Handling data and performing data cleaning

```
import pandas as pd

#loading the dataset in google colab
dataset_path = '/content/Stock Market-Historical Data of Top 10 Companies.csv'

# Loading the dataset into a pandas DataFrame
df = pd.read_csv(dataset_path)

# Displaying the initial state of the dataset
print("Initial dataset:")
print(df.head())
# Checking for missing values
missing_values = df.isnull().sum()
# Displaying the number of missing values in each column
print("\nMissing values in each column:")
print(missing_values)

# Handle missing values
df_cleaned = df.fillna(df.mean())

# Display the cleaned dataset
print("\nCleaned dataset:")
print(df_cleaned.head())
```

```
Initial dataset:
  Company Close/Last    Volume     Open       High      Low
0    AAPL    $193.99  50520160  $191.90    $194.32  $191.81
1    AAPL    $190.69  41616240  $190.23  $191.1799  $189.63
2    AAPL    $190.54  41342340  $190.50    $191.19  $189.78
3    AAPL    $189.77  60750250  $189.68    $191.70  $188.47
4    AAPL    $188.08  46638120  $189.16    $189.30  $186.60

Missing values in each column:
Company       0
Close/Last    0
Volume        0
Open          0
High          0
Low           0
dtype: int64

Cleaned dataset:
  Company Close/Last    Volume     Open       High      Low
0    AAPL    $193.99  50520160  $191.90    $194.32  $191.81
1    AAPL    $190.69  41616240  $190.23  $191.1799  $189.63
2    AAPL    $190.54  41342340  $190.50    $191.19  $189.78
3    AAPL    $189.77  60750250  $189.68    $191.70  $188.47
4    AAPL    $188.08  46638120  $189.16    $189.30  $186.60
<ipython-input-4-09e8381ceeb4>:19: FutureWarning: The default value of numeric_only in DataFrame.mean is deprecated. In a future ve
  df_cleaned = df.fillna(df.mean())
```

```
import pandas as pd

#loading the excel dataset
dataset_path = '/content/MF_Behavior.xlsx'

# Loading the dataset into a pandas DataFrame
df = pd.read_excel(dataset_path)

# Displaying the initial state of the dataset
print("Initial dataset:")
print(df.head())

# Checking for missing values
missing_values = df.isnull().sum()

# Displaying the number of missing values in each column
print("\nMissing values in each column:")
print(missing_values)

# Handling missing values (for simplicity, filling missing values with the mean)
df_cleaned = df.fillna(df.mean())

# Displaying the cleaned dataset
print("\nCleaned dataset:")
print(df_cleaned.head())
```

```
0           1         1       0    1       1          1               2
1           2         2       0    1       1          2               1
2           3         8       0    2       1          0               1
3           4         8       0    2       1          1               2
4           5        18       0    3       1          2               0

   Affordability  Liquidity  Growth  Trustworthiness  Technology  Integrity  \
0              6          4       3                3           5          2
1              6          3       4                2           6          2
2              7          5       3                3           3          4
3              6          5       4                4           2          4
4              7          7       3                6           0          6

   BrandValue     AUM
0           2   98062
1           1   99187
2           3  180043
3           2  159982
4           6  459815

Missing values in each column:
Investor_ID       0
Longevity         0
Female            0
Age               0
Income            0
ProfManage        0
Diversification   0
Affordability     0
Liquidity         0
Growth            0
Trustworthiness   0
Technology        0
Integrity         0
BrandValue        0
AUM               0
dtype: int64

Cleaned dataset:
   Investor_ID  Longevity  Female  Age  Income  ProfManage  Diversification  \
0            1          1       0    1       1           1                2
1            2          2       0    1       1           2                1
2            3          8       0    2       1           0                1
3            4          8       0    2       1           1                2
4            5         18       0    3       1           2                0

   Affordability  Liquidity  Growth  Trustworthiness  Technology  Integrity  \
0              6          4       3                3           5          2
1              6          3       4                2           6          2
2              7          5       3                3           3          4
3              6          5       4                4           2          4
4              7          7       3                6           0          6

   BrandValue     AUM
0           2   98062
1           1   99187
2           3  180043
3           2  159982
4           6  459815
```

