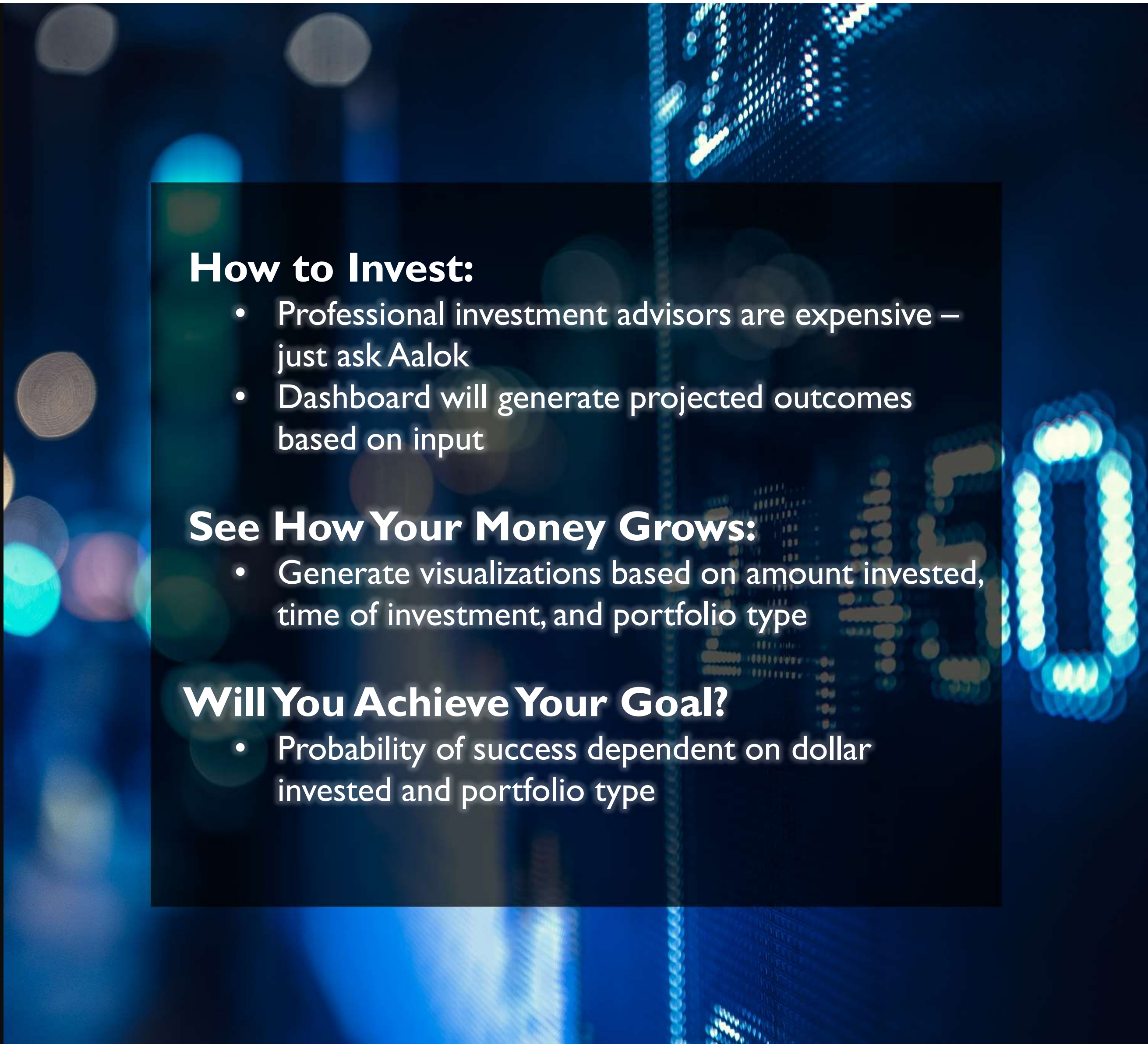
INVESTMENT DASHBOARD

How Will Your Money Grow?

Aalok Devkota
Gurcharan Singh
Heather Marshall
Stephanie Lin

PROJECT PURPOSE



How to Invest:

- Professional investment advisors are expensive – just ask Aalok
- Dashboard will generate projected outcomes based on input

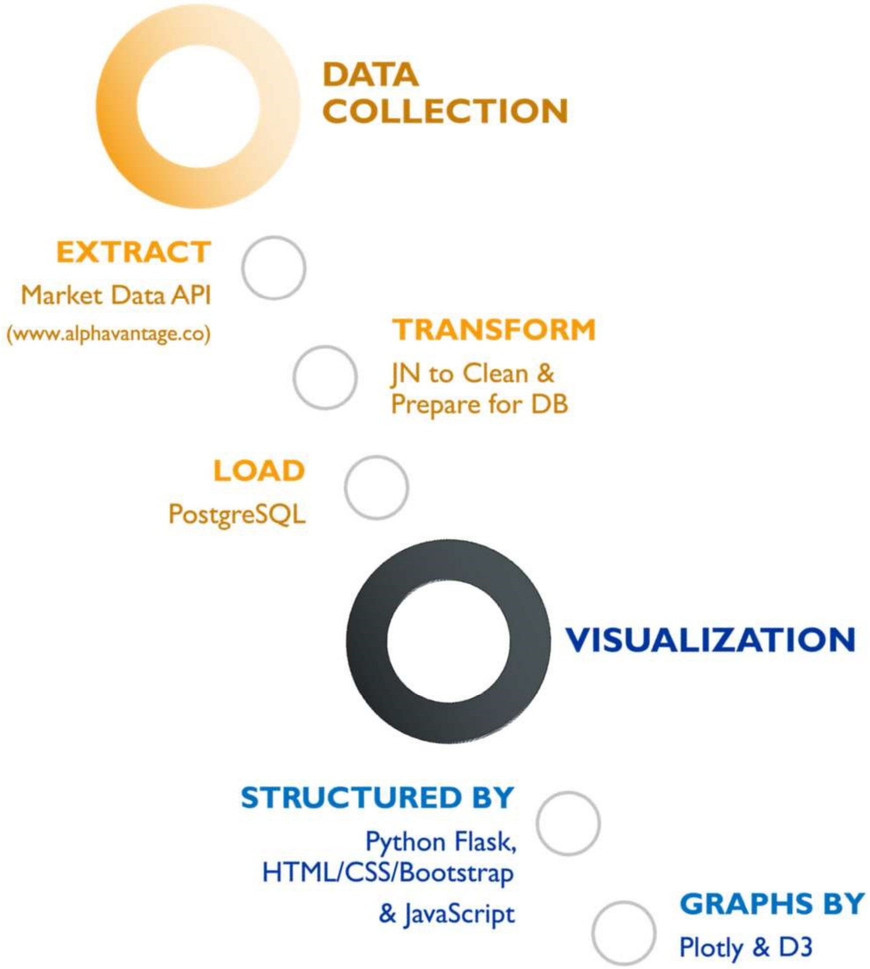
See How Your Money Grows:

