CANARY QUANT

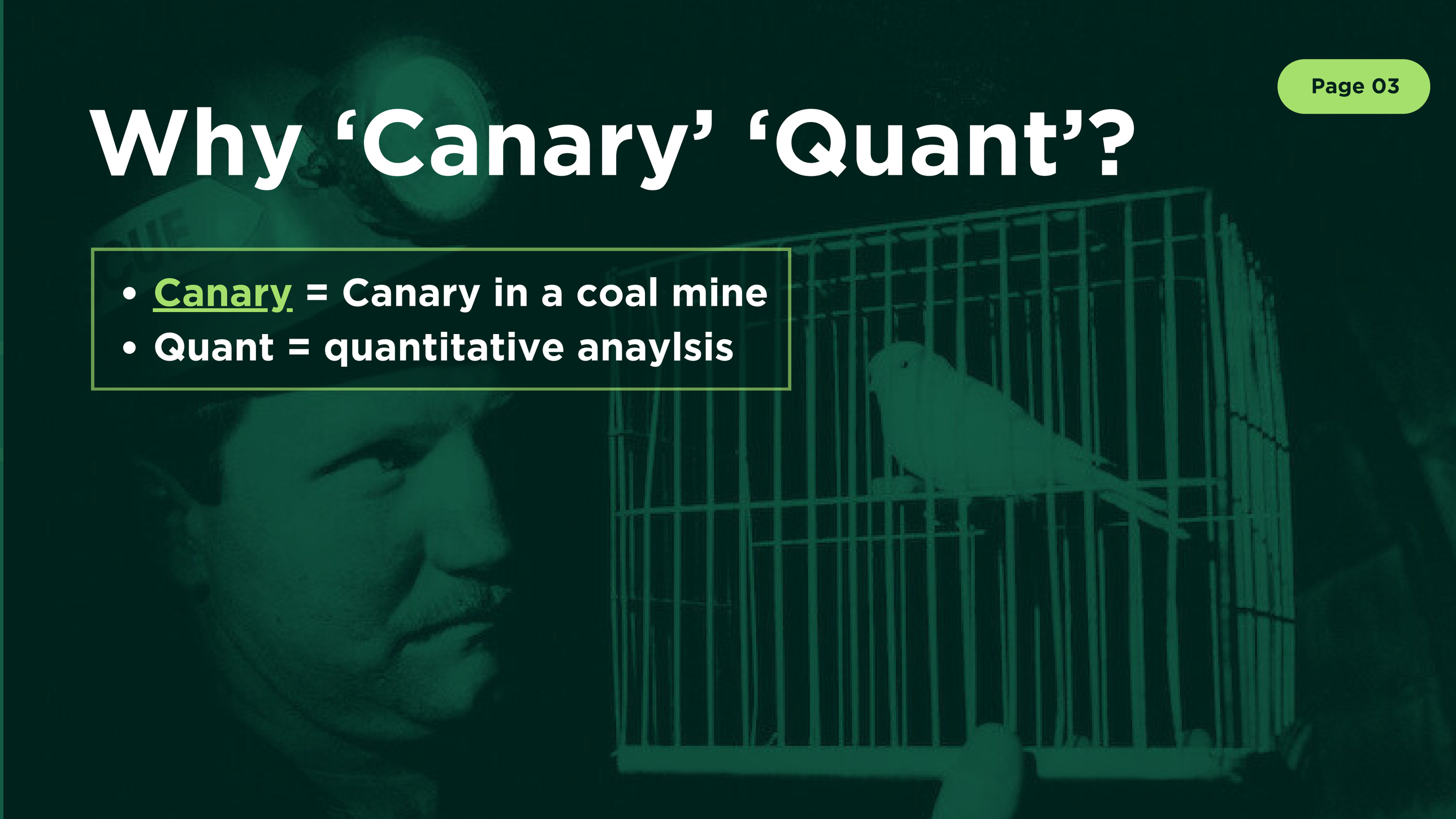
Prepared by

Gino, Elliot, Jorge,
Jenny, & David

Executive Summary

Canary Quant is a portfolio and stock analysis software that provides investors with key financial metrics to inform their investment decisions.

