Mutual Fund Sector-wise Analysis

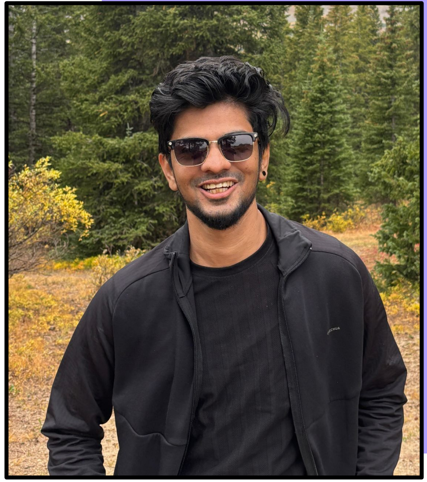
Ashwini Dubbewar
Dhruv Pathak
Kshitij Jadhav

CS 5502

OUR TEAM

Ashwini Dubbewar

Dhruv Pathak



Kshitij Jadhav

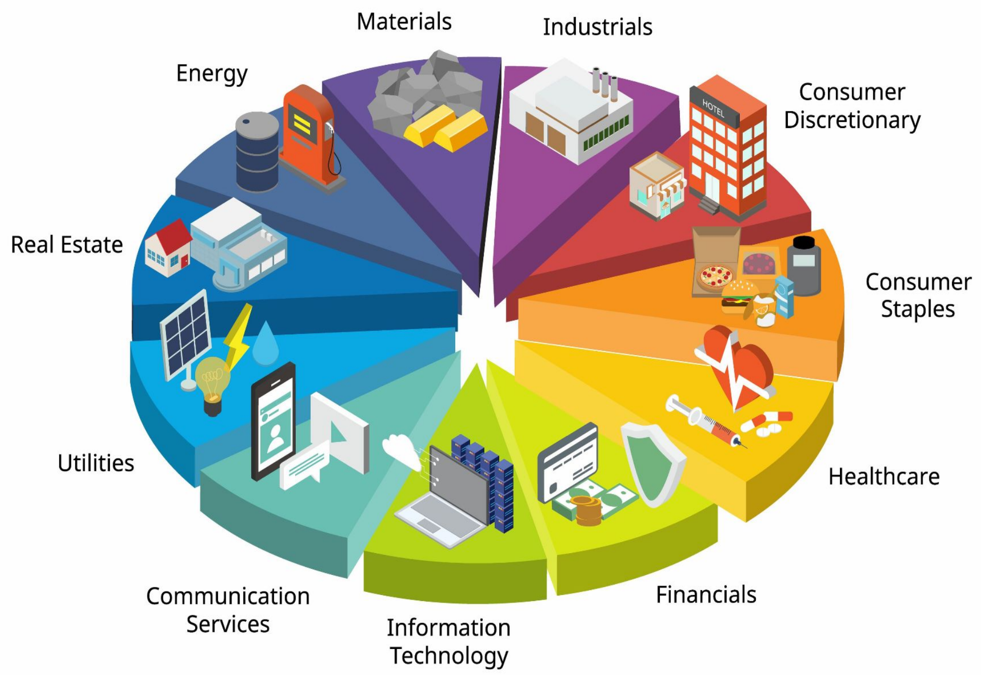
What are Mutual Funds ?

A **mutual fund** is an investment vehicle that pools money from many investors to buy a diversified portfolio of stocks, bonds, or other securities. It is managed by a professional fund manager, who decides how the money is invested based on the fund's goals.

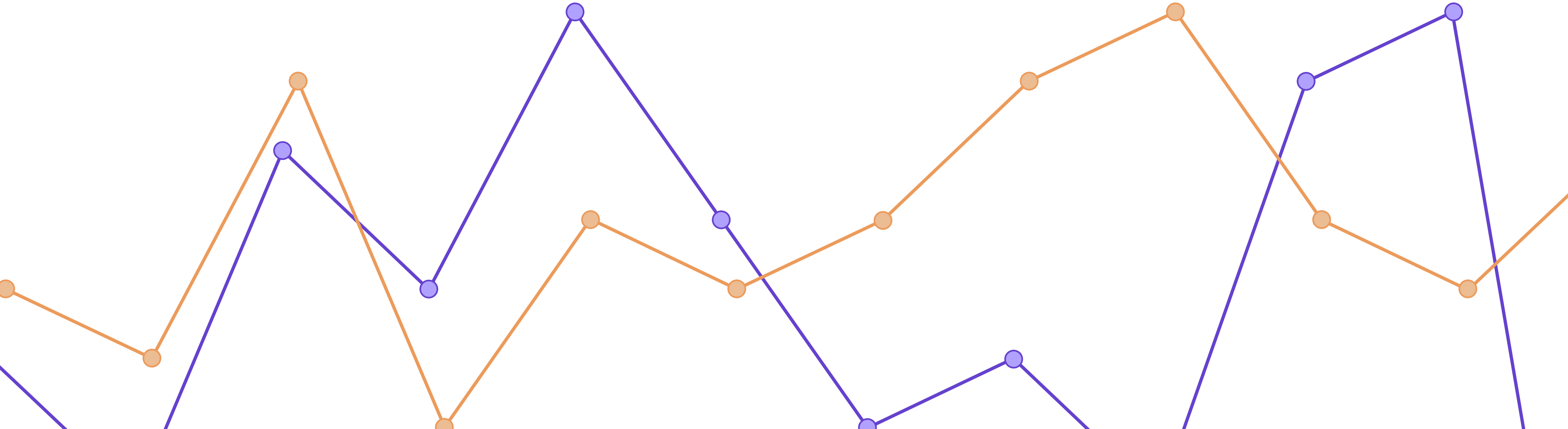
Benefits of Mutual Funds: If you Investing in a mutual fund allows you to invest in a wide range of assets without having to buy individual stocks or bonds yourself.

Sectors Wise Mutual Funds (MF) Investments

It is a type of mutual fund that focuses on investing in a particular industry or sector of the economy. These funds aim to provide investors exposure to specific sectors they believe will outperform the broader market.



Background Studies



Previous Studies on Mutual Fund Sector Allocation and Performance

Sector Exposure and Mutual Fund Performance in Global Financial Markets

- **Findings:** This study looks at how sector exposure impacts the risk and return profiles of mutual funds. It concluded that sector exposure plays a crucial role in driving mutual fund performance, especially in periods of high market volatility. Funds heavily exposed to defensive sectors (e.g., Utilities, Healthcare) tend to perform better during market downturns.
- **Methodology:** The study employed performance attribution analysis to separate the effects of sector exposure from stock selection and timing abilities.

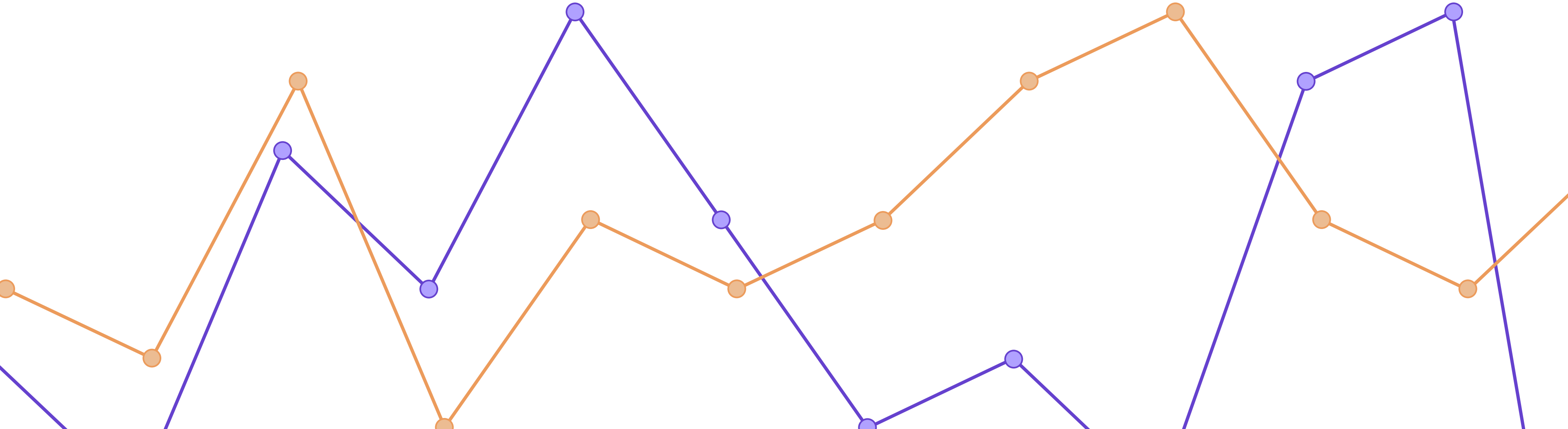
Sector-Specific Risk Factors and Mutual Fund Performance

- **Findings:** The study investigated how sector-specific risk factors influence the risk-adjusted returns of mutual funds. It concluded that funds with concentrated exposure to high-risk sectors (such as Technology or Energy) tend to exhibit greater performance volatility but also potential for higher returns in favorable market conditions..
- **Methodology:** The authors applied multi-factor models to assess how different sector exposures contribute to fund risk and returns, controlling for market-wide risks.

The Impact of Sector Allocation on Mutual Fund Performance

- **Findings:** This study investigates the role of sector allocation in the performance of mutual funds. It found that mutual funds with superior sector allocation strategies tend to outperform those that rely primarily on stock picking. Sector-specific expertise can significantly enhance overall fund performance.
- **Methodology:** The researchers used regression models to examine the relationship between sector allocation and fund returns, controlling for other factors such as stock selection ability and market timing.