- Generate visualizations based on amount invested, time of investment, and portfolio type

Will You Achieve Your Goal?

- Probability of success dependent on dollar invested and portfolio type

WORK PROCESS



DATA COLLECTION

EXTRACT
Market Data API
(www.alphavantage.co)

TRANSFORM
JN to Clean &
Prepare for DB

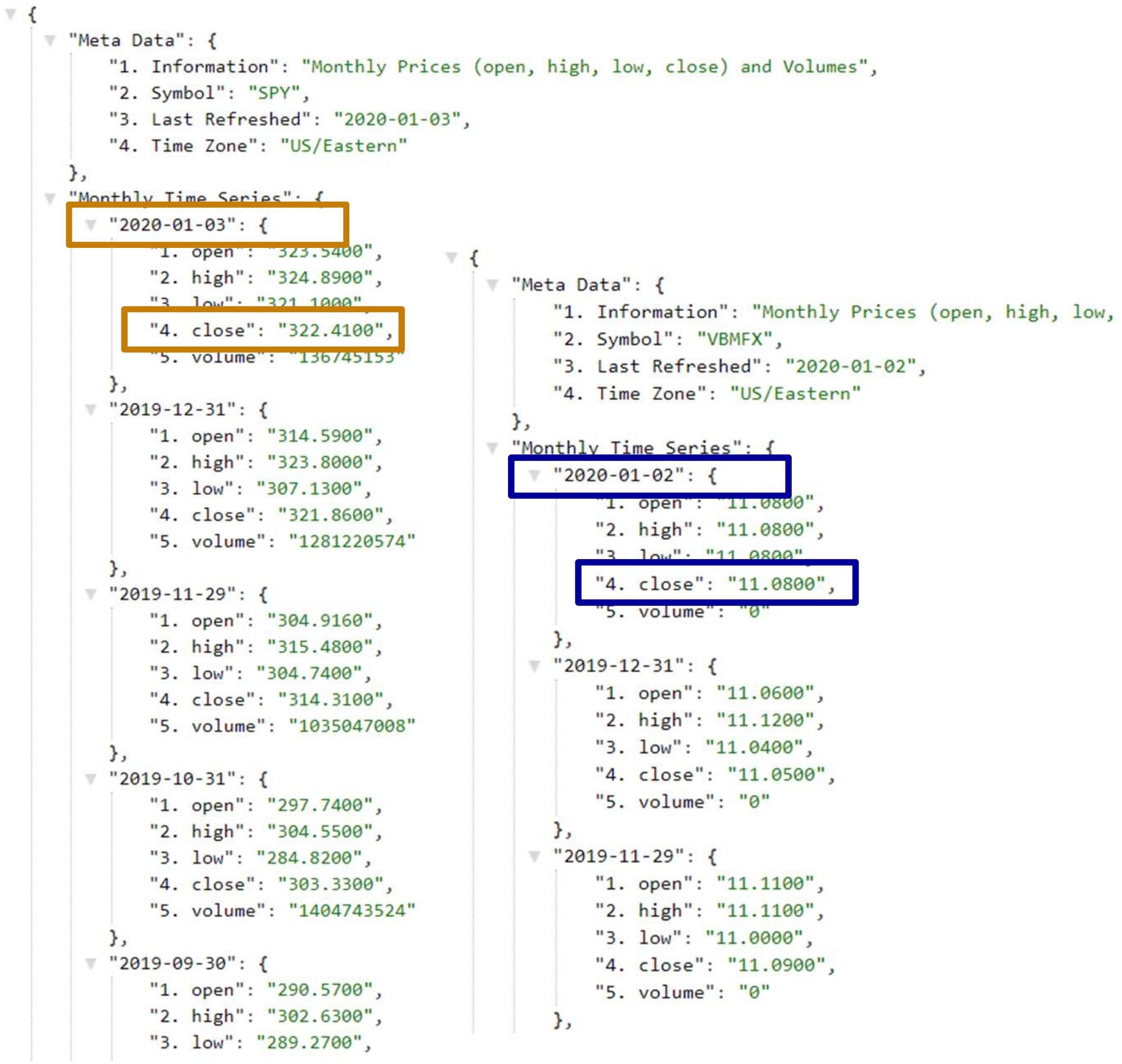
LOAD
PostgreSQL

VISUALIZATION

STRUCTURED BY
Python Flask,
HTML/CSS/Bootstrap
& JavaScript

GRAPHS BY
Plotly & D3

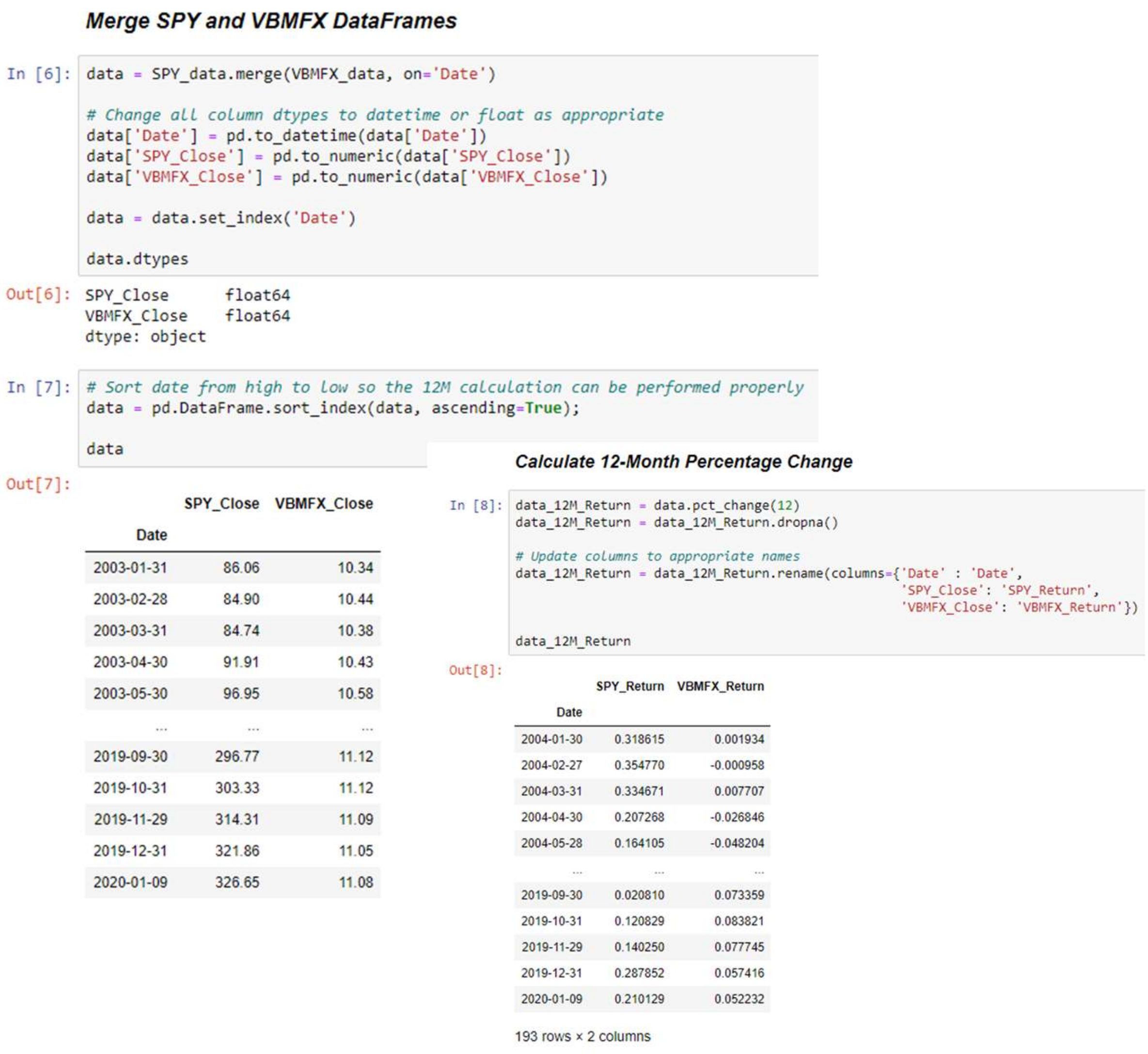
META DATA



```
{
  "Meta Data": {
    "1. Information": "Monthly Prices (open, high, low, close) and Volumes",
    "2. Symbol": "SPY",
    "3. Last Refreshed": "2020-01-03",
    "4. Time Zone": "US/Eastern"
  },
  "Monthly Time Series": {
    "2020-01-03": {
      "1. open": "323.5400",
      "2. high": "324.8900",
      "3. low": "321.1000",
      "4. close": "322.4100",
      "5. volume": "136745153"
    },
    "2019-12-31": {
      "1. open": "314.5900",
      "2. high": "323.8000",
      "3. low": "307.1300",
      "4. close": "321.8600",
      "5. volume": "1281220574"
    },
    "2019-11-29": {