Why ‘Canary’ ‘Quant’?



- Canary = Canary in a coal mine
- Quant = quantitative anaylsis

Our Approach

- Assigned individual responsibilities for each member of the group
- Met every Wed to review progress and make sure everything was going along smoothly

Research Process

- Pulled stock information from Alpaca
- How do we do write these functions in python?
- How do we display the results of these calculations visually?

Financial Metrics

Cumulative Returns

```
def cumulative_returns(df, investment):
    calculation = (1 + df).cumprod()
    profit = round(investment * calculation, 2)
    profit_df = pd.DataFrame(profit)
    profit_df.columns = ['Profit']
    return profit_df
```

Measures the overall performance of an investment -- total return over time.

Financial Metrics

Beta

```
def beta(covariance, variance):
    user_beta = covariance / variance
    user_beta_df = pd.DataFrame(user_beta)
    user_beta_df.columns = ['Beta']
    user_beta_df = user_beta_df.dropna()
    return user_beta_df
```

Measures volatility relative to the overall market.

- **1 = ~ market.**
- **>1 = volatile**
- **<1 = less volatile**

Financial Metrics

Tracking Error

```python
def annual_return(df, ticker):
    total_return = (df[ticker].iloc[-1] - df[ticker].iloc[0]) / df[ticker].iloc[0]
    annual_return = round((((1 + total_return)**(1/5))-1)*100, 2)
    return annual_return
```

```python
def tracking_error(df, df2):
    return round(df - df2, 2)
```

Measures the deviation of a portfolio's returns from a benchmark index.

Financial Metrics

Variance

```python
def variance(df, market):
    rolling_variance = df[market].rolling(window=21).var()
    return rolling_variance
```

Measures the dispersion of a set of data points around their mean value. Commonly used to assess volatility.

Financial Metrics

Covariance

```
def covariance(df, tickers, market):
    covariance_rolling = df[tickers].rolling(window=21).cov(df[market])
    return covariance_rolling
```

Measures the degree to which two variables move in relation to each other.

Financial Metrics

Sharpe Ratio

```python
def sharpe_ratio(df):
    sharpe = round((df.mean()*252) / (df.std() * np.sqrt(252)), 2)
    return sharpe
```

Risk-adjusted measure to evaluates an investment's performance relative to its volatility. A higher ratio indicating a more favorable risk-reward balance.

Financial Metrics

Annualized Returns

```
def annual_return(df, ticker):
    total_return = (df[ticker].iloc[-1] - df[ticker].iloc[0]) / df[ticker].iloc[0]
    annual_return = round((((1 + total_return)**(1/5))-1)*100, 2)
    return annual_return
```

Measures the average increase on an investment each year over a certain duration.

Financial Metrics

Return on Investment (ROI)

```
def return_on_investment(df, tickers, investment):
    return round(((df[tickers].iloc[-1] - investment) / investment)*100, 2)
```

Evaluates the efficiency or profitability of an investment -- the net gain from an investment relative to its cost, expressed as a percentage.

Financial Metrics

Standard Deviation

```
def standard_deviation(df):
    rolling_std = round(df.rolling(window = 21).std()*100, 2)
    rolling_std_df = pd.DataFrame(rolling_std)
    rolling_std_df = rolling_std_df.dropna()
    return rolling_std_df
```

Measures the dispersion of a set of data from its mean. Commonly used to sense volatility.