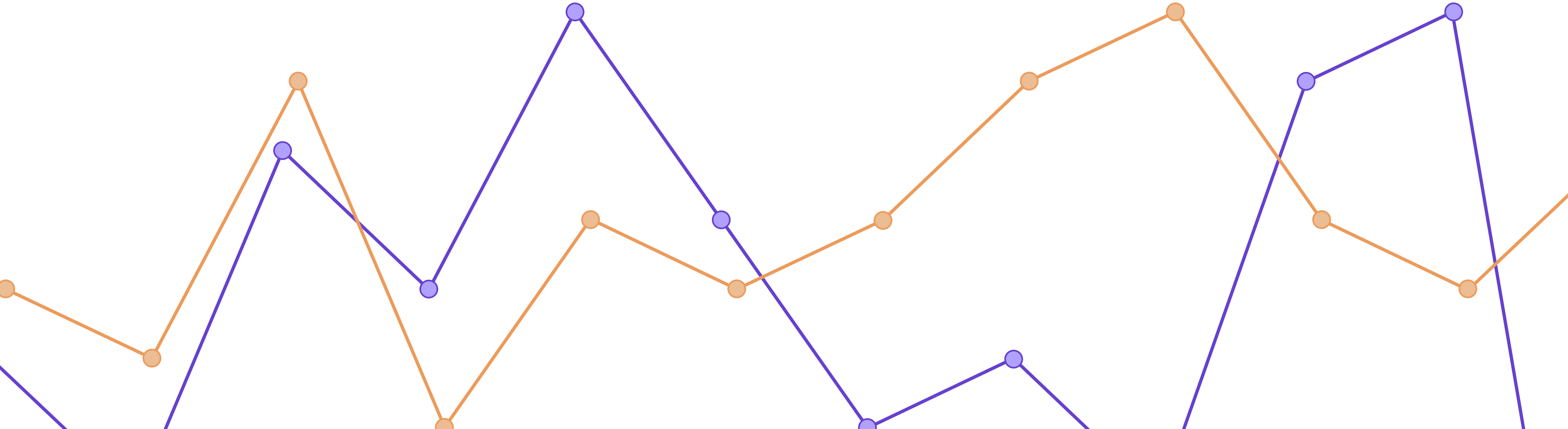
Project Motivation



Financial markets have always been a fascinating area of study for our team. Through extensive research and background analysis, we identified significant advancements in risk-based analysis across various sectors, such as healthcare and finance. However, a notable gap emerged: **the lack of in-depth, risk-based analysis for individual mutual funds within these sectors.**

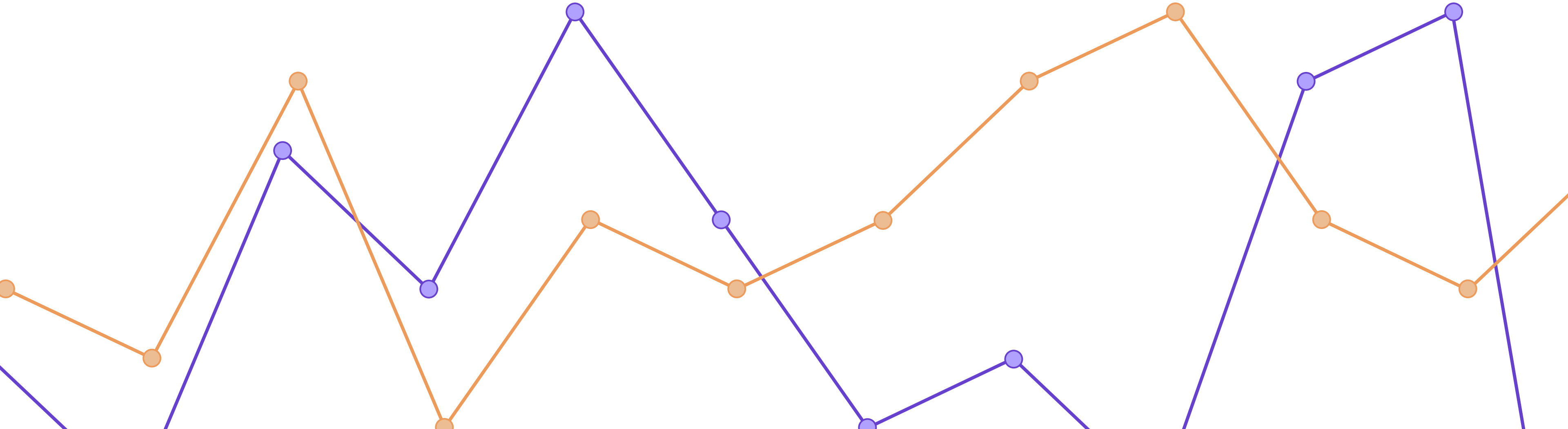
This discovery became the driving force behind our project, motivating us to address this gap. Our objective was to develop a comprehensive risk assessment framework tailored to mutual funds. Additionally, we aimed to **leverage data mining models to analyze large-cap and mid-cap datasets, providing valuable insights into their performance and associated risks.**

Understanding Financial Terms from the Datasets



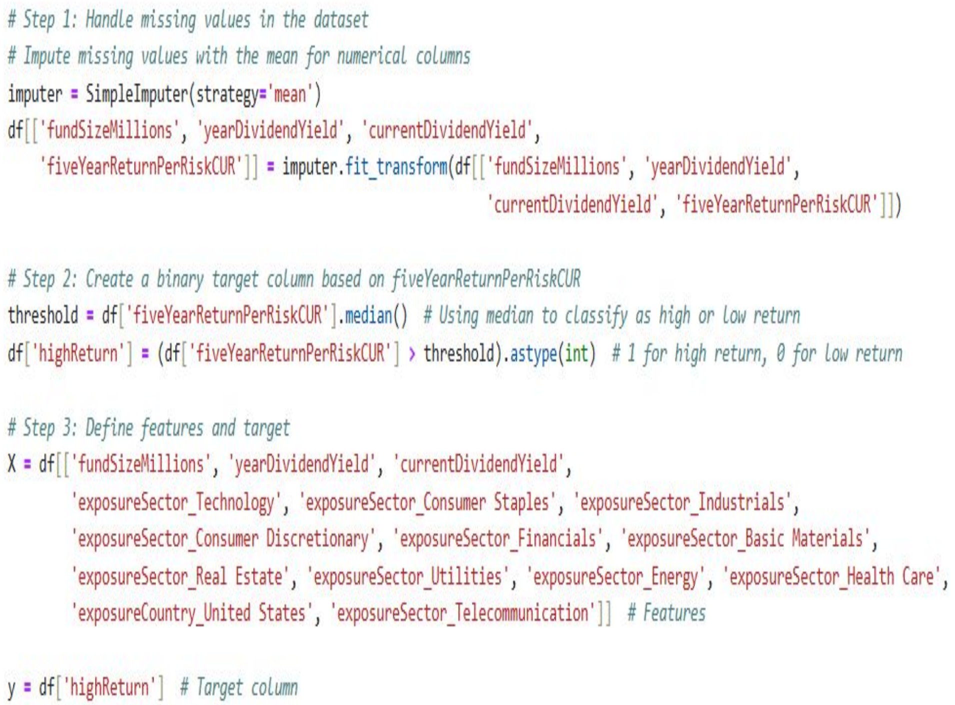
Total Assets: This represents the size of the fund, indicating the total amount of money invested. Larger funds may have more stability and resources, which can influence fund performance.
YTD (Year-to-Date): Shows the fund's performance so far this year, helping investors assess its recent growth or decline.
Avg Volume: Indicates the average number of shares traded daily, which gives an idea of liquidity. Higher average volume suggests easier buying and selling without impacting the price.
Annual Dividends: The yearly payout per share to investors, representing the income generated by the fund, which is particularly relevant for income-focused investors.
P/E Ratio (Price-to-Earnings Ratio): Helps evaluate the fund’s valuation. A lower P/E ratio can indicate that the fund is more "affordable" compared to its earnings.
Beta: Measures how volatile the fund is relative to the market. A beta above 1 means the fund is more volatile, while below 1 suggests it's less volatile than the market.
RSI (Relative Strength Index): A momentum indicator that shows if the fund is overbought (high RSI) or oversold (low RSI), which can be useful for timing buys and sells.
Upper Bollinger Band: It indicates a potential resistance level where prices may be considered high or overbought.
Lower Bollinger Band: It indicates a potential support level where prices may be considered low or underbought.
5 Year % Change: The 5-Year Percentage Change in stocks measures the growth or decline in a stock's price over a five-year period. This metric is essential for investors who want to evaluate long-term performance and trends in their investments.

Model Training

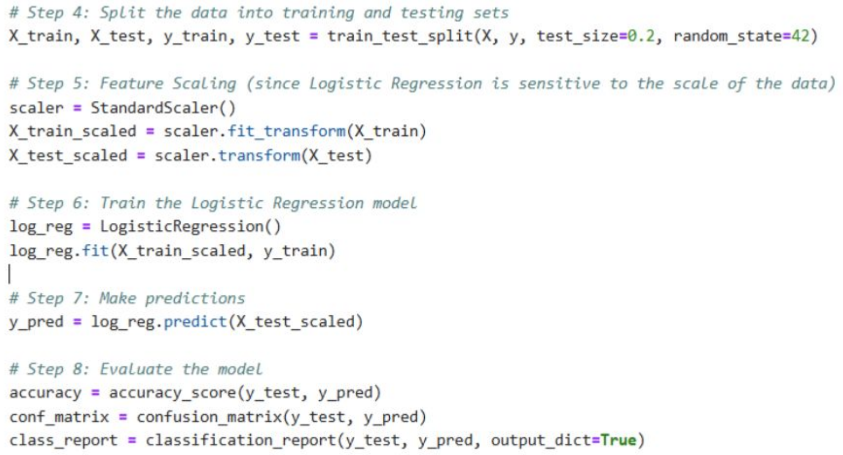


Logistic Regression

Code Execution

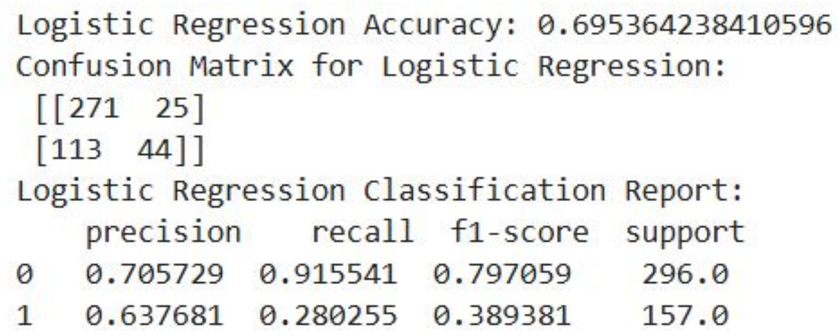


```
# Step 1: Handle missing values in the dataset
# Impute missing values with the mean for numerical columns
imputer = SimpleImputer(strategy='mean')
df[['fundSizeMillions', 'yearDividendYield', 'currentDividendYield',
    'fiveYearReturnPerRiskCUR']] = imputer.fit_transform(df[['fundSizeMillions', 'yearDividendYield',
                                                              'currentDividendYield', 'fiveYearReturnPerRiskCUR']])

# Step 2: Create a binary target column based on fiveYearReturnPerRiskCUR
threshold = df['fiveYearReturnPerRiskCUR'].median()  # Using median to classify as high or low return
df['highReturn'] = (df['fiveYearReturnPerRiskCUR'] > threshold).astype(int)  # 1 for high return, 0 for low return

# Step 3: Define features and target
X = df[['fundSizeMillions', 'yearDividendYield', 'currentDividendYield',
        'exposureSector_Technology', 'exposureSector_Consumer Staples', 'exposureSector_Industrials',
        'exposureSector_Consumer Discretionary', 'exposureSector_Financials', 'exposureSector_Basic Materials',
        'exposureSector_Real Estate', 'exposureSector_Utilities', 'exposureSector_Energy', 'exposureSector_Health Care',
        'exposureCountry_United States', 'exposureSector_Telecommunication']]  # Features

y = df['highReturn']  # Target column
```



```
# Step 4: Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Step 5: Feature Scaling (since Logistic Regression is sensitive to the scale of the data)
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Step 6: Train the Logistic Regression model
log_reg = LogisticRegression()
log_reg.fit(X_train_scaled, y_train)

# Step 7: Make predictions
y_pred = log_reg.predict(X_test_scaled)

# Step 8: Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
conf_matrix = confusion_matrix(y_test, y_pred)
class_report = classification_report(y_test, y_pred, output_dict=True)
```