      "1. open": "304.9160",
      "2. high": "315.4800",
      "3. low": "304.7400",
      "4. close": "314.3100",
      "5. volume": "1035047008"
    },
    "2019-10-31": {
      "1. open": "297.7400",
      "2. high": "304.5500",
      "3. low": "284.8200",
      "4. close": "303.3300",
      "5. volume": "1404743524"
    },
    "2019-09-30": {
      "1. open": "290.5700",
      "2. high": "302.6300",
      "3. low": "289.2700",
```

```
{
  "Meta Data": {
    "1. Information": "Monthly Prices (open, high, low,
    "2. Symbol": "VBMFX",
    "3. Last Refreshed": "2020-01-02",
    "4. Time Zone": "US/Eastern"
  },
  "Monthly Time Series": {
    "2020-01-02": {
      "1. open": "11.0800",
      "2. high": "11.0800",
      "3. low": "11.0800",
      "4. close": "11.0800",
      "5. volume": "0"
    },
    "2019-12-31": {
      "1. open": "11.0600",
      "2. high": "11.1200",
      "3. low": "11.0400",
      "4. close": "11.0500",
      "5. volume": "0"
    },
    "2019-11-29": {
      "1. open": "11.1100",
      "2. high": "11.1100",
      "3. low": "11.0000",
      "4. close": "11.0900",
      "5. volume": "0"
    },
```