Financial Metrics

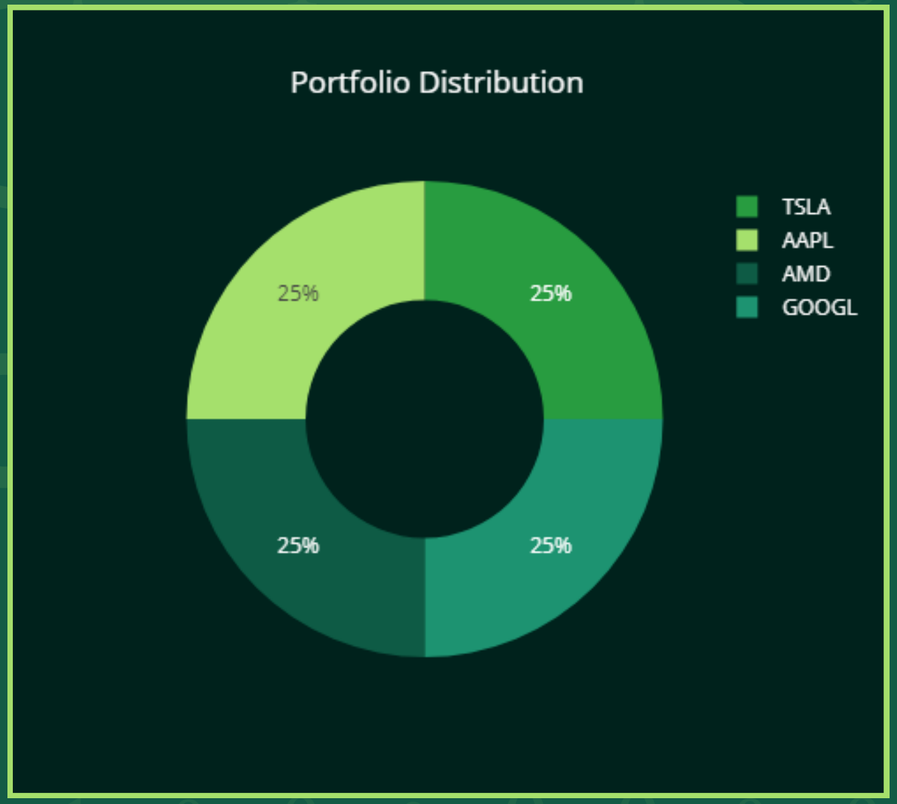
Drawdown

```python
def daily_drawdown(df):
    roll_max = df.cummax()
    roll_min = df.cummin()
    daily_drawdown = round(((roll_max - roll_min) / roll_max)*100, 2)
    return daily_drawdown
```

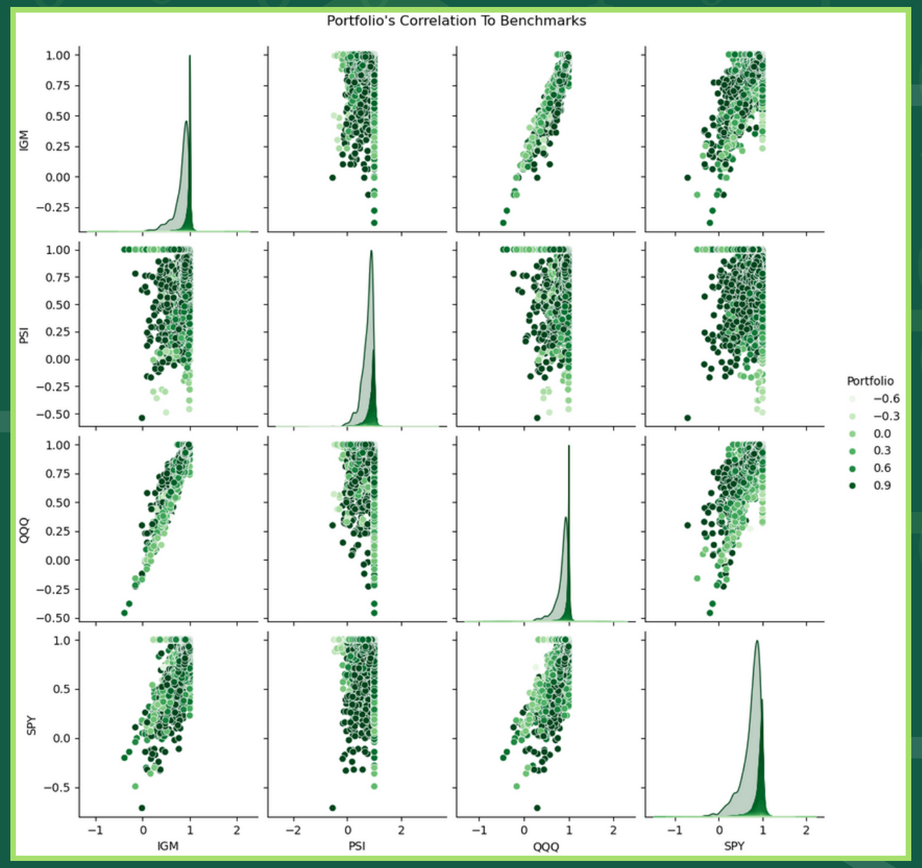
The peak-to-trough decline in the value of an investment.