Reading the columns of the datasets

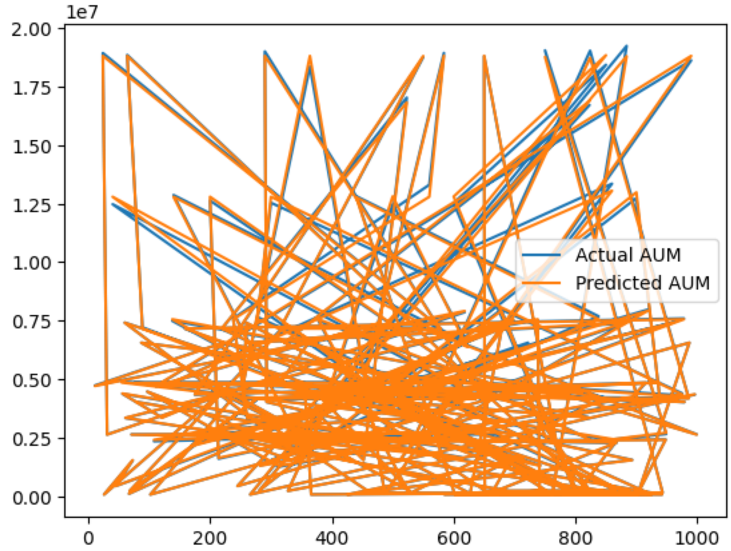
```
import pandas as pd

# Reading the columns of the Stock Market dataset
stock_data = pd.read_csv('/content/Stock Market-Historical Data of Top 10 Companies.csv')
print("Columns of Stock Market dataset:")
print(stock_data.columns)

# Reading the columns of the MF_Behavior dataset
behavior_data = pd.read_excel('/content/MF_Behavior.xlsx')
print("\nColumns of MF_Behavior dataset:")
print(behavior_data.columns)
```

```
Columns of Stock Market dataset:
Index(['Company', 'Close/Last', 'Volume', 'Open', 'High', 'Low'], dtype='object')

Columns of MF_Behavior dataset:
Index(['Investor_ID', 'Longevity', 'Female', 'Age', 'Income', 'ProfManage',
       'Diversification', 'Affordability', 'Liquidity', 'Growth',
       'Trustworthiness', 'Technology', 'Integrity', 'BrandValue', 'AUM'],
      dtype='object')
```

Random forest regression model

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_squared_error
import matplotlib.pyplot as plt

# Load the dataset
df = pd.read_excel('/content/MF_Behavior.xlsx')
df = df.set_index('Investor_ID')
X = df.drop(['AUM'], axis=1)
y = df['AUM']
# Split the data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
# Build and train a random forest regression model
rf_model = RandomForestRegressor(n_estimators=100, random_state=42)
rf_model.fit(X_train, y_train)
# Predict on test data
y_pred = rf_model.predict(X_test)
# Evaluate the model
mse = mean_squared_error(y_test, y_pred)
print(f'Mean Squared Error: {mse}')
# plots
plt.plot(y_test.index, y_test, label='Actual AUM')
plt.plot(y_test.index, y_pred, label='Predicted AUM')
plt.legend()
plt.show()
```

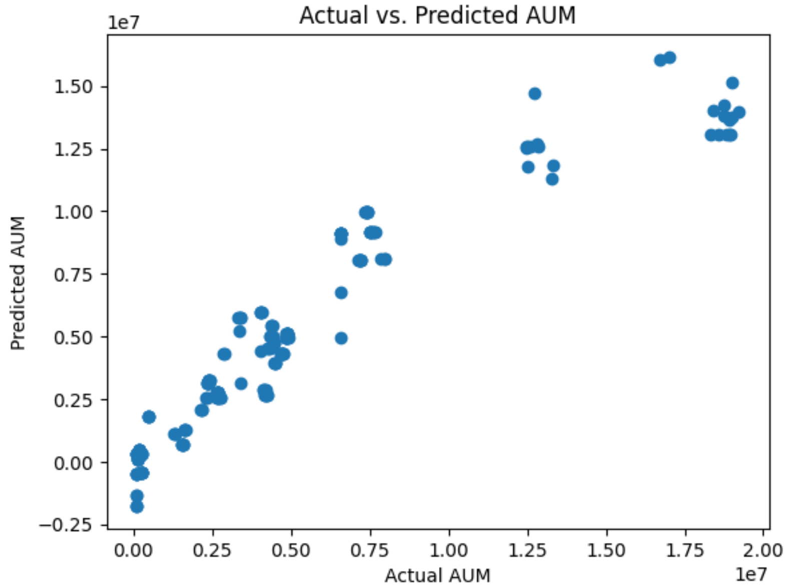
```
Mean Squared Error: 9623771122.450775
```



Regression Model for Correlation Studies

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score
import matplotlib.pyplot as plt
# Load the dataset
df = pd.read_excel('/content/MF_Behavior.xlsx')
df = df.set_index('Investor_ID')
# Assuming you want to predict 'AUM' based on other features
# Replace 'AUM' with the target variable you are interested in
X = df.drop(['AUM'], axis=1)
y = df['AUM']
# 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)

# Initialize the linear regression model
model = LinearRegression()

# Training the model
model.fit(X_train, y_train)

# Predicting on the test set
y_pred = model.predict(X_test)

# Evaluating the model
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)
print(f'Mean Squared Error: {mse}')
print(f'R^2 Score: {r2}')
# Visualization of the predicted vs. actual values
plt.scatter(y_test, y_pred)
plt.xlabel('Actual AUM')
plt.ylabel('Predicted AUM')
plt.title('Actual vs. Predicted AUM')
plt.show()
```

```
Mean Squared Error: 2846936691156.8086
R^2 Score: 0.8761351422647697
```