EXTRACT



Merge SPY and VBMFX DataFrames

In [6]:
```
data = SPY_data.merge(VBMFX_data, on='Date')

# Change all column dtypes to datetime or float as appropriate
data['Date'] = pd.to_datetime(data['Date'])
data['SPY_Close'] = pd.to_numeric(data['SPY_Close'])
data['VBMFX_Close'] = pd.to_numeric(data['VBMFX_Close'])

data = data.set_index('Date')

data.dtypes
```

Out[6]:
```
SPY_Close      float64
VBMFX_Close    float64
dtype: object
```

In [7]:
```
# Sort date from high to low so the 12M calculation can be performed properly
data = pd.DataFrame.sort_index(data, ascending=True);

data
```

Out[7]:

	SPY_Close	VBMFX_Close
Date		
2003-01-31	86.06	10.34
2003-02-28	84.90	10.44
2003-03-31	84.74	10.38
2003-04-30	91.91	10.43
2003-05-30	96.95	10.58
...	...	...
2019-09-30	296.77	11.12
2019-10-31	303.33	11.12
2019-11-29	314.31	11.09
2019-12-31	321.86	11.05
2020-01-09	326.65	11.08

Calculate 12-Month Percentage Change

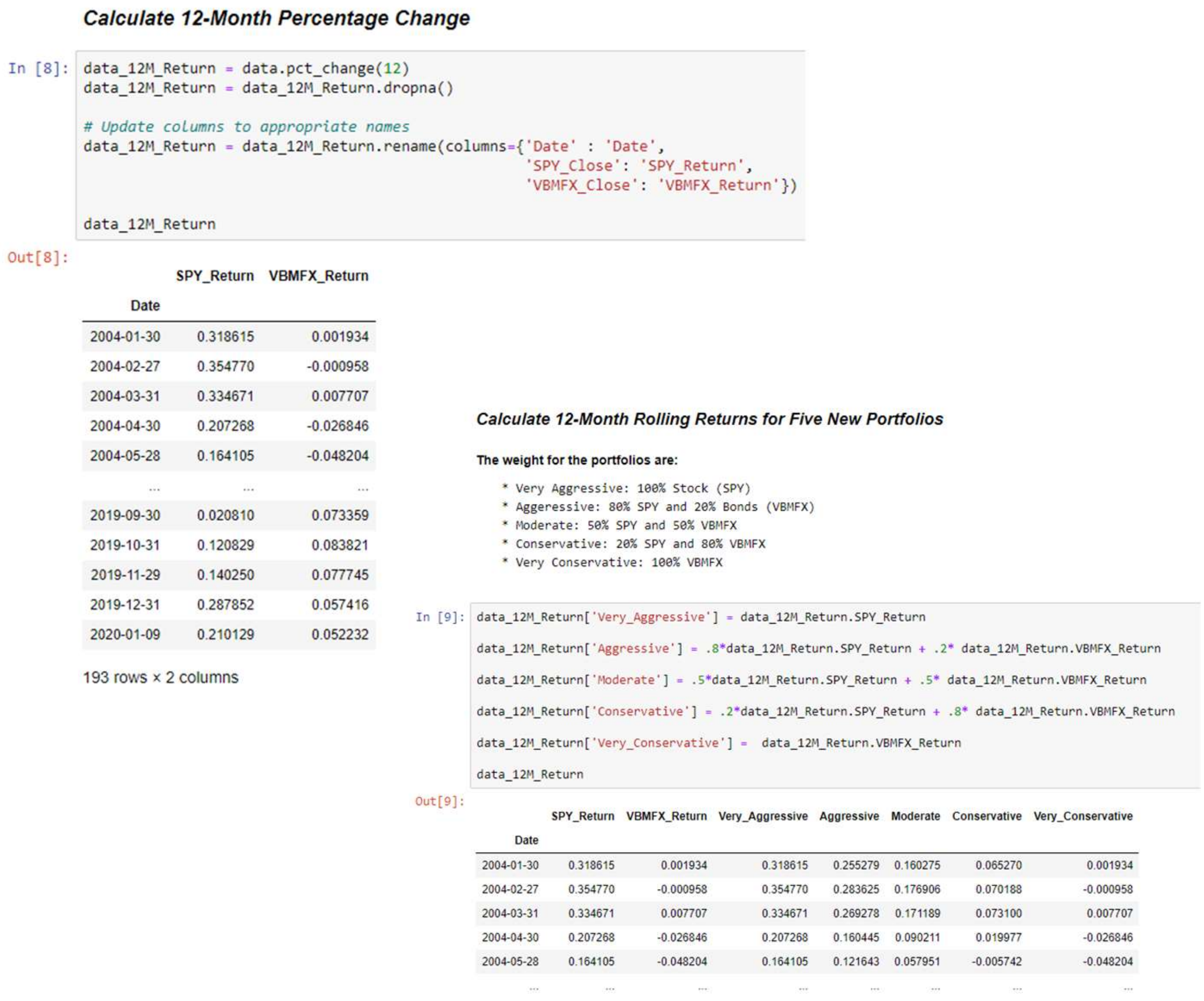
In [8]:
```
data_12M_Return = data.pct_change(12)
data_12M_Return = data_12M_Return.dropna()

# Update columns to appropriate names
data_12M_Return = data_12M_Return.rename(columns={'Date' : 'Date',
                                                  'SPY_Close': 'SPY_Return',
                                                  'VBMFX_Close': 'VBMFX_Return'})

data_12M_Return
```

Out[8]:

	SPY_Return	VBMFX_Return
Date		
2004-01-30	0.318615	0.001934
2004-02-27	0.354770	-0.000958
2004-03-31	0.334671	0.007707
2004-04-30	0.207268	-0.026846
2004-05-28	0.164105	-0.048204
...	...	...
2019-09-30	0.020810	0.073359
2019-10-31	0.120829	0.083821
2019-11-29	0.140250	0.077745
2019-12-31	0.287852	0.057416
2020-01-09	0.210129	0.052232