Visualizations

```python
def portfolio_distribution_chart(tickers, weights):
    chart = px.pie(values=weights, title='Portfolio Distribution', names=tickers, hole=.5,
                   color_discrete_sequence=['#289c40', '#a5e06c', '#0e5b45', '#1d9371'])
    chart.update_layout({
        'plot_bgcolor': '#00221c',
        'paper_bgcolor': '#00221c'
    })
    chart.update_layout(legend={'font': {'color': 'white'}})
    chart.update_layout(title={'font': {'color': 'white'}})
    chart.update_layout(title_x=0.487)
    chart.update_layout(width=500)
    return chart
```



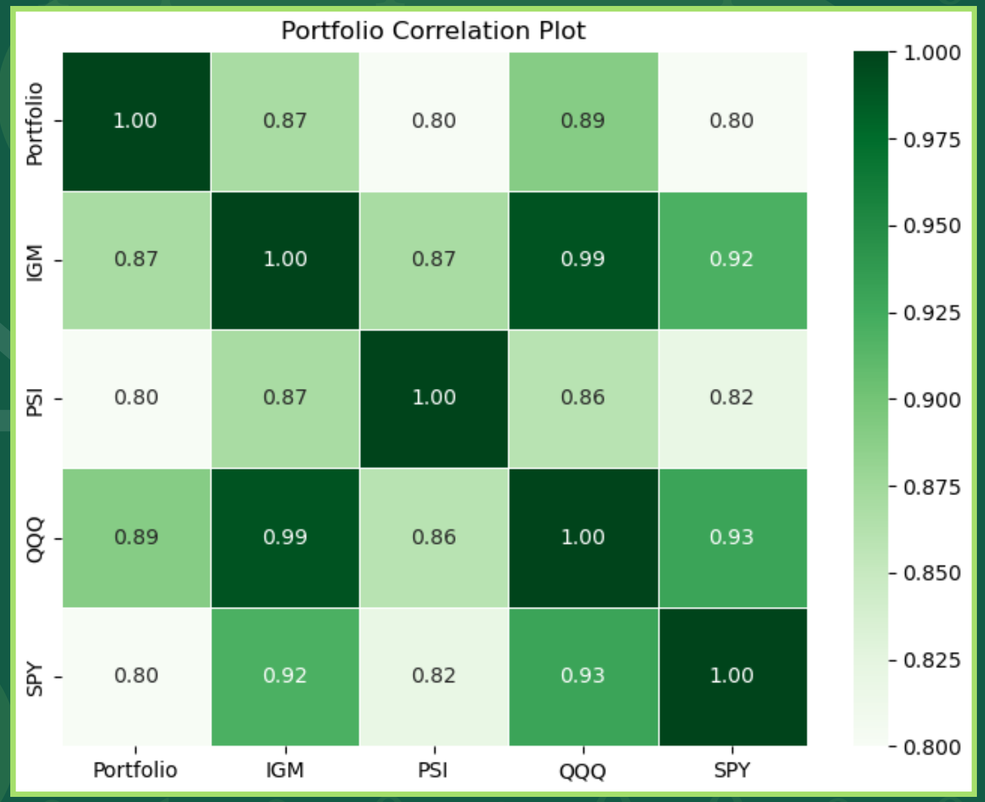
Visualizations




```python
def correlation_scatter_chart(df):
    sns.pairplot(df, hue='Portfolio', palette='Greens')
    plt.suptitle("Portfolio's Correlation To Benchmarks", y=1.02)
    return plt.show()
```

