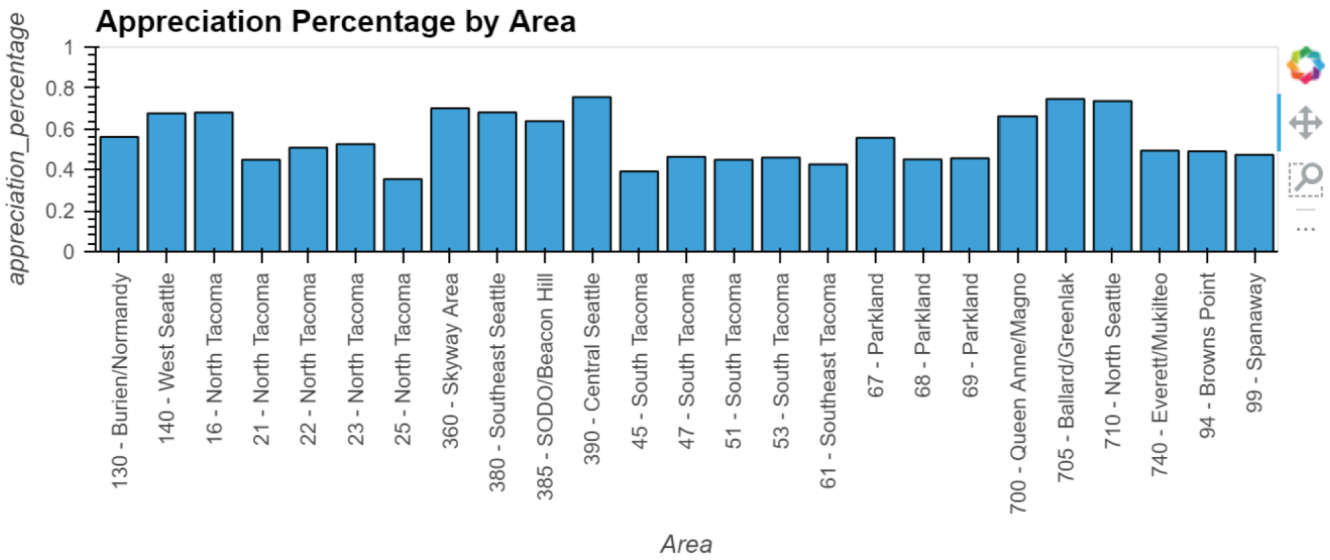
```
#Using hvplot, plot the average sale price by QTR. Use Date on the x-axis and use groupby Area.
sea_data_df.hvplot(
    x='Date',
    y='Average Sale Price',
    title='Average Sale Price by Quarter',
    groupby='Area',
    yformatter='%.0f'
).opts(xrotation=90)
```



```python
#Create a plot to analyze neighborhood info
all_neighborhoods_df.hvplot.points(
    'Lon',
    'Lat',
    geo=True,
    size='Sales, Number of',
    color='Average Sale Price',
    hover_cols='Area',
    tiles='OSM',
    frame_width=700,
    frame_height=500,
    yformatter='%.0f'
)
```



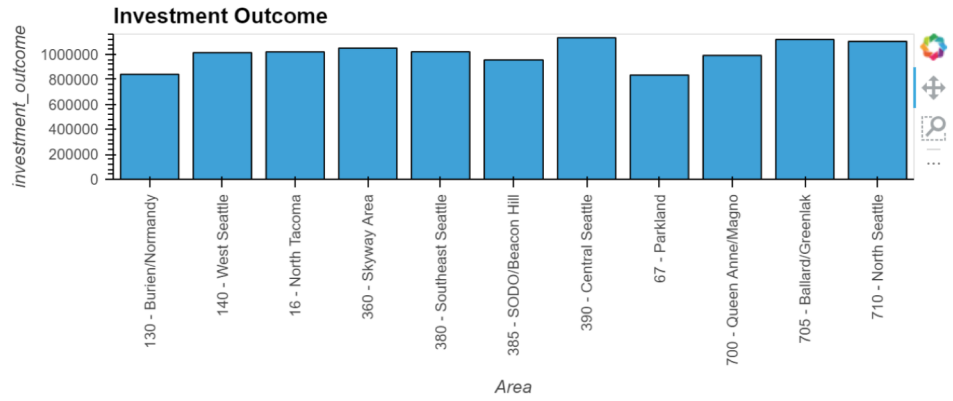
```
#with hvplots, plot beginning_end_avg_price dataframe into a bar plot with Area as the x-axis, appreciation_percentage as the y-axis. Title the plot.
beginning_end_avg_price.hvplot.bar(
    x='Area',
    y='appreciation_percentage',
    ylim=([0,1]),
    title="Appreciation Percentage by Area",
).opts(xrotation=90)
```



```
#create new column that calculates the investment outcome for the investment amount over the investment period per area.
greater_than_mean['investment_outcome'] = (greater_than_mean.annual_appreciation_percentage * investment_amount) * investment_period
#convert scientific notation
pd.options.display.float_format = '{:.2f}'.format
#display dataframe and sort values by investment outcome
greater_than_mean.sort_values('investment_outcome')
```