193 rows × 2 columns

TRANSFORM



Calculate 12-Month Percentage Change

```
In [8]: data_12M_Return = data.pct_change(12)
        data_12M_Return = data_12M_Return.dropna()

        # Update columns to appropriate names
        data_12M_Return = data_12M_Return.rename(columns={'Date' : 'Date',
                                                          'SPY_Close': 'SPY_Return',
                                                          'VBMFX_Close': 'VBMFX_Return'})

        data_12M_Return
```

Out[8]:

Date	SPY_Return	VBMFX_Return
2004-01-30	0.318615	0.001934
2004-02-27	0.354770	-0.000958
2004-03-31	0.334671	0.007707
2004-04-30	0.207268	-0.026846
2004-05-28	0.164105	-0.048204
...	...	...
2019-09-30	0.020810	0.073359
2019-10-31	0.120829	0.083821
2019-11-29	0.140250	0.077745
2019-12-31	0.287852	0.057416
2020-01-09	0.210129	0.052232

193 rows × 2 columns

Calculate 12-Month Rolling Returns for Five New Portfolios

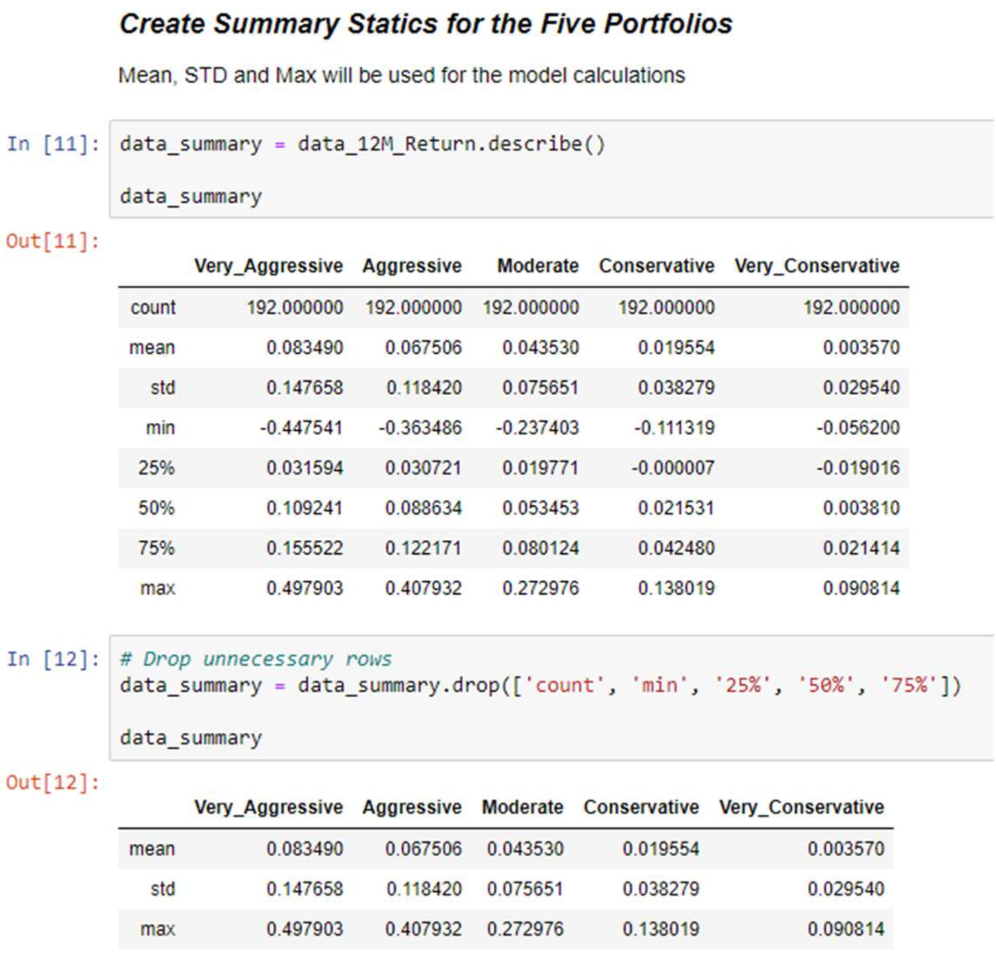
The weight for the portfolios are:

```
  * Very Aggressive: 100% Stock (SPY)
  * Aggeressive: 80% SPY and 20% Bonds (VBMFX)
  * Moderate: 50% SPY and 50% VBMFX
  * Conservative: 20% SPY and 80% VBMFX
  * Very Conservative: 100% VBMFX
```

```
In [9]: data_12M_Return['Very_Aggressive'] = data_12M_Return.SPY_Return

        data_12M_Return['Aggressive'] = .8*data_12M_Return.SPY_Return + .2* data_12M_Return.VBMFX_Return

        data_12M_Return['Moderate'] = .5*data_12M_Return.SPY_Return + .5* data_12M_Return.VBMFX_Return

        data_12M_Return['Conservative'] = .2*data_12M_Return.SPY_Return + .8* data_12M_Return.VBMFX_Return

        data_12M_Return['Very_Conservative'] =  data_12M_Return.VBMFX_Return

        data_12M_Return
```

Out[9]:

Date	SPY_Return	VBMFX_Return	Very_Aggressive	Aggressive	Moderate	Conservative	Very_Conservative
2004-01-30	0.318615	0.001934	0.318615	0.255279	0.160275	0.065270	0.001934
2004-02-27	0.354770	-0.000958	0.354770	0.283625	0.176906	0.070188	-0.000958
2004-03-31	0.334671	0.007707	0.334671	0.269278	0.171189	0.073100	0.007707
2004-04-30	0.207268	-0.026846	0.207268	0.160445	0.090211	0.019977	-0.026846
2004-05-28	0.164105	-0.048204	0.164105	0.121643	0.057951	-0.005742	-0.048204
...	...	...	...	...	...	...	...