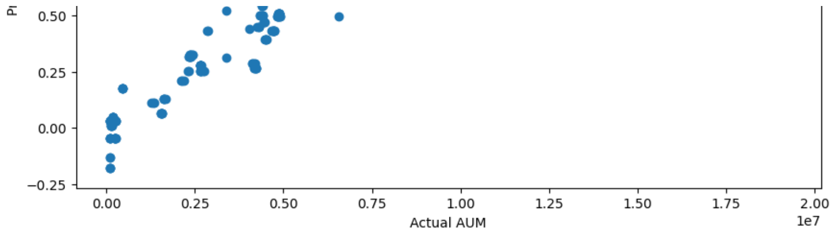
Time series analysis and hypothesis testing

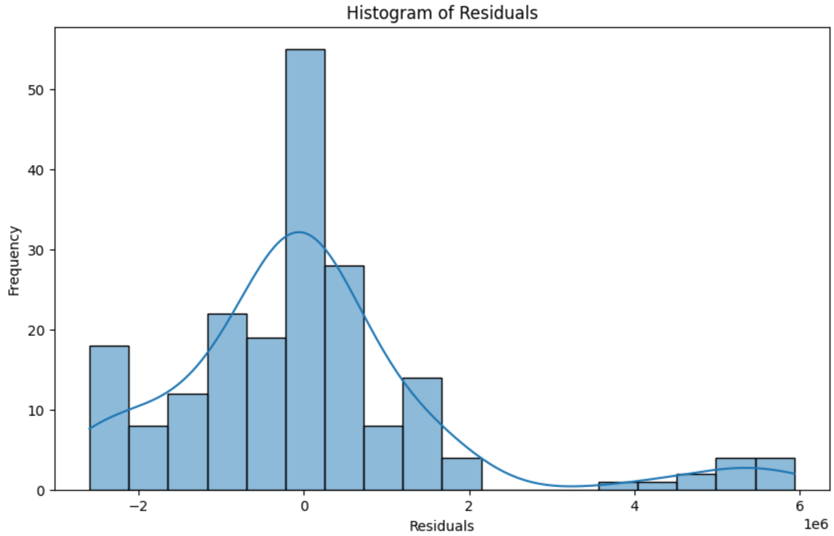
```
import statsmodels.api as sm
# Selecting features and target variable
features = ['Longevity', 'Female', 'Age', 'Income', 'ProfManage', 'Diversification',
            'Affordability', 'Liquidity', 'Growth', 'Trustworthiness', 'Technology',
            'Integrity', 'BrandValue']
target = 'AUM'
X = df[features]
y = df[target]
# Splitting 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)
# Training a linear regression model using scikit-learn
model = LinearRegression()
model.fit(X_train, y_train)
y_pred = model.predict(X_test)
mse = mean_squared_error(y_test, y_pred)
# Displaying model evaluation results
print(f'Mean Squared Error: {mse}')
coefficients = pd.DataFrame({'Feature': features, 'Coefficient': model.coef_})
print(coefficients)
# Hypothesis testing using statsmodels
X_train_sm = sm.add_constant(X_train)
ols_model = sm.OLS(y_train, X_train_sm).fit()
print(ols_model.summary())
```