Area	Avg_Sale_Price_Q1_2017	Avg_Sale_Price_Q1_2022	appreciation_percentage	annual_appreciation_percentage	investment_outcome
67 - Parkland	274788.89	493700.00	0.56	0.11	834886.24
130 - Burien/Normandy	358911.69	639934.59	0.56	0.11	841285.26
385 - SODO/Beacon Hill	500109.09	784049.14	0.64	0.13	956781.40
700 - Queen Anne/Magno	1033912.44	1563506.88	0.66	0.13	991916.75
140 - West Seattle	598894.30	886026.73	0.68	0.14	1013898.81
16 - North Tacoma	422295.83	621000.00	0.68	0.14	1020038.24
380 - Southeast Seattle	590673.86	867838.89	0.68	0.14	1020939.26
360 - Skyway Area	433107.14	618142.86	0.70	0.14	1050987.97
710 - North Seattle	916333.63	1244900.52	0.74	0.15	1104104.64
705 - Ballard/Greenlak	768720.32	1030170.64	0.75	0.15	1119310.18
390 - Central Seattle	948653.61	1255577.88	0.76	0.15	1133327.08

```
#with hvplot, plot the investment outcome with area as the x-axis and investment_coutcome as the y-axis.
greater_than_mean.hvplot.bar(
    x='Area',
    y='investment_outcome',
    yformatter= '%.0f',
    title="Investment Outcome",
).opts(xrotation=90)
```



Stocks & Bonds

```
#function that accepts a variable string ticker for pulling the dataframe from alpaca
def puller(ticker):
    df1 = alpaca.get_bars(ticker,
                          timeframe,
                          start=start_date,
                          end=end_date,
                          limit=limit_rows).df
    return df1

#function for structuring dataframe so MCsimulation will accept as input
def structuredata(df, ticker):
    col_names = [(ticker, x) for x in df.columns]
    df.columns = pd.MultiIndex.from_tuples(col_names)
    return df

#function to merge two dfs into MCsimulation format
def merger(df1, df2):
        df_merged = pd.merge(df1, df2, how = "inner", left_index=True, right_index=True)
        return df_merged

#run function that accepts a list of strings and other defined other variables needed for structuring the combined data
def run(list, currentdf, ticker):
        if(list.index(ticker) == 0):
                df = puller(ticker)
                df = structuredata(df, ticker)
                #display(df.head())
                #print('hi')
                return df
        else:
                df = puller(ticker)
                df = structuredata(df, ticker)
                df = merger(currentdf, df)
                #display(df.head())
                #print('ho')
                return df

MCdf = pd.DataFrame()

for i in tickers:
        MCdf = run(tickers, MCdf, i)

display(MCdf.head())
```

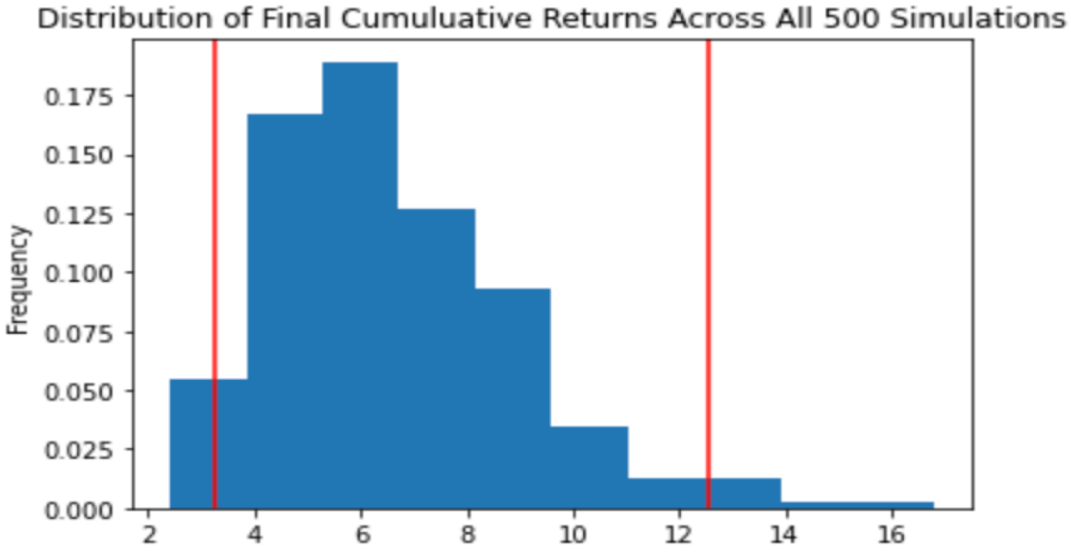
```
# Configure the Monte Carlo simulation to forecast 30 years cumulative returns
# The weights can be split in any fashion between tickers ['SPY', 'AGG', 'QQQ','VTV', 'IEMG', 'IWM']
# Run 500 samples.
MC_equal_weight = MCSimulation(
    portfolio_data = MCdf,
    weights = [.4, .2, .1, .1, .1, .1],
    num_simulation = 500,
    num_trading_days = 252*15
)

# Review the simulation input data
# Printing the first five rows of the simulation input data
MC_equal_weight.portfolio_data.head()
```