Visualizations

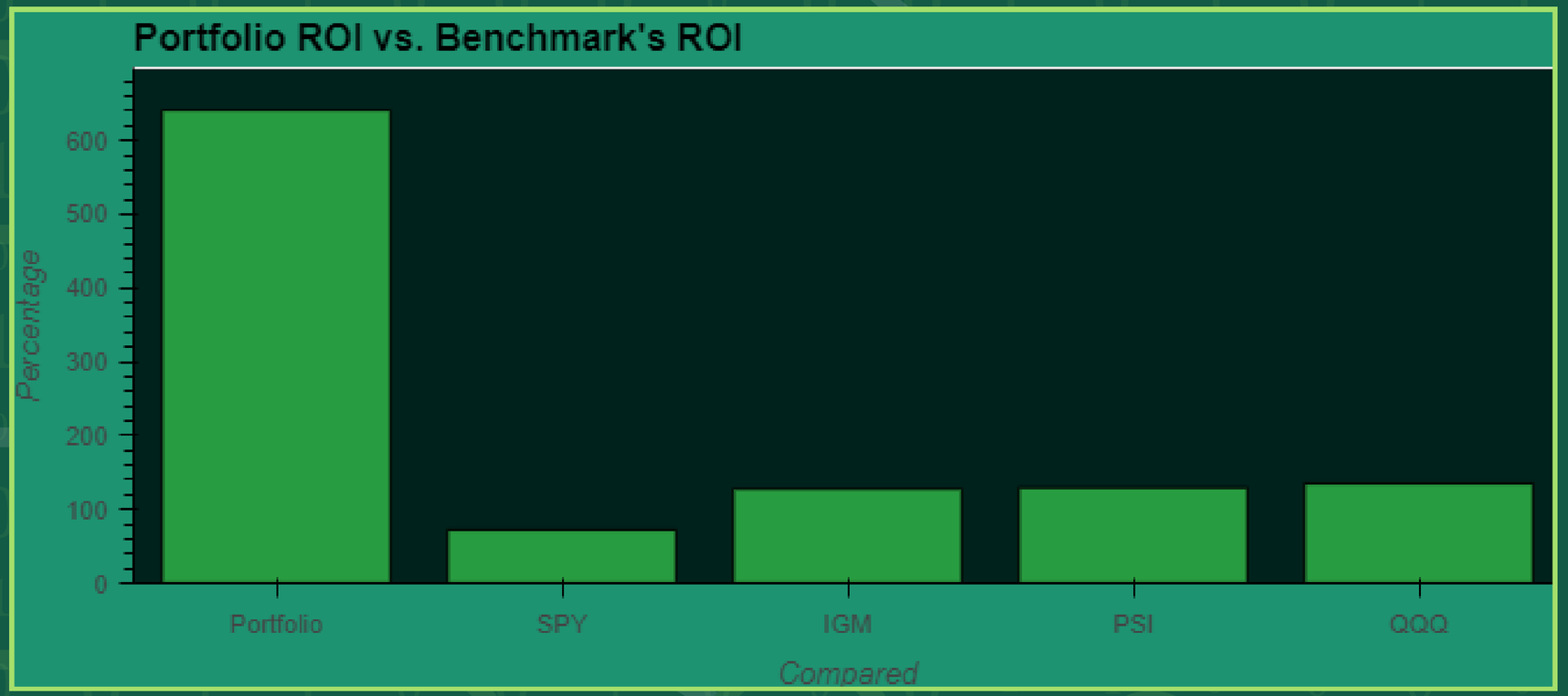
```python
def correlation_heatmap(df):
    plt.figure(figsize=(8,6))
    sns.heatmap(df, annot=True, cmap='Greens', fmt='.2f', linewidths=.5)
    plt.title('Portfolio Correlation Plot')
    return plt.show()
```