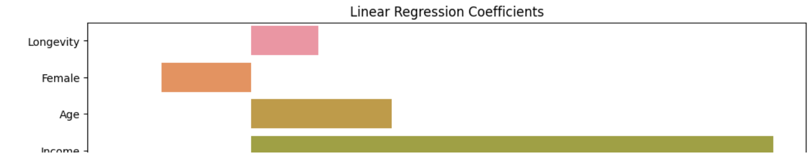
```
Mean Squared Error: 2846936691156.8086
            Feature   Coefficient
0         Longevity  3.323931e+05
1            Female -4.400684e+05
2               Age  6.949502e+05
3            Income  2.574281e+06
4        ProfManage  2.437515e+05
5   Diversification  3.675460e+05
6     Affordability  1.265962e+06
7         Liquidity -6.446150e+05
8            Growth  1.859462e+05
9   Trustworthiness  3.884825e+05
10       Technology  4.306308e+05
11        Integrity  9.718941e+04
12       BrandValue -5.439152e+05
                            OLS Regression Results
==============================================================================
Dep. Variable:                    AUM   R-squared:                       0.889
Model:                            OLS   Adj. R-squared:                  0.887
Method:                 Least Squares   F-statistic:                     483.8
Date:                Sun, 10 Dec 2023   Prob (F-statistic):               0.00
Time:                        22:28:02   Log-Likelihood:                -12424.
No. Observations:                 800   AIC:                         2.488e+04
Df Residuals:                     786   BIC:                         2.494e+04
Df Model:                          13
Covariance Type:            nonrobust
===================================================================================
                      coef    std err          t      P>|t|      [0.025      0.975]
-----------------------------------------------------------------------------------
const           -1.391e+07   9.13e+05    -15.237      0.000   -1.57e+07   -1.21e+07
Longevity        3.324e+05   5.81e+04      5.722      0.000    2.18e+05    4.46e+05
Female          -4.401e+05   1.04e+05     -4.220      0.000   -6.45e+05   -2.35e+05
Age               6.95e+05   5.29e+05      1.315      0.189   -3.43e+05    1.73e+06
Income           2.574e+06    1.7e+05     15.185      0.000    2.24e+06    2.91e+06
ProfManage       2.438e+05   6.07e+04      4.015      0.000    1.25e+05    3.63e+05
Diversification  3.675e+05   5.29e+04      6.945      0.000    2.64e+05    4.71e+05
Affordability    1.266e+06   7.91e+04     16.007      0.000    1.11e+06    1.42e+06
Liquidity       -6.446e+05   4.81e+04    -13.412      0.000   -7.39e+05    -5.5e+05
Growth           1.859e+05   7.33e+04      2.536      0.011     4.2e+04     3.3e+05
Trustworthiness  3.885e+05    8.5e+04      4.571      0.000    2.22e+05    5.55e+05
Technology       4.306e+05   4.34e+04      9.931      0.000    3.46e+05    5.16e+05
Integrity        9.719e+04    5.9e+04      1.647      0.100   -1.86e+04    2.13e+05
BrandValue      -5.439e+05   6.37e+04     -8.536      0.000   -6.69e+05   -4.19e+05
==============================================================================
Omnibus:                      287.067   Durbin-Watson:                   2.036
Prob(Omnibus):                  0.000   Jarque-Bera (JB):             1438.717
Skew:                           1.564   Prob(JB):                         0.00
Kurtosis:                       8.777   Cond. No.                         358.
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
```