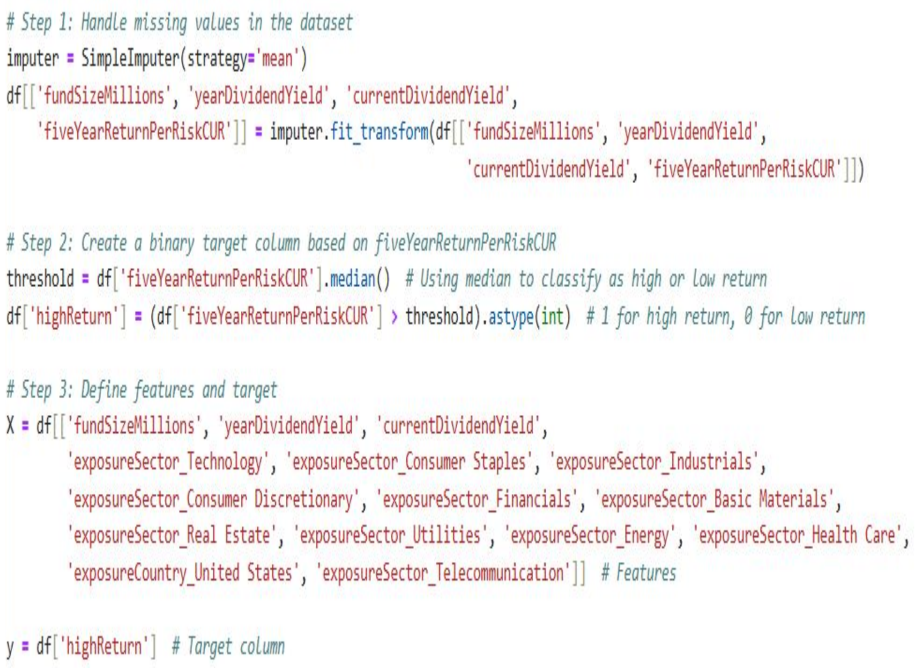
Output Results



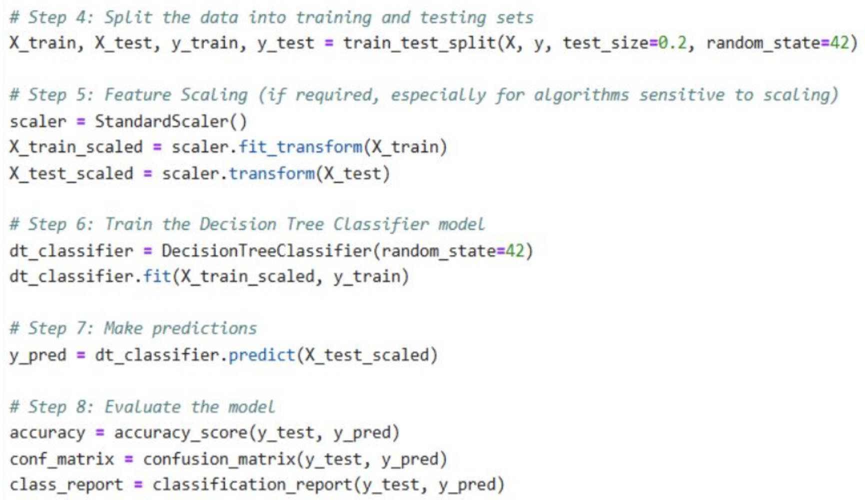
```
Logistic Regression Accuracy: 0.695364238410596
Confusion Matrix for Logistic Regression:
 [[271  25]
 [113  44]]
Logistic Regression Classification Report:
    precision    recall  f1-score   support
0    0.705729  0.915541  0.797059     296.0
1    0.637681  0.280255  0.389381     157.0
```

Decision Tree

Code Execution

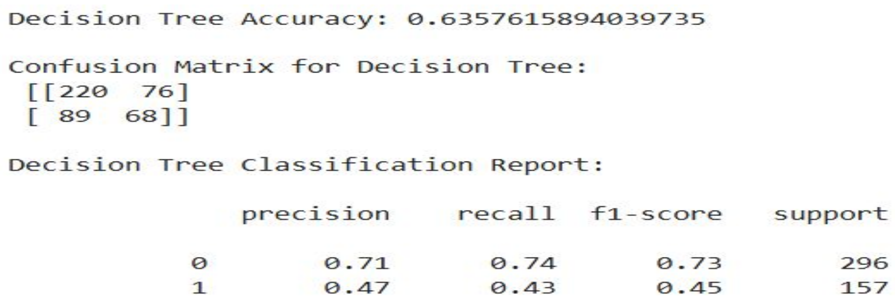


```
# Step 1: Handle missing values in the dataset
imputer = SimpleImputer(strategy='mean')
df[['fundSizeMillions', 'yearDividendYield', 'currentDividendYield',
    'fiveYearReturnPerRiskCUR']] = imputer.fit_transform(df[['fundSizeMillions', 'yearDividendYield',
                                                           'currentDividendYield', 'fiveYearReturnPerRiskCUR']])

# Step 2: Create a binary target column based on fiveYearReturnPerRiskCUR
threshold = df['fiveYearReturnPerRiskCUR'].median()  # Using median to classify as high or low return
df['highReturn'] = (df['fiveYearReturnPerRiskCUR'] > threshold).astype(int)  # 1 for high return, 0 for low return

# Step 3: Define features and target
X = df[['fundSizeMillions', 'yearDividendYield', 'currentDividendYield',
        'exposureSector_Technology', 'exposureSector_Consumer Staples', 'exposureSector_Industrials',
        'exposureSector_Consumer Discretionary', 'exposureSector_Financials', 'exposureSector_Basic Materials',
        'exposureSector_Real Estate', 'exposureSector_Utilities', 'exposureSector_Energy', 'exposureSector_Health Care',
        'exposureCountry_United States', 'exposureSector_Telecommunication']]  # Features

y = df['highReturn']  # Target column
```



```
# Step 4: Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Step 5: Feature Scaling (if required, especially for algorithms sensitive to scaling)
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Step 6: Train the Decision Tree Classifier model
dt_classifier = DecisionTreeClassifier(random_state=42)
dt_classifier.fit(X_train_scaled, y_train)

# Step 7: Make predictions
y_pred = dt_classifier.predict(X_test_scaled)

# Step 8: Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
conf_matrix = confusion_matrix(y_test, y_pred)
class_report = classification_report(y_test, y_pred)
```

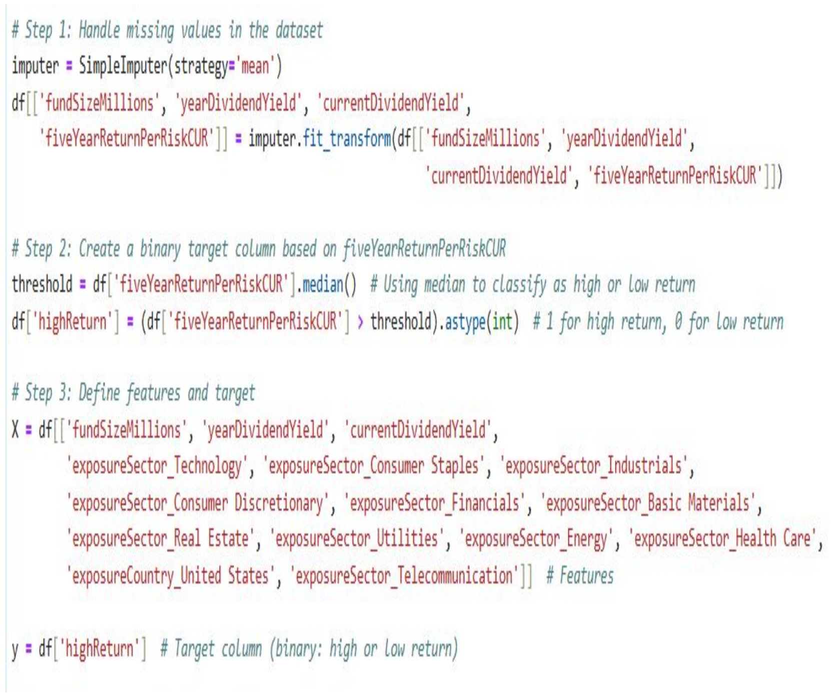
Output Results



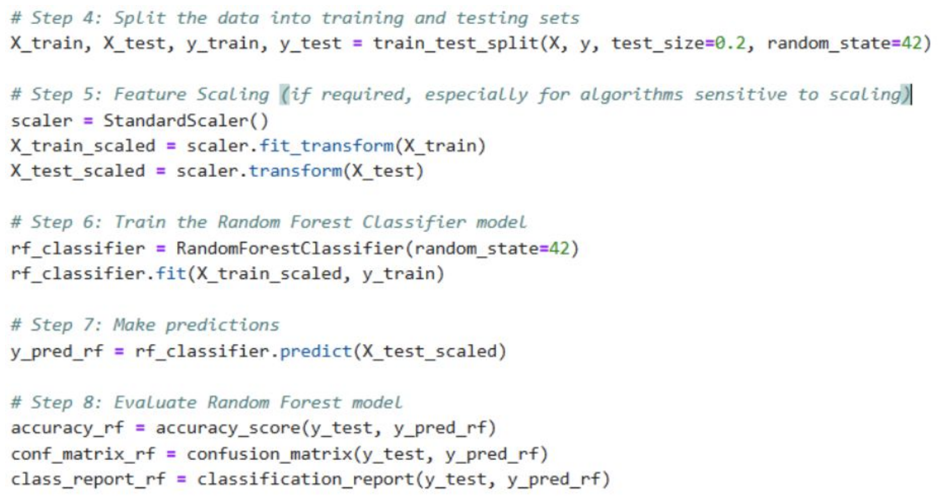
```
Decision Tree Accuracy: 0.6357615894039735

Confusion Matrix for Decision Tree:
 [[220  76]
 [ 89  68]]

Decision Tree Classification Report:

              precision    recall  f1-score   support

           0       0.71      0.74      0.73       296
           1       0.47      0.43      0.45       157
```