	SPY								AGG		...	IEMG				
	open	high	low	close	volume	trade_count	vwap	daily_return	open	high	...	vwap	daily_return	open	high	
timestamp																
2017-01-03 05:00:00+00:00	225.07	225.8300	223.8837	225.24	91366522	314573	224.642686	NaN	107.73	108.22	...	42.880855	NaN	136.52	136.8295	1
2017-01-04 05:00:00+00:00	225.64	226.7500	225.6100	226.58	78744433	259503	226.196593	0.005949	108.16	108.21	...	43.244844	0.008856	136.05	137.9600	1
2017-01-05 05:00:00+00:00	226.28	226.5800	225.4800	226.40	78379012	218284	226.251697	-0.000794	108.37	108.68	...	43.720961	0.011550	137.50	137.7600	1
2017-01-06 05:00:00+00:00	226.53	227.7500	225.9000	227.21	71559922	235983	227.113087	0.003578	108.43	108.50	...	43.561160	-0.004567	136.41	136.7100	1
2017-01-09 05:00:00+00:00	226.90	227.0701	226.4163	226.46	46939676	158755	226.731966	-0.003301	108.54	108.54	...	43.551103	-0.002294	135.54	135.7200	1

5 rows × 48 columns

```
#MC_equal_weight_plot = MC_equal_weight.plot_simulation()
MC_equal_weight_dist = MC_equal_weight.plot_distribution()
MC_summary_stats = MC_equal_weight.summarize_cumulative_return()
print(MC_summary_stats)
```

```
count            500.000000
mean               6.622267
std                2.319590
min                2.410756
25%                4.959231
50%                6.187289
75%                7.965954
max               16.786017
95% CI Lower       3.253924
95% CI Upper      12.540549
Name: 3780, dtype: float64
```



Crypto

```python
#function that creates url string using list input by user
def get_url(i):
    coinurl = f"https://api.polygon.io/v2/aggs/ticker/X:{i}USD/range/1/day/{startdate}/{enddate}?adjusted=true&sort=asc&lim
    return coinurl


#function that accepts a variable string URL for pulling the json response file from polygon
def polygonurl(coinurl):
        response = requests.get(coinurl).json()
        return response

#function for turning the json file into the dataframe with required columns
def jsontodf(json):
        df = pd.json_normalize(json, record_path =['results'],
        meta =['ticker']
        )
        raw_data = pd.DataFrame(columns= ["t", "o", "h", "l", "c", "v", "n", "vw", "ticker"])
        df = pd.concat([raw_data, df])
        return df

#function for structuring dataframe so MCsimulation will accept as input
def structuredata(df):
        df.columns = ['timestamp', 'open', 'high', 'low', 'close', 'volume', 'trade_count', 'vwap', 'symbol']
        #converts epoch time to a standard date format (polygon API used epoch time)
        df['timestamp'] = pd.to_datetime(df['timestamp'], unit='ms')
        df = df.set_index('timestamp')
        coin = f'{df["symbol"].iloc[0]}'
        col_names = [(coin, x) for x in df.columns]
        df.columns = pd.MultiIndex.from_tuples(col_names)
        return df


#function for saving an invidiual raw coin data as .csv, not currently being used
def savetocsv(df, i):
        filepath = Path(f'{i}.csv')
        df.to_csv(filepath)

#function to merge two dfs into MCsimulation format
def merger(df1, df2):
        df_merged = pd.merge(df1, df2, how = "inner", left_index=True, right_index=True)
        return df_merged
```