Double-click (or enter) to edit

```
import seaborn as sns
plt.figure(figsize=(10, 6))
plt.scatter(y_test, y_pred)
plt.xlabel('Actual AUM')
plt.ylabel('Predicted AUM')
plt.title('Predicted vs. Actual AUM')
plt.show()
# Plotting histogram of residuals
residuals = y_test - y_pred
plt.figure(figsize=(10, 6))
sns.histplot(residuals, kde=True)
plt.xlabel('Residuals')
plt.ylabel('Frequency')
plt.title('Histogram of Residuals')
plt.show()
# Bar chart of coefficients
plt.figure(figsize=(12, 8))
sns.barplot(x='Coefficient', y='Feature', data=coefficients)
plt.title('Linear Regression Coefficients')
plt.xlabel('Coefficient Value')
plt.ylabel('Feature')
plt.show()
```







Support Vector Machines (SVM)


```
from sklearn.svm import SVR
X = df.drop(['AUM'], axis=1)
y = df['AUM']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
model = SVR(kernel='linear')
model.fit(X_train, y_train)
y_pred = model.predict(X_test)
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)
print(f'Mean Squared Error: {mse}')
print(f'R^2 Score: {r2}')
plt.scatter(y_test, y_pred)
plt.xlabel('Actual AUM')
plt.ylabel('Predicted AUM')
plt.title('Actual vs. Predicted AUM (SVM Regression)')
plt.show()
```

```
Mean Squared Error: 25249146232039.78
```

Gaussian Process Regresssion

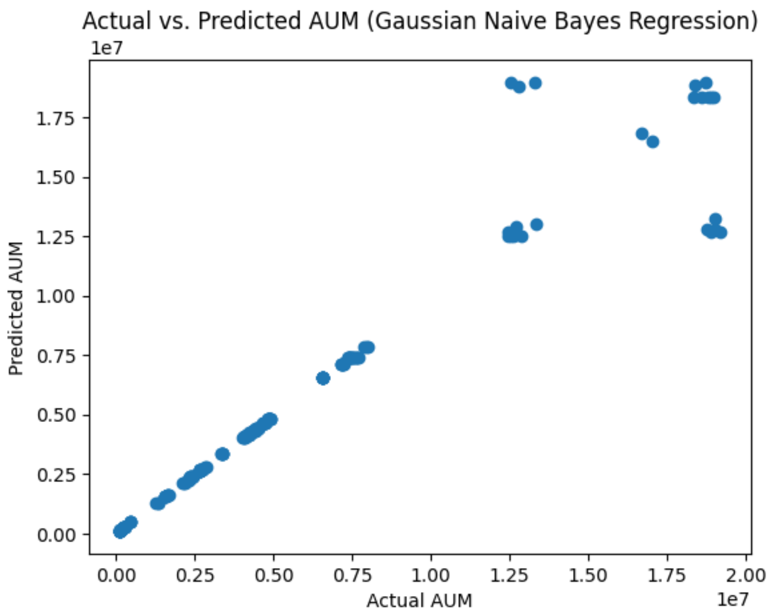
```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.gaussian_process import GaussianProcessRegressor
from sklearn.gaussian_process.kernels import ConstantKernel, RBF
from sklearn.metrics import mean_squared_error
import matplotlib.pyplot as plt


features = ['Longevity', 'Female', 'Age', 'Income', 'ProfManage', 'Diversification',
            'Affordability', 'Liquidity', 'Growth', 'Trustworthiness', 'Technology',
            'Integrity', 'BrandValue']
target = 'AUM'
X = df[features]
y = df[target]

# Splitting 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)

kernel = ConstantKernel(1.0, (1e-3, 1e3)) * RBF(1.0, (1e-2, 1e2))
model = GaussianProcessRegressor(kernel=kernel, n_restarts_optimizer=10, random_state=42)
model.fit(X_train, y_train)
y_pred, sigma = model.predict(X_test, return_std=True)
mse = mean_squared_error(y_test, y_pred)
print(f'Mean Squared Error: {mse}')
print("\nRegression Coefficients:")
for feature, coef in zip(features, model.kernel_.theta[:-1]):
    print(f"{feature}: {coef}")
print("\nFeature Importance:")
feature_importance = np.abs(model.kernel_.theta[:-1])
feature_importance /= feature_importance.sum()
for feature, importance in zip(features, feature_importance):
    print(f"{feature}: {importance}")
plt.figure(figsize=(10, 6))
plt.errorbar(y_test, y_pred, yerr=1.96 * sigma, fmt='o', ecolor='red', markersize=8, alpha=0.7)
plt.plot([min(y_test), max(y_test)], [min(y_test), max(y_test)], '--', color='gray')
plt.xlabel('Actual AUM')
plt.ylabel('Predicted AUM')
plt.title('Actual vs. Predicted AUM (Gaussian Process Regression)')
plt.show()
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/gaussian_process/kernels.py:430: ConvergenceWarning: The optimal value found for di
  warnings.warn(
Mean Squared Error: 24154005972.33
```

