Original Method

```
import pandas as pd
```

```
df = pd.read_csv("/content/drive/MyDrive/Annually Mutual Fund Returns.csv")
```

```
print(f"📊 The dataset has {df.shape[0]} rows and {df.shape[1]} columns.")
```

```
📊 The dataset has 1646 rows and 14 columns.
```

```
df
```

	Fund	Fund Manager	Category	2015	2016	2017	2018	2019	2020	2021	2022	2023	2024	classification
0	360 ONE Balanced Hybrid Fund Reg(G)	Mayur Patel	HY-EQ	-	-	-	-	-	-	-	-	-	16.06	Hybrid : Equity Oriented
1	360 ONE Dynamic Bond Fund Reg(G)	Milan Mody	DT-DYN	6.34	7.83	7.45	5.16	7.76	8.16	5.61	3.55	6.78	9.55	Debt : Dynamic Bond
2	360 ONE ELSS Tax Saver Nifty 50 Index Fund Reg(G)	Parijat Garg	EQ-ELSS	-	-	-	-	-	-	-	-	20.11	9.5	Equity : Tax Saving (ELSS)
3	360 ONE Flexicap Fund Reg(G)	Mayur Patel	EQ-FLEX	-	-	-	-	-	-	-	-	-	26.8	Equity : Flexi Cap
4	360 ONE Focused Equity Fund Reg(G)	Mayur Patel	EQ-MLC	1.82	9.88	29.95	-6.81	27.31	23.83	36.45	-0.92	29.79	14.75	Equity : Multi Cap
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
1641	WOC Ultra Short Duration Fund Reg(G)	Piyush Baranwal	DT-USD	-	-	-	-	-	3.55	3.99	3.9	6.38	6.89	Debt : Ultra Short Duration

```
df.columns
```

```
Index(['Fund', 'Fund Manager', 'Category', '2015', '2016', '2017', '2018',
       '2019', '2020', '2021', '2022', '2023', '2024', 'classification'],
      dtype='object')
```

```
df.head(10)
```

	Fund	Fund Manager	Category	2015	2016	2017	2018	2019	2020	2021	2022	2023	2024	classification
0	360 ONE Balanced Hybrid Fund Reg(G)	Mayur Patel	HY-EQ	-	-	-	-	-	-	-	-	-	16.06	Hybrid : Equity Oriented
1	360 ONE Dynamic Bond Fund Reg(G)	Milan Mody	DT-DYN	6.34	7.83	7.45	5.16	7.76	8.16	5.61	3.55	6.78	9.55	Debt : Dynamic Bond
2	360 ONE ELSS Tax Saver Nifty 50 Index Fund Reg(G)	Parijat Garg	EQ-ELSS	-	-	-	-	-	-	-	-	20.11	9.5	Equity : Tax Saving (ELSS)
3	360 ONE Flexicap Fund Reg(G)	Mayur Patel	EQ-FLEX	-	-	-	-	-	-	-	-	-	26.8	Equity : Flexi Cap
4	360 ONE Focused Equity Fund Reg(G)	Mayur Patel	EQ-MLC	1.82	9.88	29.95	-6.81	27.31	23.83	36.45	-0.92	29.79	14.75	Equity : Multi Cap
5	360 ONE Liquid Fund Reg(G)	Milan Mody	DT-LIQ	7.79	7.19	6.19	6.84	5.92	3.43	3	4.71	6.91	7.19	Debt : Liquid

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1646 entries, 0 to 1645
Data columns (total 14 columns):
 #   Column          Non-Null Count  Dtype 
---  ------          --------------  ----- 
 0   Fund            1646 non-null   object
 1   Fund Manager    1646 non-null   object
 2   Category        1646 non-null   object
 3   2015            1646 non-null   object
 4   2016            1646 non-null   object
 5   2017            1646 non-null   object
 6   2018            1646 non-null   object
 7   2019            1646 non-null   object
```

```
 8   2020            1646 non-null   object
 9   2021            1646 non-null   object
 10  2022            1646 non-null   object
 11  2023            1646 non-null   object
 12  2024            1646 non-null   object
 13  classification  1646 non-null   object
dtypes: object(14)
memory usage: 180.2+ KB
```