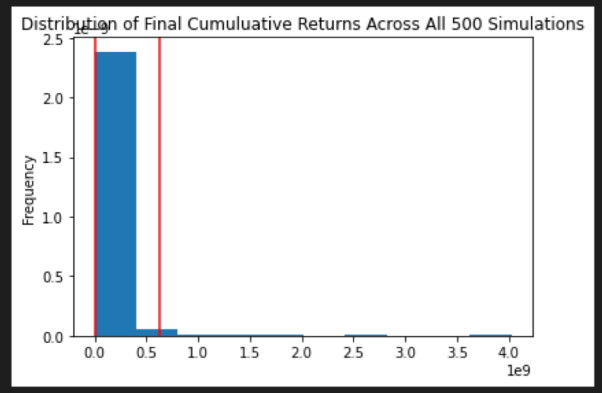
```python
#run function that accepts a list of urls defined other variables needed for structuring the combined data
def run(list, currentdf, i):
        if(list.index(i) == 0):
                coinurl = get_url(i)
                response = polygonurl(coinurl)
                df = jsontodf(response)
                #savetocsv(df, i)
                df = structuredata(df)
                #display(df.head())
                #print('hi')
                return df
        else:
                coinurl = get_url(i)
                response = polygonurl(coinurl)
                df = jsontodf(response)
                #savetocsv(df, i)
                df = structuredata(df)
                df = merger(currentdf, df)
                #display(df.head())
                #print('ho')
                return df
```

```python
#cell for executing the data compiling functions

MCdf = pd.DataFrame()

#looping through the urls to build MCsimulation dataframe input
for i in coinlist:
        MCdf = run(coinlist, MCdf, i)

display(MCdf)
```

```
#MC_equal_weight_plot = MC_6040_weight.plot_simulation()
MC_equal_weight_dist = MC_equal_weight.plot_distribution()
MC_summary_stats = MC_equal_weight.summarize_cumulative_return()
print(MC_summary_stats)
```

[8] ✓ 3.2s

```
count           5.000000e+02
mean            7.569390e+07
std             2.906096e+08
min             2.575435e+04
25%             2.228695e+06
50%             9.428284e+06
75%             4.111532e+07
max             4.030935e+09
95% CI Lower    1.527448e+05
95% CI Upper    6.258366e+08
Name: 5475, dtype: float64
```




```
# Using the lower and upper `95%` confidence intervals from the summary statistics,
# calculate the range of the probable cumulative returns for a $500000 investment
ci_95_lower_cumulative_return = MC_summary_stats[8] * 500000
ci_95_upper_cumulative_return = MC_summary_stats[9] * 500000

print(f"There is a 95% chance that an initial investment of $500,000 in the portfolio"
    f" over the next 15 years will end within in the range of"
    f" ${ci_95_lower_cumulative_return: .2f} and ${ci_95_upper_cumulative_return: .2f}.")
```

[9] ✓ 0.8s

```
There is a 95% chance that an initial investment of $500,000 in the portfolio over the next 15 years will end within in the range of $ 76372399044.75 and $ 312918308820224.25.
```

Our Analysis:

- Real estate: Based on historical data analysis, Central Seattle provided the greatest amount of annual appreciation at just over 15%. If the investor chooses to invest their $500,000 into the real estate market in Central Seattle, we estimate their investment would be worth approximately $1,633,327 at the end of a 15 year ownership period.
- Stocks and bonds: There is a 95% chance that an initial investment of $500,000 in the portfolio over the next 15 years will end within in the range of $1,626,961.92 and $6,270,274.26.
- Crypto: There is a 95% chance that an initial investment of $500,000 in the portfolio over the next 15 years will end within in the range of $53,063,020,989,972,864.00 and $754,312,303,589,329,600,512.00.

Our Analysis Disclaimer:

Although this result presents potentially fantastic news, it's important to note that these forecasted return values are based on only three years of historical price data. The forecast simulates more variability than the data that the simulation is based on includes. In general, it's ideal to supply one year of historical data for each year of simulated data. If we simulate using only small amounts of data during a recent time when markets are booming, or instead falling precipitously, a Monte-Carlo Analysis will inadvertently extrapolate this temporary market movement too far into the future. Getting data over a longer time period mitigates this effect. Due to the limitations of the Alpaca API, however, we can typically produce just three full years of historical data.

Portfolio Recommendation:

- Given the provided data, we would recommend the client invest her $500,000.00 in the stocks and bonds portfolio that we analyzed.

Executive Summary

- With the programming provided, it makes it easy for financial advisors to utilize this application to come up with the best options for clients who want to invest in their retirement via different portfolio options.
- The application provides risk profiles of conservative, moderate, and high risk tolerance.

Potential Next Steps:

- If we had more time to work on this project, we would create a CLI or web based app. This way, people can utilize investing opportunities while cutting out the financial advisor.

Contributors:

- Cody Schroeder, codeman@uw.edu
- Hilary Willis, hilarywillis@gmail.com
- Theo Prentice, theoprentice14@gmail.com
- Aaron Bumgarner, aaron.j.bumgarner@gmail.com
- Aranda Furth, arandafurth@gmail.com

Questions?


© 2012 Ted Goff
"Our data analysis experts can't read your minds. You're going to reach your own decision regarding hiring us, even though it's the same decision we knew in advance you'd make."