Naive Bayes Regresssion

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import mean_squared_error
import matplotlib.pyplot as plt
features = ['Longevity', 'Female', 'Age', 'Income', 'ProfManage', 'Diversification',
            'Affordability', 'Liquidity', 'Growth', 'Trustworthiness', 'Technology',
            'Integrity', 'BrandValue']
target = 'AUM'
X = df[features]
y = df[target]
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
model = GaussianNB()
model.fit(X_train, y_train)
y_pred = model.predict(X_test)
mse = mean_squared_error(y_test, y_pred)
print(f'Mean Squared Error: {mse}')
plt.scatter(y_test, y_pred)
plt.xlabel('Actual AUM')
plt.ylabel('Predicted AUM')
plt.title('Actual vs. Predicted AUM (Gaussian Naive Bayes Regression)')
plt.show()
```

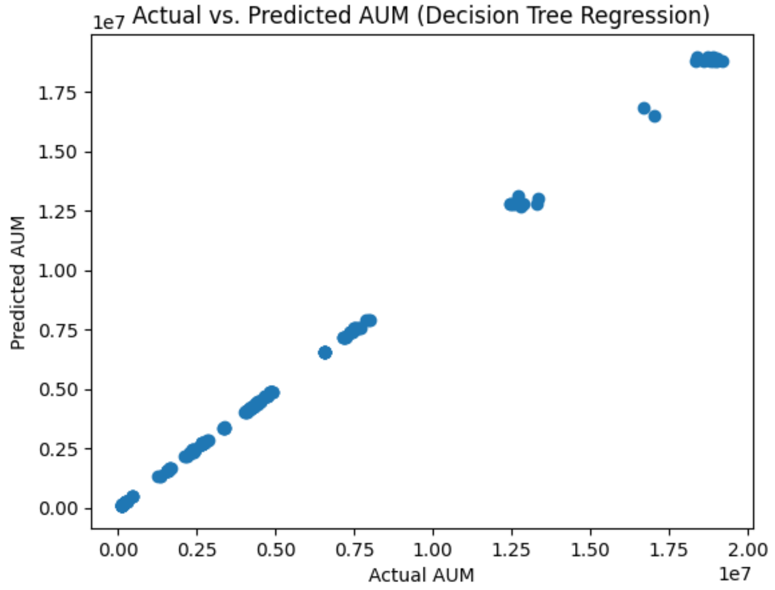
```
Mean Squared Error: 1507089733952.845
```



Decision Tree Regression

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeRegressor  # Import DecisionTreeRegressor
from sklearn.metrics import mean_squared_error
import matplotlib.pyplot as plt
features = ['Longevity', 'Female', 'Age', 'Income', 'ProfManage', 'Diversification',
            'Affordability', 'Liquidity', 'Growth', 'Trustworthiness', 'Technology',
            'Integrity', 'BrandValue']
target = 'AUM'
X = df[features]
y = df[target]
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
model = DecisionTreeRegressor(random_state=42)
model.fit(X_train, y_train)
y_pred = model.predict(X_test)
mse = mean_squared_error(y_test, y_pred)
print(f'Mean Squared Error: {mse}')
plt.scatter(y_test, y_pred)
plt.xlabel('Actual AUM')
plt.ylabel('Predicted AUM')
plt.title('Actual vs. Predicted AUM (Decision Tree Regression)')
plt.show()
```

```
Mean Squared Error: 11419955625.398727
```




```
import pandas as pd
df = pd.read_excel("/content/MF_Behavior.xlsx")
print("Columns in the dataset:")
print(df.columns)
```

```
Columns in the dataset:
Index(['Investor_ID', 'Longevity', 'Female', 'Age', 'Income', 'ProfManage',
       'Diversification', 'Affordability', 'Liquidity', 'Growth',
       'Trustworthiness', 'Technology', 'Integrity', 'BrandValue', 'AUM'],
      dtype='object')
```