Create Summary Statics for the Five Portfolios

Mean, STD and Max will be used for the model calculations

In [11]:
```
data_summary = data_12M_Return.describe()

data_summary
```

Out[11]:

	Very_Aggressive	Aggressive	Moderate	Conservative	Very_Conservative
count	192.000000	192.000000	192.000000	192.000000	192.000000
mean	0.083490	0.067506	0.043530	0.019554	0.003570
std	0.147658	0.118420	0.075651	0.038279	0.029540
min	-0.447541	-0.363486	-0.237403	-0.111319	-0.056200
25%	0.031594	0.030721	0.019771	-0.000007	-0.019016
50%	0.109241	0.088634	0.053453	0.021531	0.003810
75%	0.155522	0.122171	0.080124	0.042480	0.021414
max	0.497903	0.407932	0.272976	0.138019	0.090814

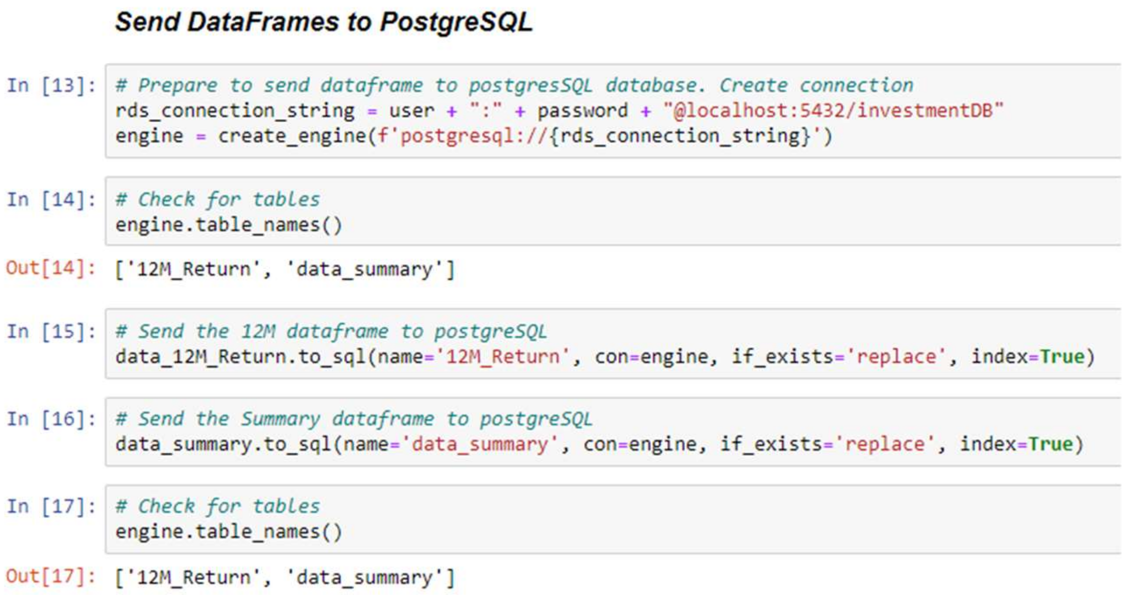
In [12]:
```
# Drop unnecessary rows
data_summary = data_summary.drop(['count', 'min', '25%', '50%', '75%'])

data_summary
```

Out[12]:

	Very_Aggressive	Aggressive	Moderate	Conservative	Very_Conservative
mean	0.083490	0.067506	0.043530	0.019554	0.003570
std	0.147658	0.118420	0.075651	0.038279	0.029540
max	0.497903	0.407932	0.272976	0.138019	0.090814