Random Forest

Code Execution

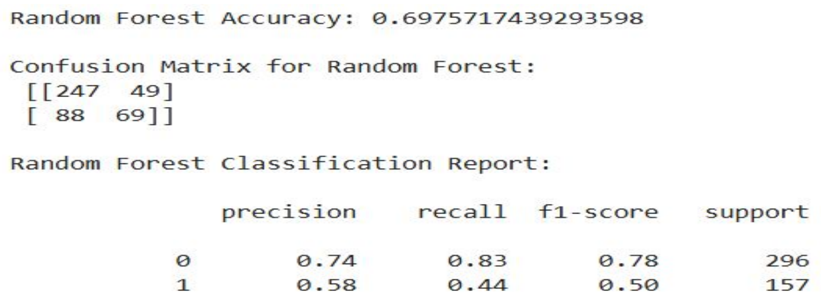


```python
# Step 1: Handle missing values in the dataset
imputer = SimpleImputer(strategy='mean')
df[['fundSizeMillions', 'yearDividendYield', 'currentDividendYield',
    'fiveYearReturnPerRiskCUR']] = imputer.fit_transform(df[['fundSizeMillions', 'yearDividendYield',
                                                          'currentDividendYield', 'fiveYearReturnPerRiskCUR']])

# Step 2: Create a binary target column based on fiveYearReturnPerRiskCUR
threshold = df['fiveYearReturnPerRiskCUR'].median()  # Using median to classify as high or low return
df['highReturn'] = (df['fiveYearReturnPerRiskCUR'] > threshold).astype(int)  # 1 for high return, 0 for low return

# Step 3: Define features and target
X = df[['fundSizeMillions', 'yearDividendYield', 'currentDividendYield',
        'exposureSector_Technology', 'exposureSector_Consumer Staples', 'exposureSector_Industrials',
        'exposureSector_Consumer Discretionary', 'exposureSector_Financials', 'exposureSector_Basic Materials',
        'exposureSector_Real Estate', 'exposureSector_Utilities', 'exposureSector_Energy', 'exposureSector_Health Care',
        'exposureCountry_United States', 'exposureSector_Telecommunication']]  # Features

y = df['highReturn']  # Target column (binary: high or low return)
```



```python
# Step 4: Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Step 5: Feature Scaling (if required, especially for algorithms sensitive to scaling)
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Step 6: Train the Random Forest Classifier model
rf_classifier = RandomForestClassifier(random_state=42)
rf_classifier.fit(X_train_scaled, y_train)

# Step 7: Make predictions
y_pred_rf = rf_classifier.predict(X_test_scaled)

# Step 8: Evaluate Random Forest model
accuracy_rf = accuracy_score(y_test, y_pred_rf)
conf_matrix_rf = confusion_matrix(y_test, y_pred_rf)
class_report_rf = classification_report(y_test, y_pred_rf)
```

Output Results

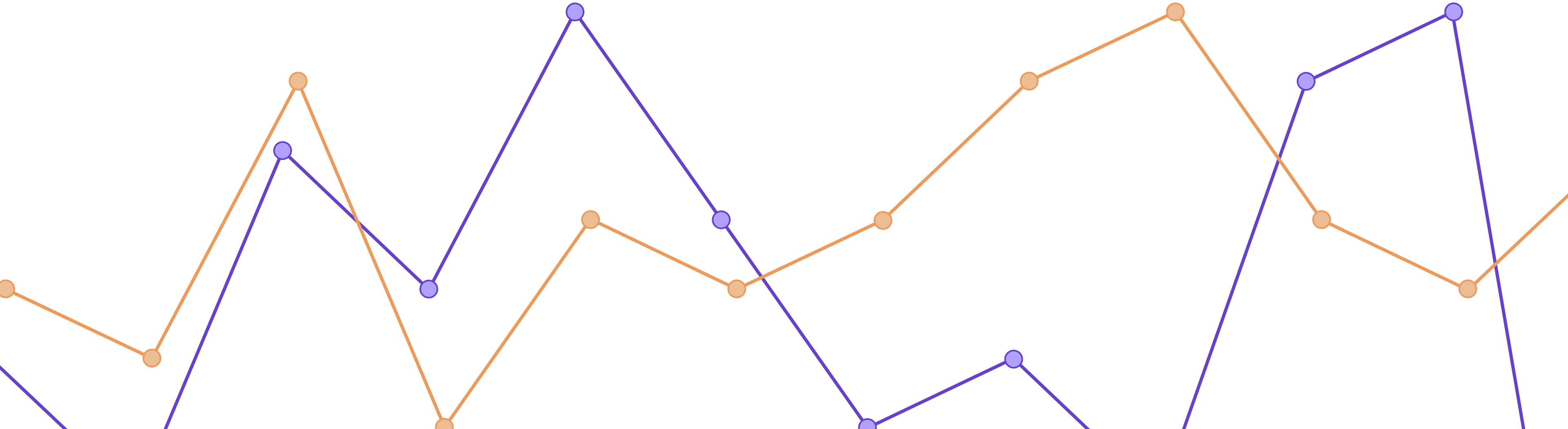


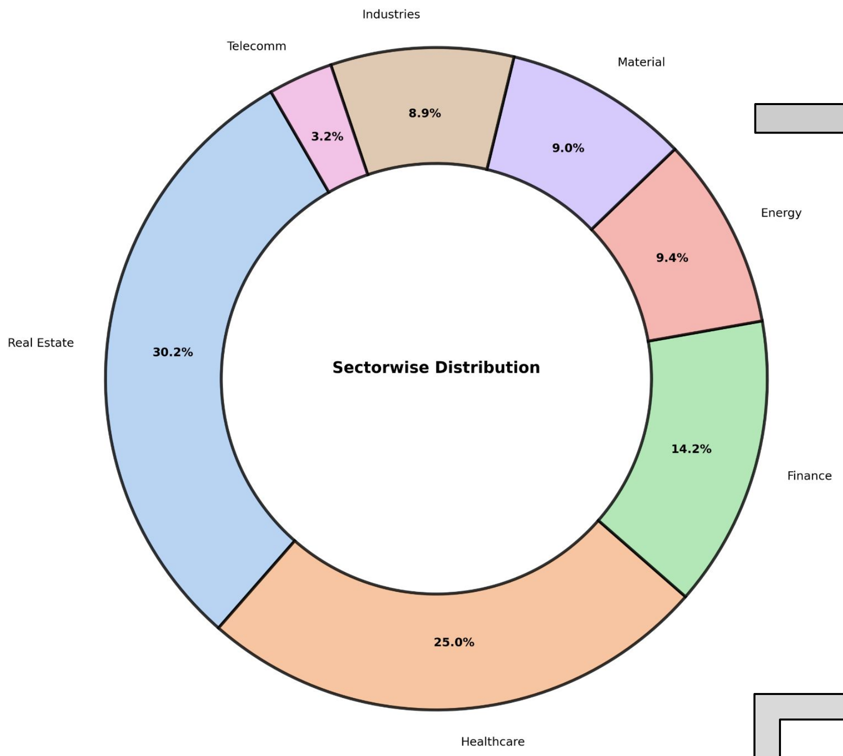
```
Random Forest Accuracy: 0.6975717439293598

Confusion Matrix for Random Forest:
 [[247  49]
 [ 88  69]]

Random Forest Classification Report:

              precision    recall  f1-score   support

           0       0.74      0.83      0.78       296
           1       0.58      0.44      0.50       157
```

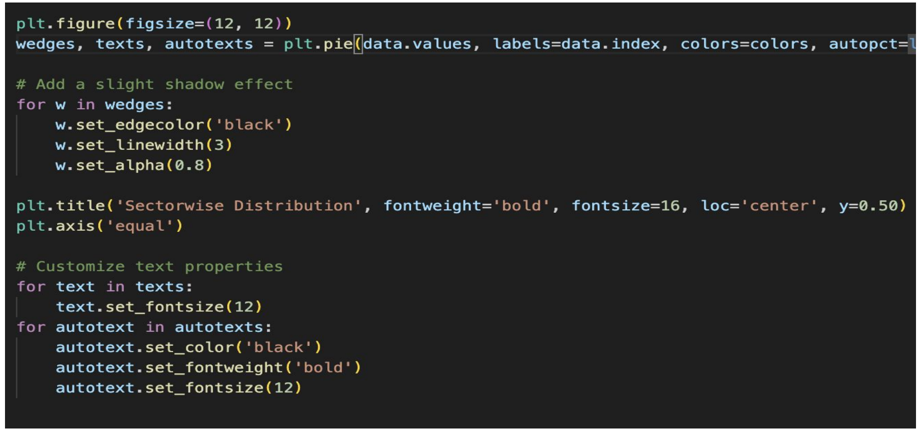
Comparing Models

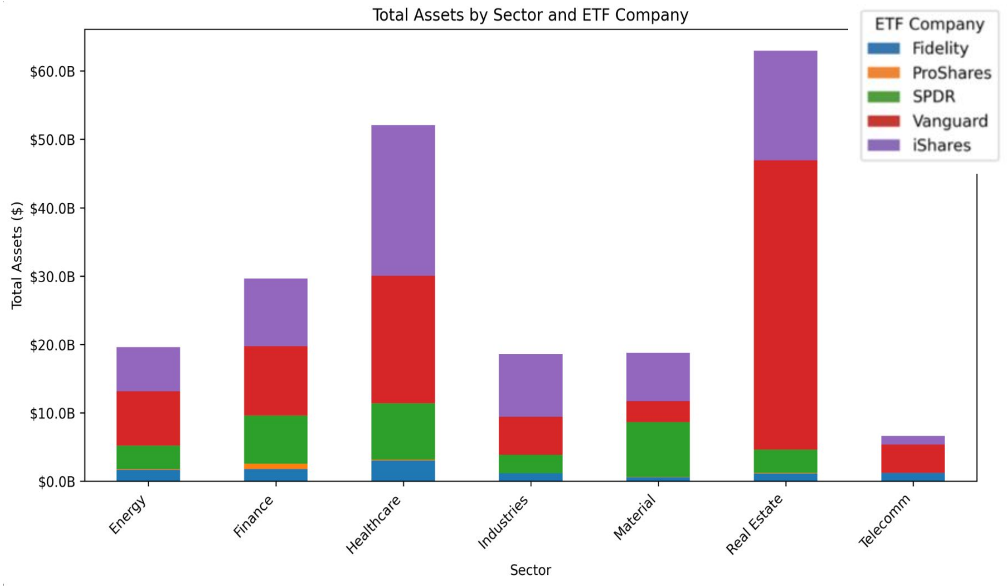
Model	Accuracy	Precision (0)	Recall (0)	F1-Score (0)	Precision (1)	Recall (1)	F1-Score (1)
Random Forest	0.6976	0.7400	0.8300	0.7800	0.5800	0.4400	0.5000
Decision Tree	0.6358	0.7100	0.7400	0.7300	0.4700	0.4300	0.4500
Logistic Regression	0.6954	0.7057	0.9155	0.7971	0.6377	0.2803	0.3894