```
df.drop(columns=['classification'], inplace=True)
```

```
return_cols = [str(year) for year in range(2015, 2025)]
df[return_cols] = df[return_cols].apply(pd.to_numeric, errors='coerce')
```

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1646 entries, 0 to 1645
Data columns (total 13 columns):
 #   Column        Non-Null Count  Dtype
---  ------        --------------  -----
 0   Fund          1646 non-null   object
 1   Fund Manager  1646 non-null   object
 2   Category      1646 non-null   object
 3   2015          629 non-null    float64
 4   2016          675 non-null    float64
 5   2017          695 non-null    float64
 6   2018          718 non-null    float64
 7   2019          779 non-null    float64
 8   2020          880 non-null    float64
 9   2021          950 non-null    float64
 10  2022          1080 non-null   float64
 11  2023          1271 non-null   float64
 12  2024          1437 non-null   float64
dtypes: float64(10), object(3)
memory usage: 167.3+ KB
```

```
df.head(10)
```

	Fund	Fund Manager	Category	2015	2016	2017	2018	2019	2020	2021	2022	2023	2024
0	360 ONE Balanced Hybrid Fund Reg(G)	Mayur Patel	HY-EQ	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	16.06
1	360 ONE Dynamic Bond Fund Reg(G)	Milan Mody	DT-DYN	6.34	7.83	7.45	5.16	7.76	8.16	5.61	3.55	6.78	9.55
2	360 ONE ELSS Tax Saver Nifty 50 Index Fund Reg(G)	Parijat Garg	EQ-ELSS	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	20.11	9.50
3	360 ONE Flexicap Fund Reg(G)	Mayur Patel	EQ-FLEX	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	26.80
4	360 ONE Focused Equity Fund Reg(G)	Mayur Patel	EQ-MLC	1.82	9.88	29.95	-6.81	27.31	23.83	36.45	-0.92	29.79	14.75
5	360 ONE Liquid Fund Reg(G)	Milan Mody	DT-LIQ	7.79	7.19	6.19	6.84	5.92	3.43	3.00	4.71	6.91	7.19
6	360 ONE Quant Fund Reg(G)	Parijat Garg	EQ-THEM	NaN	NaN	NaN	NaN	NaN	NaN	NaN	3.01	44.78	21.69
7	Aditya Birla SL Active Debt Multi Mgr FoF(G)	Kaustubh Gupta	FOF-DOM	5.28	14.71	2.53	6.31	8.19	9.01	4.57	3.00	6.83	7.58
8	Aditya Birla SL Arbitrage Fund(G)	Lovelish Solanki	HY-ARB	7.60	6.50	5.61	6.02	6.17	4.12	3.84	4.07	7.13	7.52

```
df.isnull().sum()
```

	0
Fund	0
Fund Manager	0
Category	0
2015	1017
2016	971
2017	951
2018	928
2019	867
2020	766
2021	696
2022	566
2023	375
2024	209

dtype: int64

```
df.drop(columns=['2015', '2016', '2017', '2018'], inplace=True)
```

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1646 entries, 0 to 1645
Data columns (total 9 columns):
 #   Column        Non-Null Count  Dtype  
---  ------        --------------  -----  
 0   Fund          1646 non-null   object 
 1   Fund Manager  1646 non-null   object 
 2   Category      1646 non-null   object 
 3   2019          779 non-null    float64
 4   2020          880 non-null    float64
 5   2021          950 non-null    float64
 6   2022          1080 non-null   float64
 7   2023          1271 non-null   float64
 8   2024          1437 non-null   float64
dtypes: float64(6), object(3)
memory usage: 115.9+ KB
```

```
df.isnull().sum()
```

	0
Fund	0
Fund Manager	0
Category	0
2019	867
2020	766
2021	696
2022	566
2023	375
2024	209

dtype: int64

```
df.columns
```

```
Index(['Fund', 'Fund Manager', 'Category', '2019', '2020', '2021', '2022',
       '2023', '2024'],
      dtype='object')
```

```
df.describe()
```

Show hidden output

```
# Fill missing values with median for numeric year-based attributes
df = df.fillna({
    "2019": df["2019"].median(),
    "2020": df["2020"].median(),
    "2021": df["2021"].median(),
    "2022": df["2022"].median(),
    "2023": df["2023"].median(),
    "2024": df["2024"].median()
})
```

```
df.head()
```

	Fund	Fund Manager	Category	2019	2020	2021	2022	2023	2024
0	360 ONE Balanced Hybrid Fund Reg(G)	Mayur Patel	HY-EQ	8.22	11.79	17.445	3.53	19.95	16.06
1	360 ONE Dynamic Bond Fund Reg(G)	Milan Mody	DT-DYN	7.76	8.16	5.610	3.55	6.78	9.55
2	360 ONE ELSS Tax Saver Nifty 50 Index Fund Reg(G)	Parijat Garg	EQ-ELSS	8.22	11.79	17.445	3.53	20.11	9.50
3	360 ONE Flexicap Fund Reg(G)	Mayur Patel	EQ-FLEX	8.22	11.79	17.445	3.53	19.95	26.80
4	360 ONE Focused Equity Fund Reg(G)	Mayur Patel	EQ-MLC	27.31	23.83	36.450	-0.92	29.79	14.75