TRANSFORM

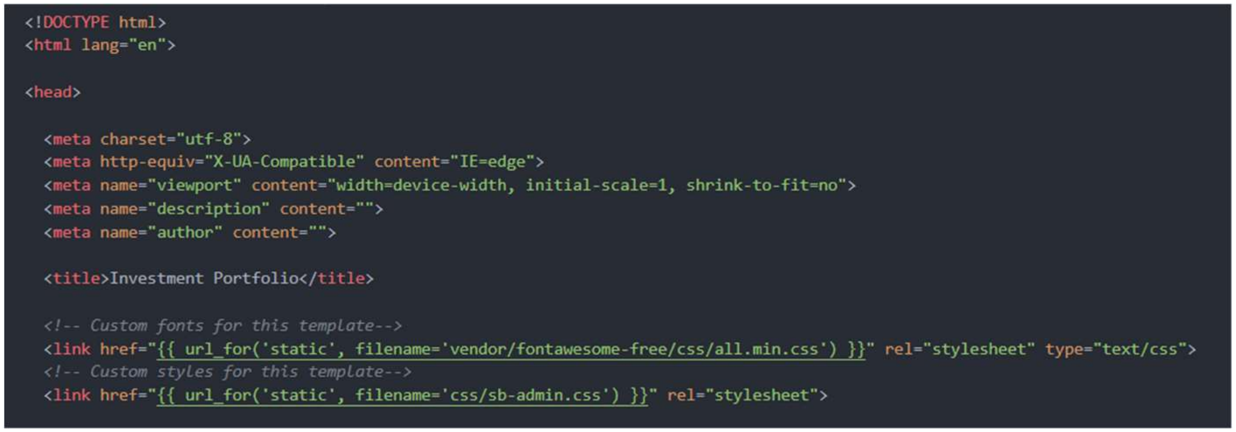
LOAD



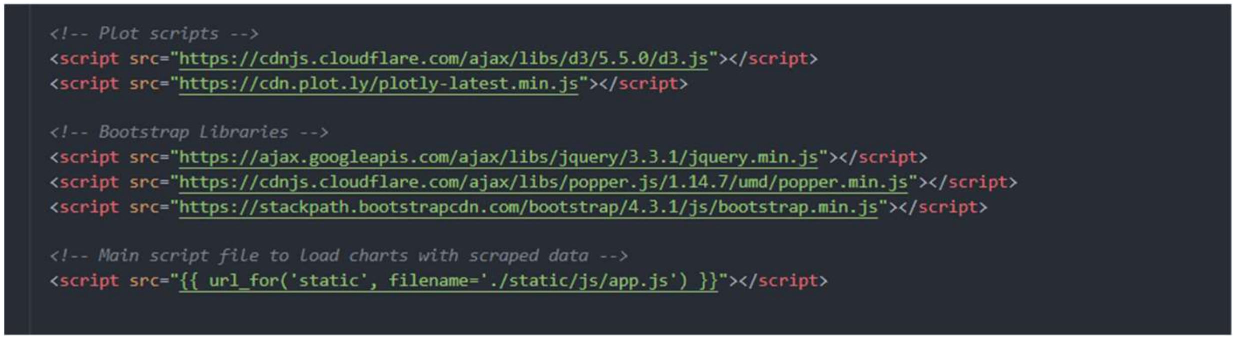
Send DataFrames to PostgreSQL

```
In [13]: # Prepare to send dataframe to postgresSQL database. Create connection
         rds_connection_string = user + ":" + password + "@localhost:5432/investmentDB"
         engine = create_engine(f'postgresql://{rds_connection_string}')

In [14]: # Check for tables
         engine.table_names()

Out[14]: ['12M_Return', 'data_summary']

In [15]: # Send the 12M dataframe to postgreSQL
         data_12M_Return.to_sql(name='12M_Return', con=engine, if_exists='replace', index=True)

In [16]: # Send the Summary dataframe to postgreSQL
         data_summary.to_sql(name='data_summary', con=engine, if_exists='replace', index=True)

In [17]: # Check for tables
         engine.table_names()

Out[17]: ['12M_Return', 'data_summary']
```

HTML

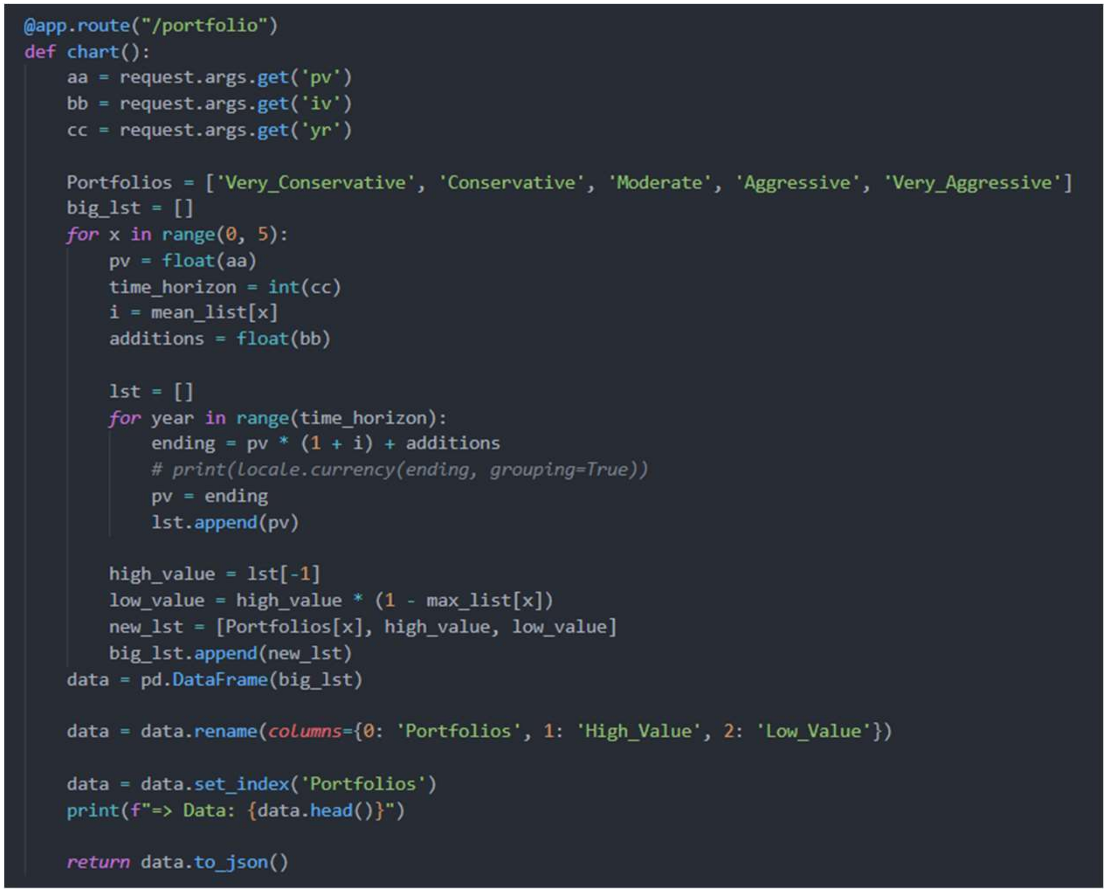


```html
<!DOCTYPE html>
<html lang="en">

<head>

  <meta charset="utf-8">
  <meta http-equiv="X-UA-Compatible" content="IE=edge">
  <meta name="viewport" content="width=device-width, initial-scale=1, shrink-to-fit=no">
  <meta name="description" content="">
  <meta name="author" content="">

  <title>Investment Portfolio</title>

  <!-- Custom fonts for this template-->
  <link href="{{ url_for('static', filename='vendor/fontawesome-free/css/all.min.css') }}" rel="stylesheet" type="text/css">
  <!-- Custom styles for this template-->
  <link href="{{ url_for('static', filename='css/sb-admin.css') }}" rel="stylesheet">
```



```html
  <!-- Plot scripts -->
  <script src="https://cdnjs.cloudflare.com/ajax/libs/d3/5.5.0/d3.js"></script>
  <script src="https://cdn.plot.ly/plotly-latest.min.js"></script>

  <!-- Bootstrap Libraries -->
  <script src="https://ajax.googleapis.com/ajax/libs/jquery/3.3.1/jquery.min.js"></script>
  <script src="https://cdnjs.cloudflare.com/ajax/libs/popper.js/1.14.7/umd/popper.min.js"></script>
  <script src="https://stackpath.bootstrapcdn.com/bootstrap/4.3.1/js/bootstrap.min.js"></script>

  <!-- Main script file to load charts with scraped data -->
  <script src="{{ url_for('static', filename='./static/js/app.js') }}"></script>
```