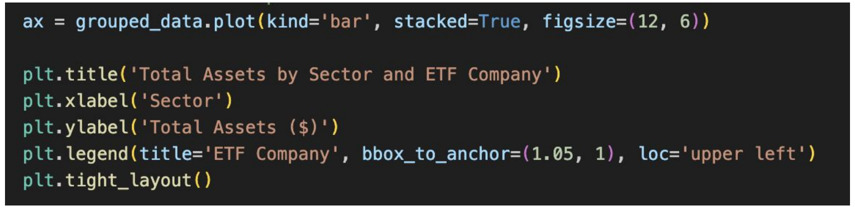
ETF and Sector Wise Insights



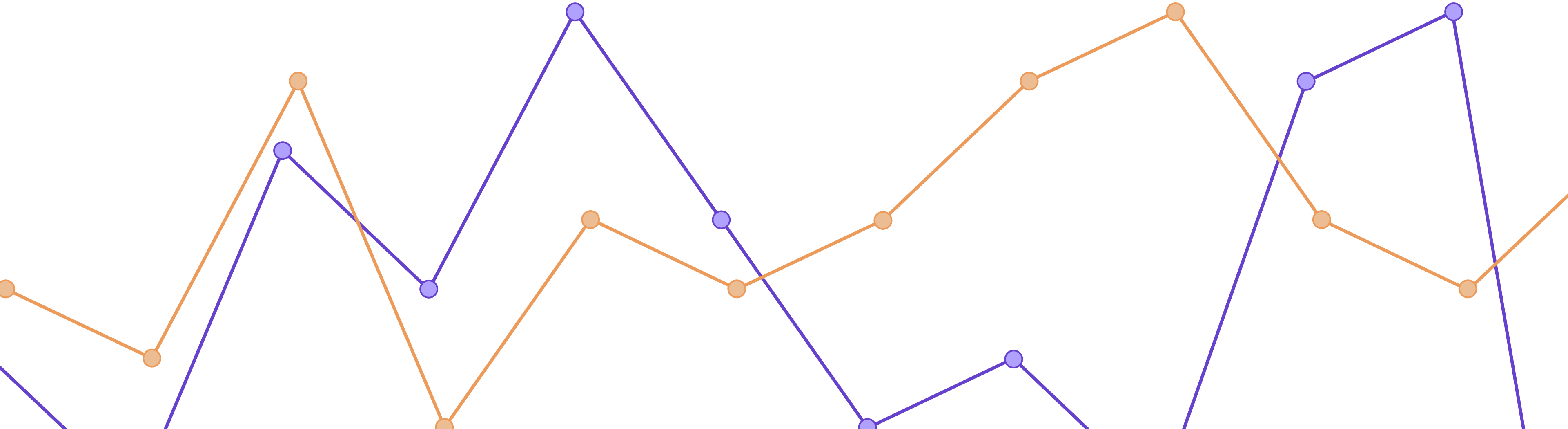



```
plt.figure(figsize=(12, 12))
wedges, texts, autotexts = plt.pie(data.values, labels=data.index, colors=colors, autopct=l

# Add a slight shadow effect
for w in wedges:
    w.set_edgecolor('black')
    w.set_linewidth(3)
    w.set_alpha(0.8)

plt.title('Sectorwise Distribution', fontweight='bold', fontsize=16, loc='center', y=0.50)
plt.axis('equal')

# Customize text properties
for text in texts:
    text.set_fontsize(12)
for autotext in autotexts:
    autotext.set_color('black')
    autotext.set_fontweight('bold')
    autotext.set_fontsize(12)
```



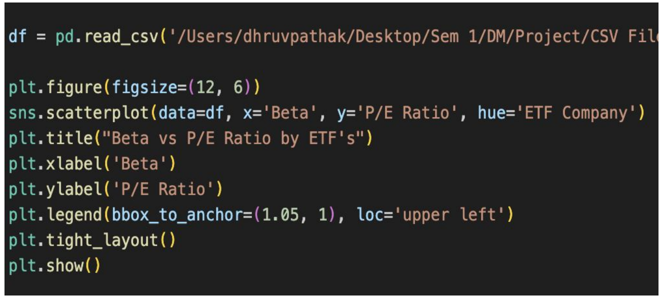

```
ax = grouped_data.plot(kind='bar', stacked=True, figsize=(12, 6))

plt.title('Total Assets by Sector and ETF Company')
plt.xlabel('Sector')
plt.ylabel('Total Assets ($)')
plt.legend(title='ETF Company', bbox_to_anchor=(1.05, 1), loc='upper left')
plt.tight_layout()
```

ETF Risk Detection

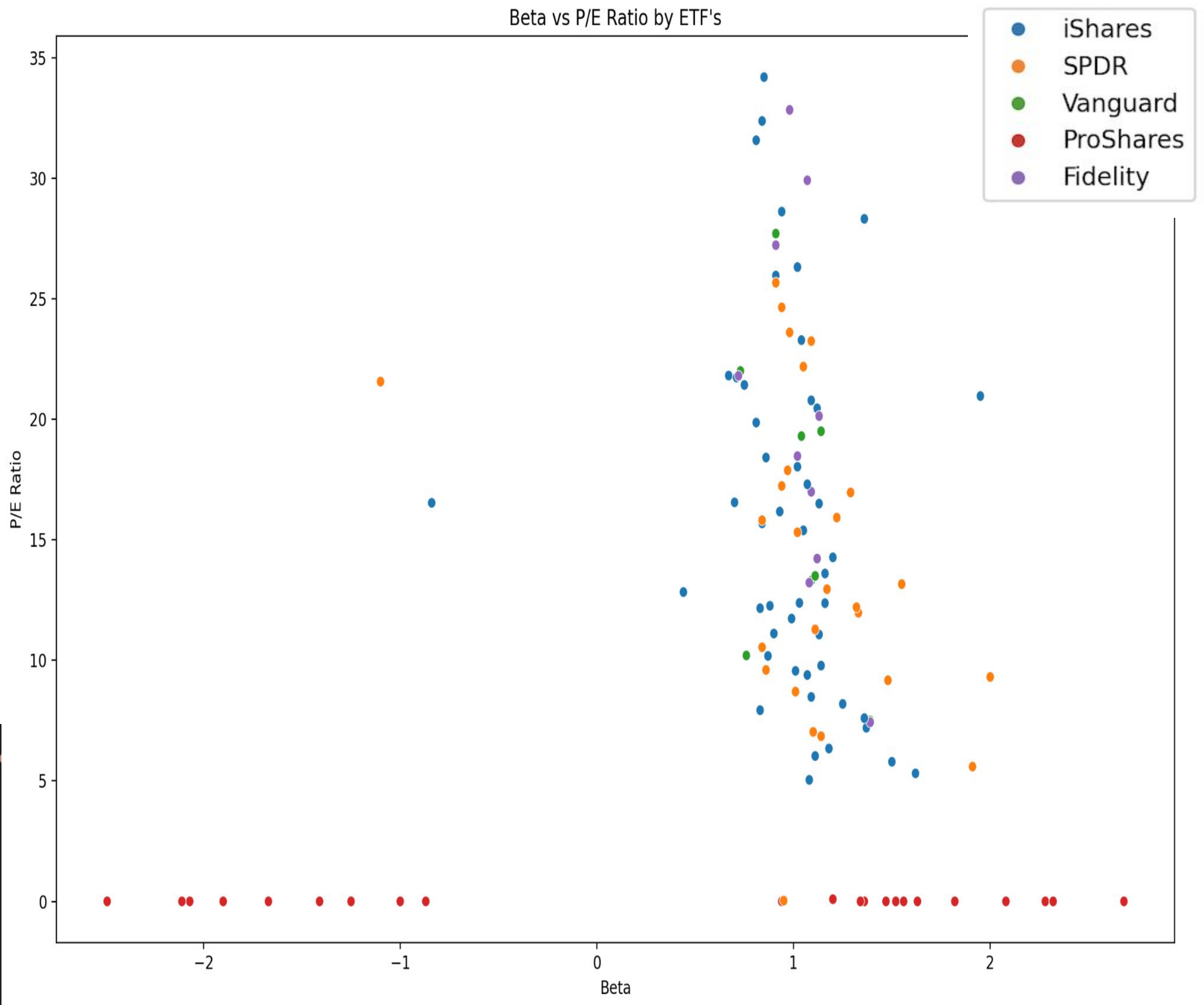


P/E and Beta based Risk Detection

ETF	Risk Profile	Volatility
ProShares	Very High Risk	High
SPDR	High Risk	Moderate
iShares	Moderate Risk	Moderate
Fidelity	Low Risk	Low
Vanguard	Low Risk	Low



```
df = pd.read_csv('/Users/dhruvpathak/Desktop/Sem 1/DM/Project/CSV Fil

plt.figure(figsize=(12, 6))
sns.scatterplot(data=df, x='Beta', y='P/E Ratio', hue='ETF Company')
plt.title("Beta vs P/E Ratio by ETF's")
plt.xlabel('Beta')
plt.ylabel('P/E Ratio')
plt.legend(bbox_to_anchor=(1.05, 1), loc='upper left')
plt.tight_layout()
plt.show()
```