presented by xxx

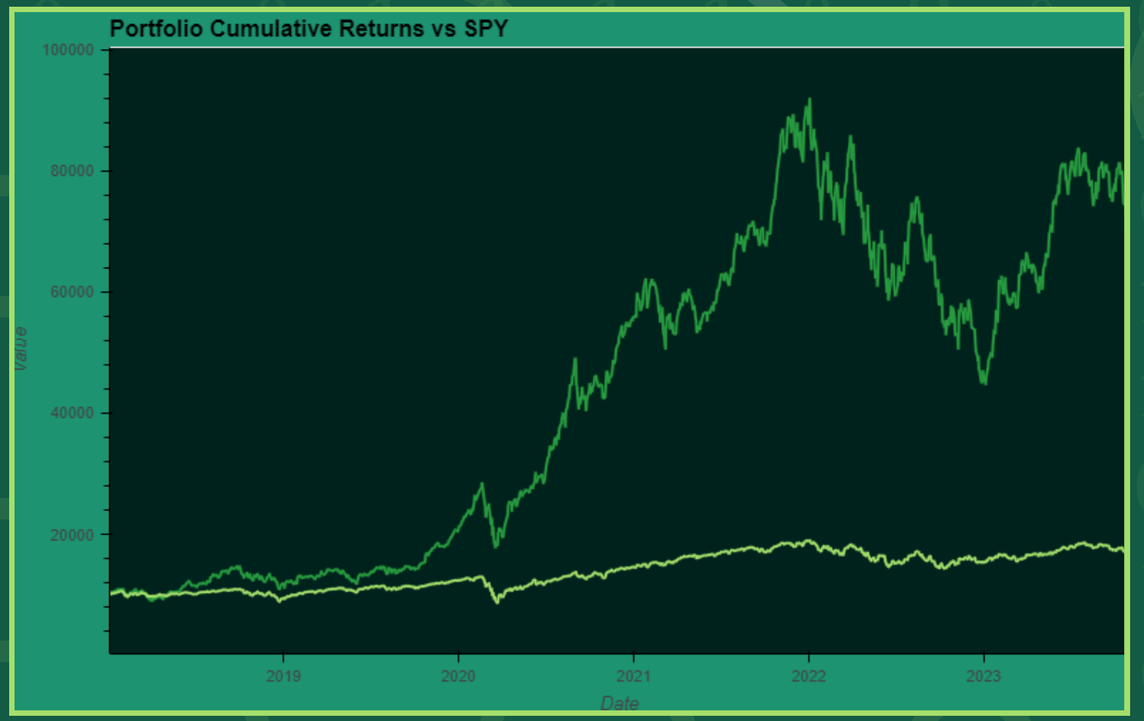
Visualizations

```python
def roi_chart(df, compare, percent):
    chart = df.hvplot.bar(x=compare, y=percent, color='#289c40', title="Portfolio ROI vs. Benchmark's ROI", ylabel='Percentage')
    chart.opts(bgcolor='#00221c')
    chart.opts(hooks=[lambda p, _: p.state.update(border_fill_color='#1d9371')])
    return chart
```