MAXIMUM LIKELIHOOD MODEL

```
import pandas as pd
import statsmodels.api as sm
df = pd.read_excel("/content/MF_Behavior.xlsx")
dependent_variable_column = 'Longevity'
independent_variable_columns = ['Female', 'Age', 'Income', 'ProfManage',
                                'Diversification', 'Affordability', 'Liquidity',
                                'Growth', 'Trustworthiness', 'Technology',
                                'Integrity', 'BrandValue', 'AUM']
X = sm.add_constant(df[independent_variable_columns])
model = sm.OLS(df[dependent_variable_column], X)
result = model.fit()

# Display the results
print(result.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:              Longevity   R-squared:                       0.988
```

```
Model:                            OLS   Adj. R-squared:                  0.988
Method:                 Least Squares   F-statistic:                     6224.
Date:                Sun, 10 Dec 2023   Prob (F-statistic):               0.00
Time:                        22:28:26   Log-Likelihood:                -1215.3
No. Observations:                1000   AIC:                             2459.
Df Residuals:                     986   BIC:                             2527.
Df Model:                          13
Covariance Type:            nonrobust
===================================================================================
                      coef    std err          t      P>|t|      [0.025      0.975]
-----------------------------------------------------------------------------------
const              -4.3951      0.540     -8.142      0.000      -5.454      -3.336
Female              0.4206      0.055      7.620      0.000       0.312       0.529
Age                 8.2417      0.111     74.077      0.000       8.023       8.460
Income              1.1380      0.098     11.560      0.000       0.945       1.331
ProfManage          0.0942      0.032      2.951      0.003       0.032       0.157
Diversification    -0.0724      0.029     -2.472      0.014      -0.130      -0.015
Affordability      -0.2586      0.048     -5.347      0.000      -0.353      -0.164
Liquidity          -0.2867      0.027    -10.659      0.000      -0.339      -0.234
Growth             -0.1907      0.038     -5.030      0.000      -0.265      -0.116
Trustworthiness     0.1154      0.046      2.495      0.013       0.025       0.206
Technology         -0.1901      0.024     -8.024      0.000      -0.237      -0.144
Integrity          -0.2651      0.031     -8.470      0.000      -0.327      -0.204
BrandValue          0.0910      0.035      2.615      0.009       0.023       0.159
AUM              1.197e-07   1.79e-08      6.669      0.000    8.45e-08    1.55e-07
==============================================================================
Omnibus:                       39.982   Durbin-Watson:                   2.155
Prob(Omnibus):                  0.000   Jarque-Bera (JB):               33.252
Skew:                           0.370   Prob(JB):                     6.02e-08
Kurtosis:                       2.500   Cond. No.                     1.26e+08
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The condition number is large, 1.26e+08. This might indicate that there are
strong multicollinearity or other numerical problems.
```

```
import pandas as pd
import statsmodels.api as sm
import matplotlib.pyplot as plt
# Load the dataset
df = pd.read_excel("/content/MF_Behavior.xlsx")
# Define the actual column names
dependent_variable_column = 'Longevity'
independent_variable_columns = ['Female', 'Age', 'Income', 'ProfManage',
                                 'Diversification', 'Affordability', 'Liquidity',
                                 'Growth', 'Trustworthiness', 'Technology',
                                 'Integrity', 'BrandValue', 'AUM']
X = sm.add_constant(df[independent_variable_columns])
model = sm.OLS(df[dependent_variable_column], X)
result = model.fit()
plt.scatter(df[dependent_variable_column], result.fittedvalues)
plt.xlabel('Observed Values')
plt.ylabel('Predicted Values')
plt.title('Observed vs. Predicted Values')
plt.show()
plt.scatter(result.fittedvalues, result.resid)
plt.xlabel('Fitted Values')
plt.ylabel('Residuals')
plt.title('Residual Plot')
plt.axhline(y=0, color='r', linestyle='--')  # Add a horizontal line at y=0
plt.show()
```