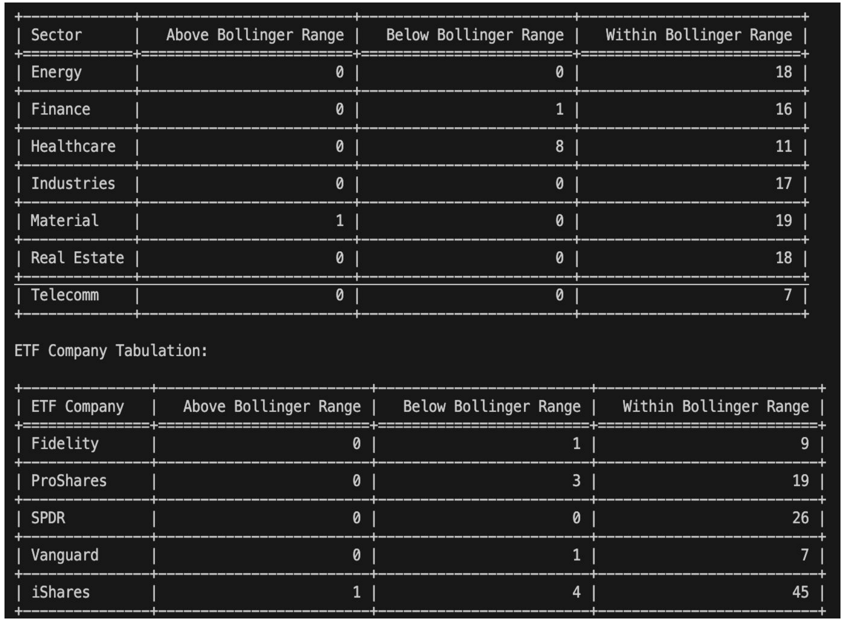
Bollinger Range Risk Detection

ETFs:

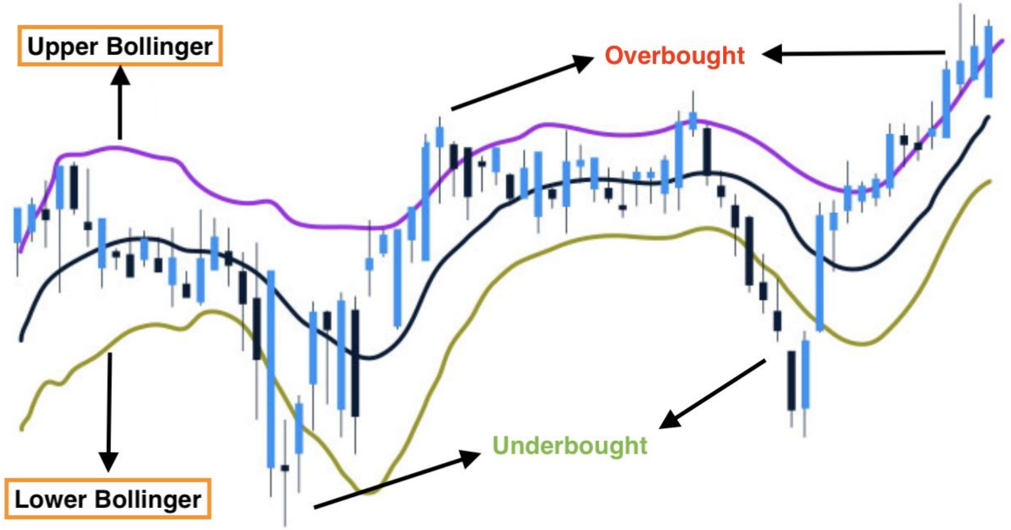
- **iShare** and **ProShares** have <u>higher potential</u>

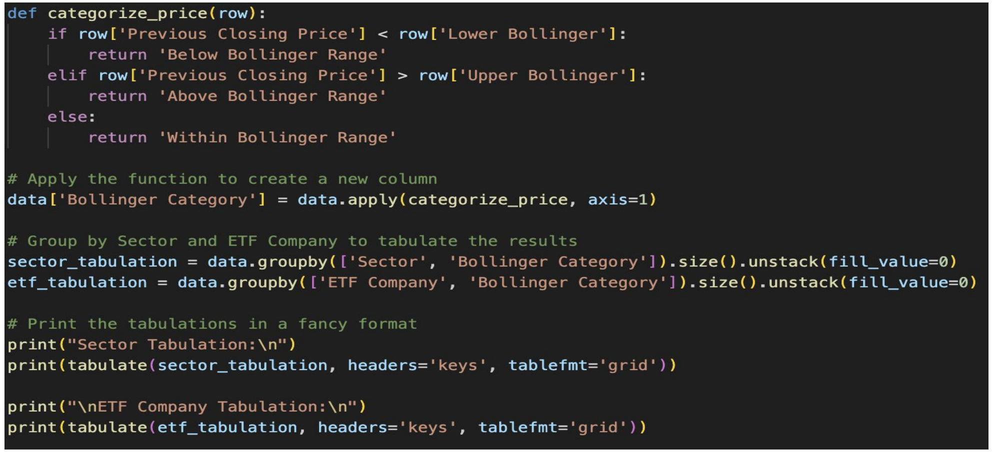
Sectors:

- **Healthcare** depicts <u>high potential.</u>
- **Material** sector on the contrary is deemed <u>high risk.</u>



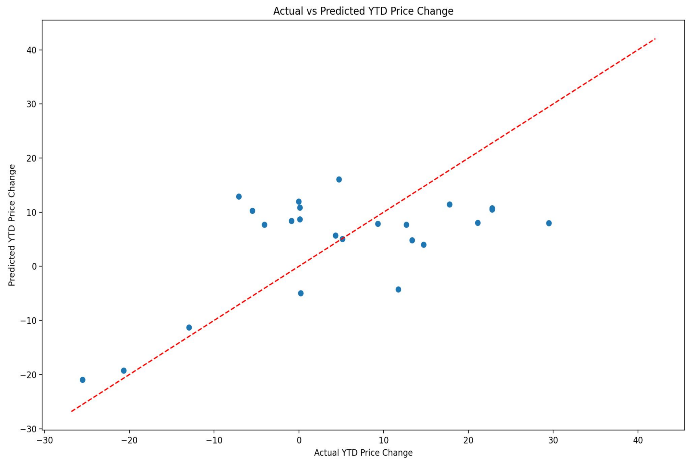
```
+-------------+-------------------------+-------------------------+--------------------------+
| Sector      |   Above Bollinger Range |   Below Bollinger Range |   Within Bollinger Range |
+=============+=========================+=========================+==========================+
| Energy      |                       0 |                       0 |                       18 |
+-------------+-------------------------+-------------------------+--------------------------+
| Finance     |                       0 |                       1 |                       16 |
+-------------+-------------------------+-------------------------+--------------------------+
| Healthcare  |                       0 |                       8 |                       11 |
+-------------+-------------------------+-------------------------+--------------------------+
| Industries  |                       0 |                       0 |                       17 |
+-------------+-------------------------+-------------------------+--------------------------+
| Material    |                       1 |                       0 |                       19 |
+-------------+-------------------------+-------------------------+--------------------------+
| Real Estate |                       0 |                       0 |                       18 |
+-------------+-------------------------+-------------------------+--------------------------+
| Telecomm    |                       0 |                       0 |                        7 |
+-------------+-------------------------+-------------------------+--------------------------+

ETF Company Tabulation:

+---------------+-------------------------+-------------------------+--------------------------+
| ETF Company   |   Above Bollinger Range |   Below Bollinger Range |   Within Bollinger Range |
+===============+=========================+=========================+==========================+
| Fidelity      |                       0 |                       1 |                        9 |
+---------------+-------------------------+-------------------------+--------------------------+
| ProShares     |                       0 |                       3 |                       19 |
+---------------+-------------------------+-------------------------+--------------------------+
| SPDR          |                       0 |                       0 |                       26 |
+---------------+-------------------------+-------------------------+--------------------------+
| Vanguard      |                       0 |                       1 |                        7 |
+---------------+-------------------------+-------------------------+--------------------------+
| iShares       |                       1 |                       4 |                       45 |
+---------------+-------------------------+-------------------------+--------------------------+
```



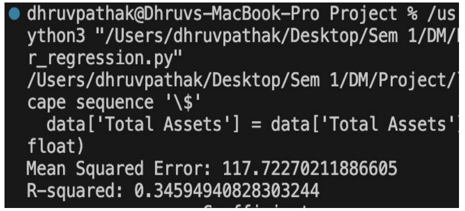


```python
def categorize_price(row):
    if row['Previous Closing Price'] < row['Lower Bollinger']:
        return 'Below Bollinger Range'
    elif row['Previous Closing Price'] > row['Upper Bollinger']:
        return 'Above Bollinger Range'
    else:
        return 'Within Bollinger Range'

# Apply the function to create a new column
data['Bollinger Category'] = data.apply(categorize_price, axis=1)

# Group by Sector and ETF Company to tabulate the results
sector_tabulation = data.groupby(['Sector', 'Bollinger Category']).size().unstack(fill_value=0)
etf_tabulation = data.groupby(['ETF Company', 'Bollinger Category']).size().unstack(fill_value=0)

# Print the tabulations in a fancy format
print("Sector Tabulation:\n")
print(tabulate(sector_tabulation, headers='keys', tablefmt='grid'))

print("\nETF Company Tabulation:\n")
print(tabulate(etf_tabulation, headers='keys', tablefmt='grid'))
```

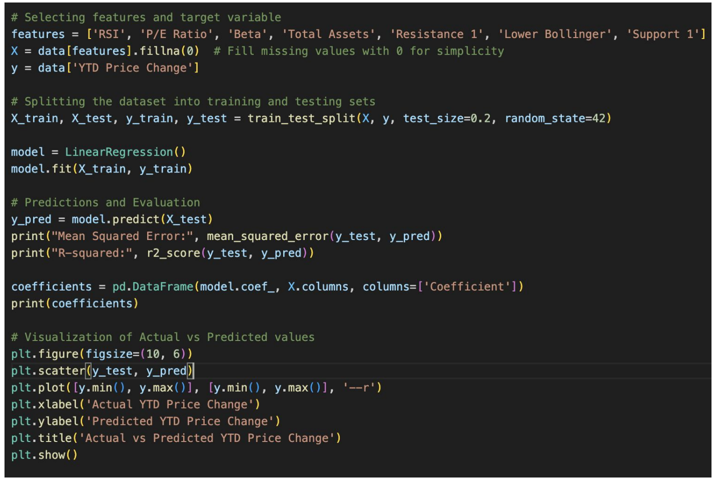
Predictive Model Training



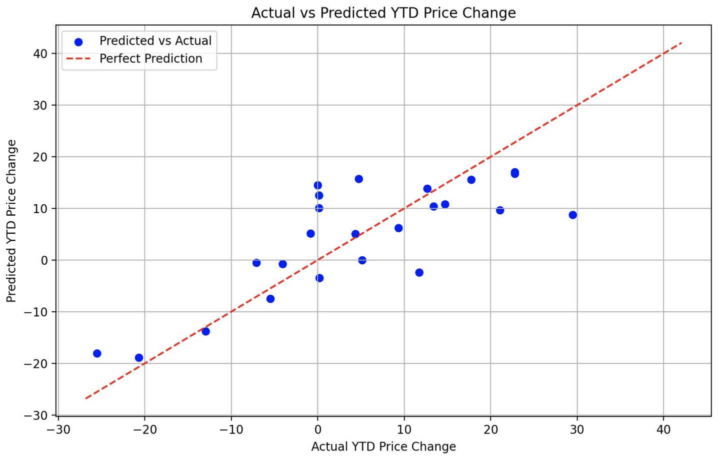
Linear Regression



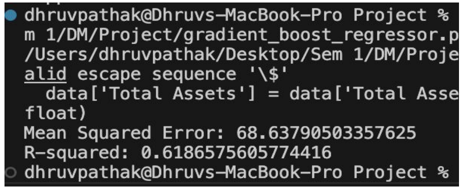
```
dhruvpathak@Dhruvs-MacBook-Pro Project % /us
ython3 "/Users/dhruvpathak/Desktop/Sem 1/DM/
r_regression.py"
/Users/dhruvpathak/Desktop/Sem 1/DM/Project/
cape sequence '\$'
  data['Total Assets'] = data['Total Assets'
float)
Mean Squared Error: 117.72270211886605
R-squared: 0.34594940828303244
```