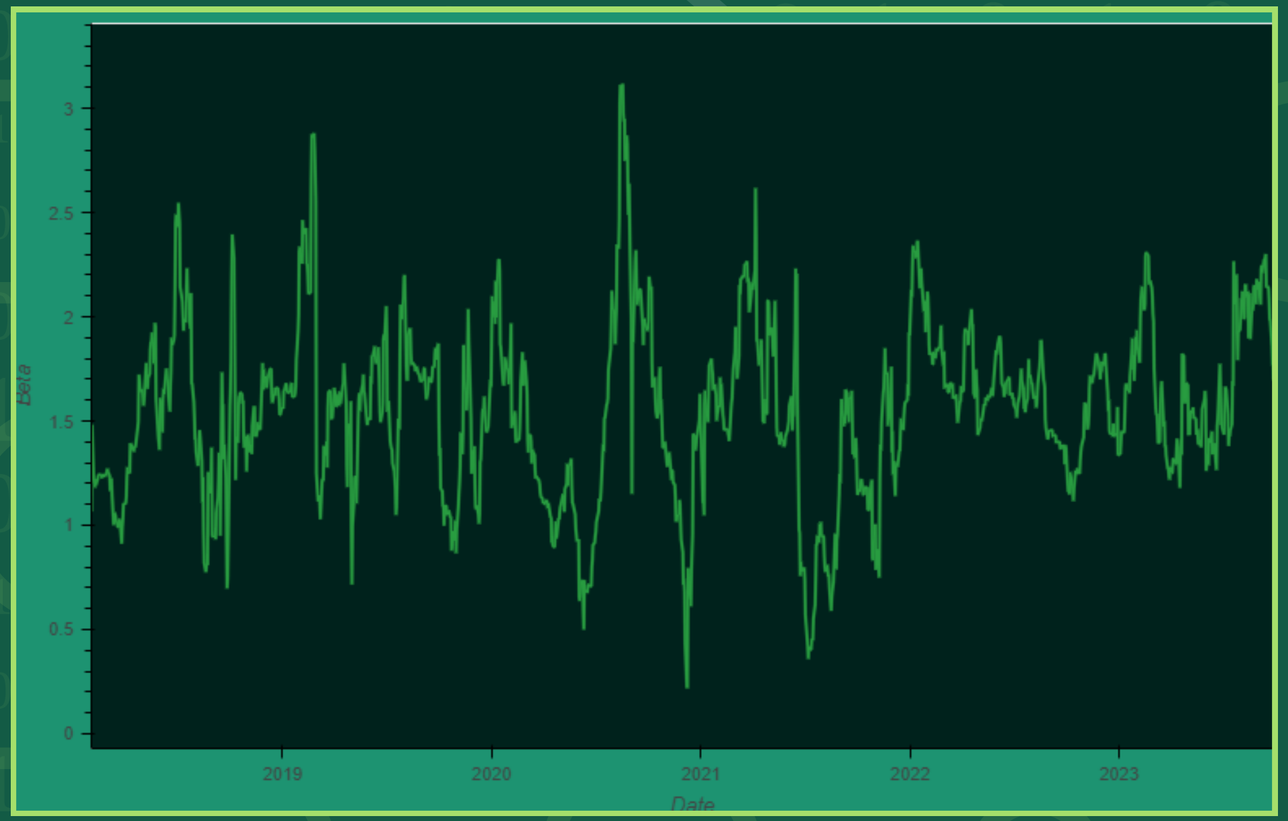
Visualizations

```
def cumulative_return_chart(df, tickers, market, date):
    formatter = DatetimeTickFormatter(months='%b %Y')
    first = df.hvplot.line(x=date, y=tickers, title='Portfolio Cumulative Returns vs SPY',
                           color='#289c40', height=500, width=820, xformatter=formatter, yformatter='%.0f')
    second = df.hvplot.line(x=date, y=market, color='#a5e06c', height=500,
                            width=820, xformatter=formatter, yformatter='%.0f')
    overlay = first * second
    overlay.opts(ylabel='Value')
    overlay.opts(bgcolor='#00221c')
    overlay.opts(hooks=[lambda p, _: p.state.update(border_fill_color='#1d9371')])
    return overlay
```



Visualizations

```python
def beta_chart(df):
    formatter = DatetimeTickFormatter(months='%b %Y')
    chart = df.hvplot.line(x='Date', y='Beta', value_label='Beta', color='#289c40', legend='top', height=500, width=820, xformatter=formatter)
    chart.opts(bgcolor='#00221c')
    chart.opts(hooks=[lambda p, _: p.state.update(border_fill_color='#1d9371')])
    return chart
```



Obstacles

Calculations

- Creating a class of defined functions
- Stock splits were difficult to incorporate

Visualization

- Data Interation
- Plotting certain data types

User Input

- Front end frameworks (css, d3.js)
- Organizing the layout
- Integrating code from teammates

Future Projects Will...

- Seek to integrate monthly contributions & dividend yields into calculations
- Ultimately transition calculation functions into an algorithmic trading software & integrate with our UI
- Investigate AI integrations, especially following OpenAI's recent announcements about GPTs
- Deploy stocks that IPOd within last 5 years

User Interface

Thank You