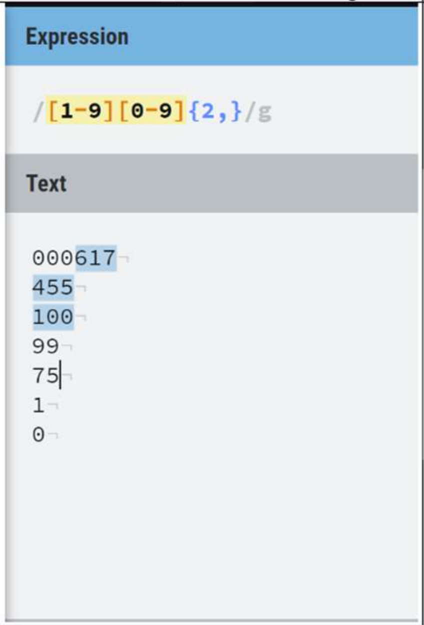
FLASK APP



```python
@app.route("/portfolio")
def chart():
    aa = request.args.get('pv')
    bb = request.args.get('iv')
    cc = request.args.get('yr')

    Portfolios = ['Very_Conservative', 'Conservative', 'Moderate', 'Aggressive', 'Very_Aggressive']
    big_lst = []
    for x in range(0, 5):
        pv = float(aa)
        time_horizon = int(cc)
        i = mean_list[x]
        additions = float(bb)

        lst = []
        for year in range(time_horizon):
            ending = pv * (1 + i) + additions
            # print(locale.currency(ending, grouping=True))
            pv = ending
            lst.append(pv)

        high_value = lst[-1]
        low_value = high_value * (1 - max_list[x])
        new_lst = [Portfolios[x], high_value, low_value]
        big_lst.append(new_lst)
    data = pd.DataFrame(big_lst)

    data = data.rename(columns={0: 'Portfolios', 1: 'High_Value', 2: 'Low_Value'})

    data = data.set_index('Portfolios')
    print(f"=> Data: {data.head()}")

    return data.to_json()
```

FORM VALIDATION

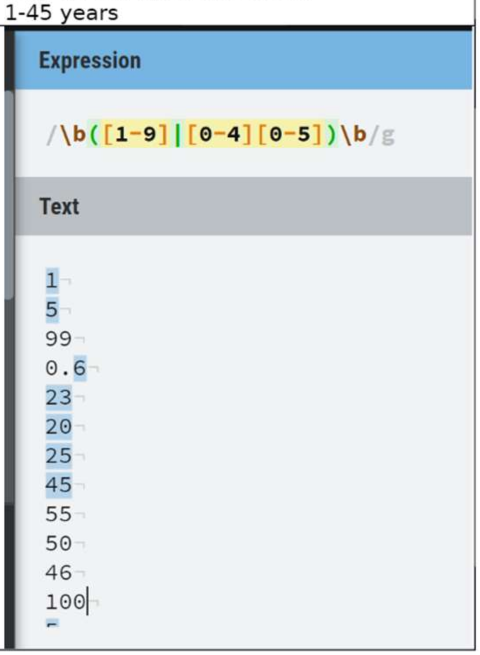
Reg Ex for Investment Amount
Whole Numbers, two or more digits



```
/[1-9][0-9]{2,}/g
```

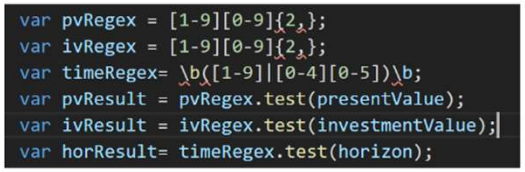
```
000617
455
100
99
75
1
0
```

Reg ex for Time Invested
1-45 years

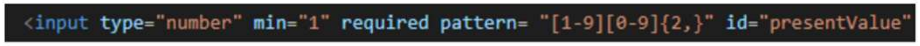


```
/\b([1-9]|[0-4][0-5])\b/g
```

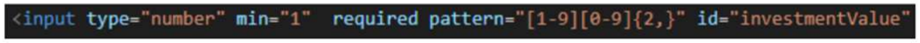
```
1
5
99
0.6
23
20
25
45
55
50
46
100
```



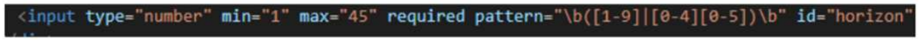
```
var pvRegex = [1-9][0-9]{2,};
var ivRegex = [1-9][0-9]{2,};
var timeRegex= \b([1-9]|[0-4][0-5])\b;
var pvResult = pvRegex.test(presentValue);
var ivResult = ivRegex.test(investmentValue);
var horResult= timeRegex.test(horizon);
```



```
<input type="number" min="1" required pattern= "[1-9][0-9]{2,}" id="presentValue"
```



```
<input type="number" min="1"  required pattern="[1-9][0-9]{2,}" id="investmentValue"
```



```
<input type="number" min="1" max="45" required pattern="\b([1-9]|[0-4][0-5])\b" id="horizon"
```

DASHBOARD

All the components together creates this:

[Investment Dashboard]