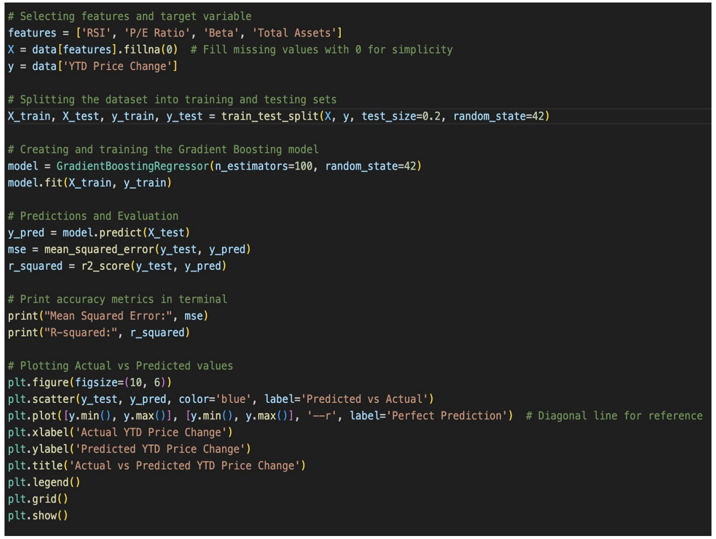
```python
# Selecting features and target variable
features = ['RSI', 'P/E Ratio', 'Beta', 'Total Assets', 'Resistance 1', 'Lower Bollinger', 'Support 1']
X = data[features].fillna(0)  # Fill missing values with 0 for simplicity
y = data['YTD Price Change']

# Splitting the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

model = LinearRegression()
model.fit(X_train, y_train)

# Predictions and Evaluation
y_pred = model.predict(X_test)
print("Mean Squared Error:", mean_squared_error(y_test, y_pred))
print("R-squared:", r2_score(y_test, y_pred))

coefficients = pd.DataFrame(model.coef_, X.columns, columns=['Coefficient'])
print(coefficients)

# Visualization of Actual vs Predicted values
plt.figure(figsize=(10, 6))
plt.scatter(y_test, y_pred)
plt.plot([y.min(), y.max()], [y.min(), y.max()], '--r')
plt.xlabel('Actual YTD Price Change')
plt.ylabel('Predicted YTD Price Change')
plt.title('Actual vs Predicted YTD Price Change')
plt.show()
```



Gradient Boosting Regression

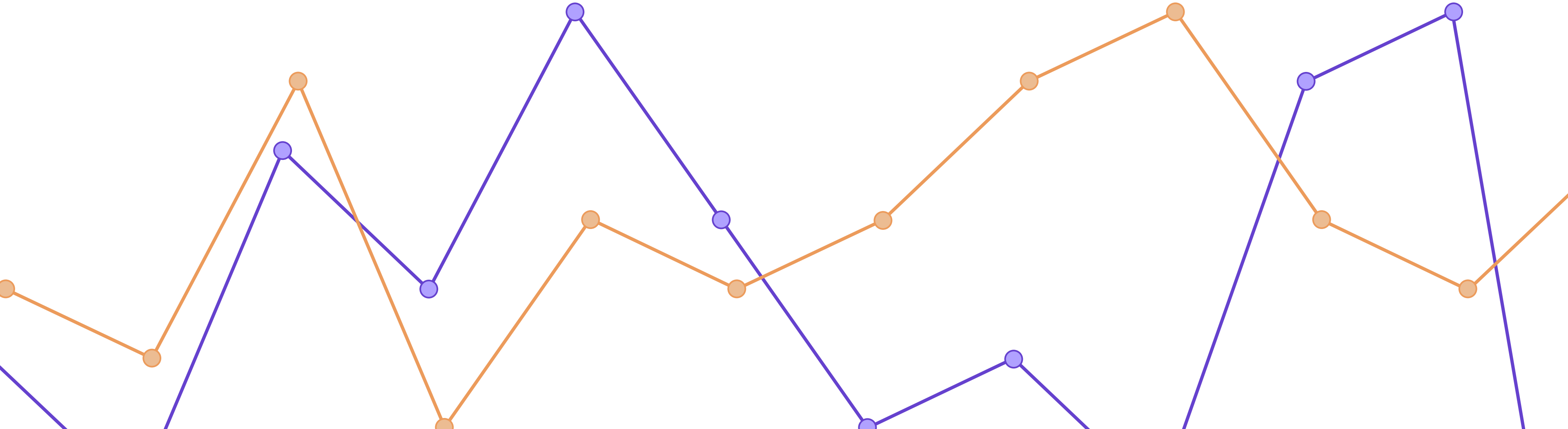


```
dhruvpathak@Dhruvs-MacBook-Pro Project %
m 1/DM/Project/gradient_boost_regressor.p
/Users/dhruvpathak/Desktop/Sem 1/DM/Proje
alid escape sequence '\$'
  data['Total Assets'] = data['Total Asse
float)
Mean Squared Error: 68.63790503357625
R-squared: 0.6186575605774416
dhruvpathak@Dhruvs-MacBook-Pro Project %
```

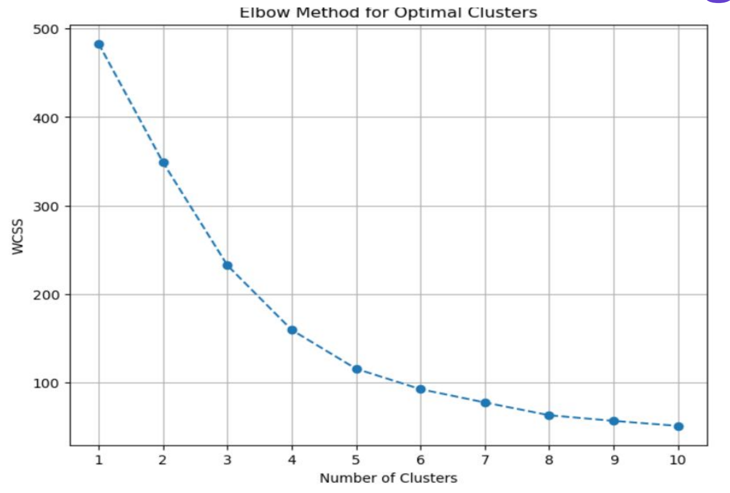


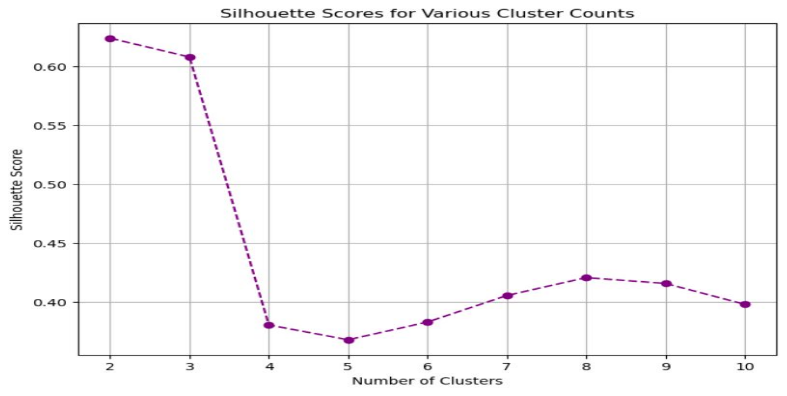
```python
# Selecting features and target variable
features = ['RSI', 'P/E Ratio', 'Beta', 'Total Assets']
X = data[features].fillna(0)  # Fill missing values with 0 for simplicity
y = data['YTD Price Change']

# Splitting the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Creating and training the Gradient Boosting model
model = GradientBoostingRegressor(n_estimators=100, random_state=42)
model.fit(X_train, y_train)

# Predictions and Evaluation
y_pred = model.predict(X_test)
mse = mean_squared_error(y_test, y_pred)
r_squared = r2_score(y_test, y_pred)

# Print accuracy metrics in terminal
print("Mean Squared Error:", mse)
print("R-squared:", r_squared)

# Plotting Actual vs Predicted values
plt.figure(figsize=(10, 6))
plt.scatter(y_test, y_pred, color='blue', label='Predicted vs Actual')
plt.plot([y.min(), y.max()], [y.min(), y.max()], '--r', label='Perfect Prediction')  # Diagonal line for reference
plt.xlabel('Actual YTD Price Change')
plt.ylabel('Predicted YTD Price Change')
plt.title('Actual vs Predicted YTD Price Change')
plt.legend()
plt.grid()
plt.show()
```

Analysing Large Cap



Large Cap Blend





```
Optimal number of clusters based on silhouette analysis: 2
```

Analysis

- Identified and grouped ETFs with similar financial characteristics.

- We performed clustering to segment ETFs into groups based on risk-return characteristics and fund size, enabling investors to identify suitable investment options.

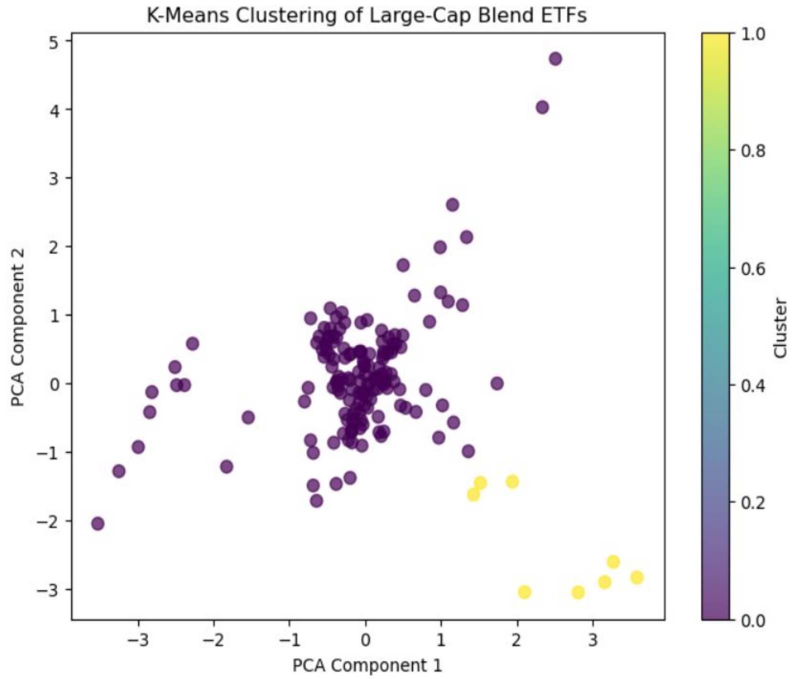
- Used the Elbow Method to determine the optimal number of clusters.

- Silhouette Analysis: Measured cluster cohesion and separation to confirm the optimal cluster count. The silhouette score of 0.62 indicates a reasonably good clustering structure

- Dimensionality Reduction: Applied PCA (Principal Component Analysis) to reduce data dimensions for visualization. **The explained variance ratios (0.36 and 0.34) show that the first two components retained most of the data's variance**

Large Cap Blend

```
Optimal number of clusters based on silhouette analysis: 2
```



```
Silhouette Score: 0.6242139151185025
Explained Variance Ratio (PCA): [0.36303226 0.34026153]
```

Insights

Cluster Visualization:
The scatter plot shows two distinct clusters, indicating ETFs with different risk-return profiles and fund sizes.

- Cluster 0 (Purple): Represents ETFs with smaller sizes or moderate returns and risk profiles.
- Cluster 1 (Yellow): Indicates a subset of ETFs with higher fund sizes or higher performance and risk.

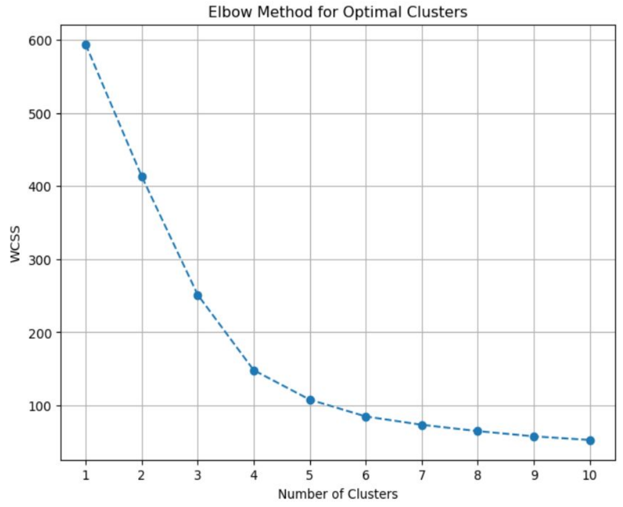
Silhouette and Elbow Method:

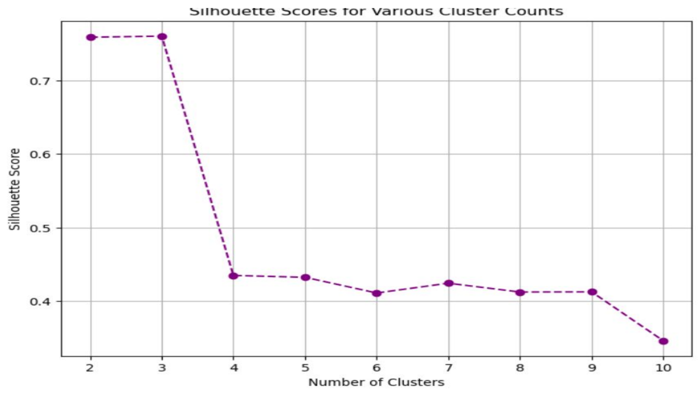
- The **silhouette score (0.62)** confirms that the clustering is reasonably compact and well-separated.
- The elbow plot shows diminishing returns in WCSS reduction beyond 2 clusters, validating the chosen cluster count.

Cluster Summary:
The mean and median values of key metrics (Total Assets, YTD, Beta) across clusters provide actionable insights into group characteristics:

- Cluster 0: Includes conservative funds with lower risk and smaller sizes.
- Cluster 1: Targets aggressive growth strategies or large-cap-focused funds with higher volatility and returns.

Large Cap Growth





```
Optimal number of clusters based on silhouette analysis: 3
```

Analysis

Normalization:

- Standardized the data using StandardScaler to bring all features to the same scale, ensuring that large values like Total Assets do not dominate clustering.

Optimal Clustering (Elbow and Silhouette Methods):

- Used the Elbow Method (plotting WCSS values) to observe diminishing returns as the number of clusters increases. The "elbow point" suggests the optimal number of clusters.
- Verified the optimal clusters with Silhouette Analysis, which evaluates cluster cohesion and separation.

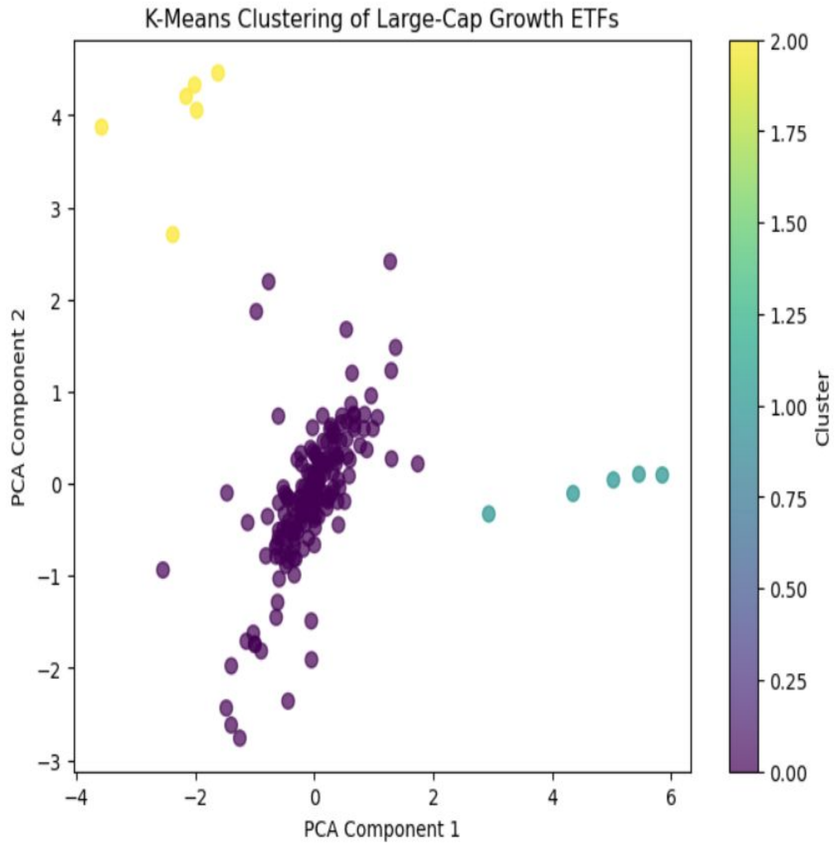
Dimensionality Reduction:

- Applied Principal Component Analysis (PCA) to reduce the data to two components for visualization, while retaining most of the variance.

Visualization:

- Created a PCA scatter plot to visualize clusters in two dimensions, with color-coded points representing cluster membership.

Large Cap Growth



```
Silhouette Score: 0.7604375443466972
Explained Variance Ratio (PCA): [0.35310305 0.34216448]
```

Insights

Optimal Clusters:

- The optimal number of clusters, based on Silhouette Scores, was determined to be **3.**

Cluster Visualization:

- Cluster 0 (purple region): Represents ETFs with moderate risk (Beta), average YTD returns, and substantial but not extremely high Total Assets.
- Cluster 1 (yellow region): Comprises ETFs with higher YTD returns, indicating aggressive growth funds that may also carry higher volatility.
- Cluster 2 (cyan region): Includes ETFs with very high Total Assets but relatively stable performance (Beta and YTD), representing well-established, large-scale funds.

Silhouette Score:

- A **Silhouette Score of 0.76** indicates a good clustering result, with well-separated and cohesive clusters.

PCA Explained Variance:

- The **Explained Variance Ratio of [0.35, 0.34]** implies that the first two PCA components account for approximately 69% of the variability in the data. This means the visualization retains significant information from the original dataset.

Challenges	
1	Gathering datasets on ETFs and Sectors due to its sensitivity.
2	Dealing with high variation in data ands extreme outliers.
3	Dealing with highly technical data and needing a good financial understanding.

Overcoming Challenges	
1	Using paid datasets from different sources.
1	Analyse a larger set of Mutual Funds and to use data from a longer time period.
2	Normalizing data and removing outliers post analysis of the impact of outliers on the dataset.
3	Learn about multiple technical metric and have a strong understanding of the market and external factors.

Future Scope	
1	Increase accuracy by combining models and using larger and more accurate datasets.
2	Implementation of more advanced techniques.
3	Include Risk Adjusted Matrix like Sharpe Ratio
4	Integration with Financial Tools and APIs.
5	Implement a User Interface for better visualization.
6	Factor in external factor to better align the model with the real world.

Project Github Repository

Conclusion

This project provided a comprehensive analysis of mutual funds by leveraging data mining techniques to assess sector-specific risk and performance. Key takeaways include:

1. **Sector-wise Insights**:
   a. Sectors like Healthcare perform better during market downturns, while sectors like Materials and Telecom offer higher variability and growth potential.
   b. Outliers across sectors emphasize the importance of using median values for central tendencies in risk and performance assessment.
2. **ETF Analysis**:
   a. Dominance of key players (e.g., Vanguard, iShares) highlights market concentration, while smaller companies target niche opportunities.
3. **Risk-Return Insights**:
   a. Higher Beta values correlated with increased volatility and potential returns, while Total Assets showed heavy skewness, dominated by large-cap ETFs.
   b. Minimal correlation between features like Total Assets and Beta suggests independent influences on market volatility.

This analysis bridges gaps in mutual fund risk assessment, offering valuable tools for investors to align strategies with risk tolerance and market conditions.

THANK YOU :)