```
df.describe()
```

Show hidden output

```
df.isnull().sum()
```

	0
Fund	0
Fund Manager	0
Category	0
2019	0
2020	0
2021	0
2022	0
2023	0
2024	0

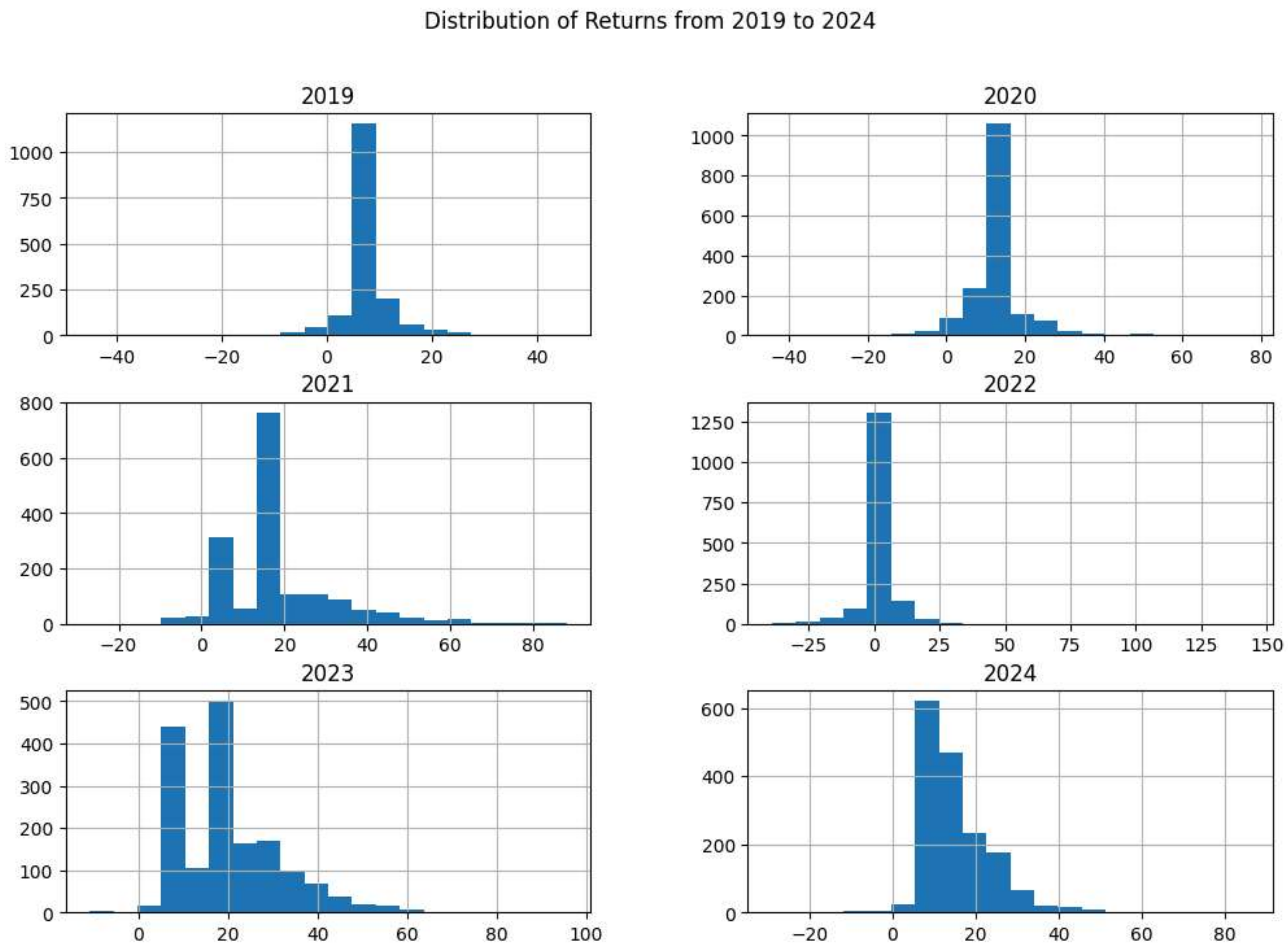
dtype: int64

```
#print("🔍 Number of zeros in y_test_5y:", np.sum(y_test_5y == 0))
```

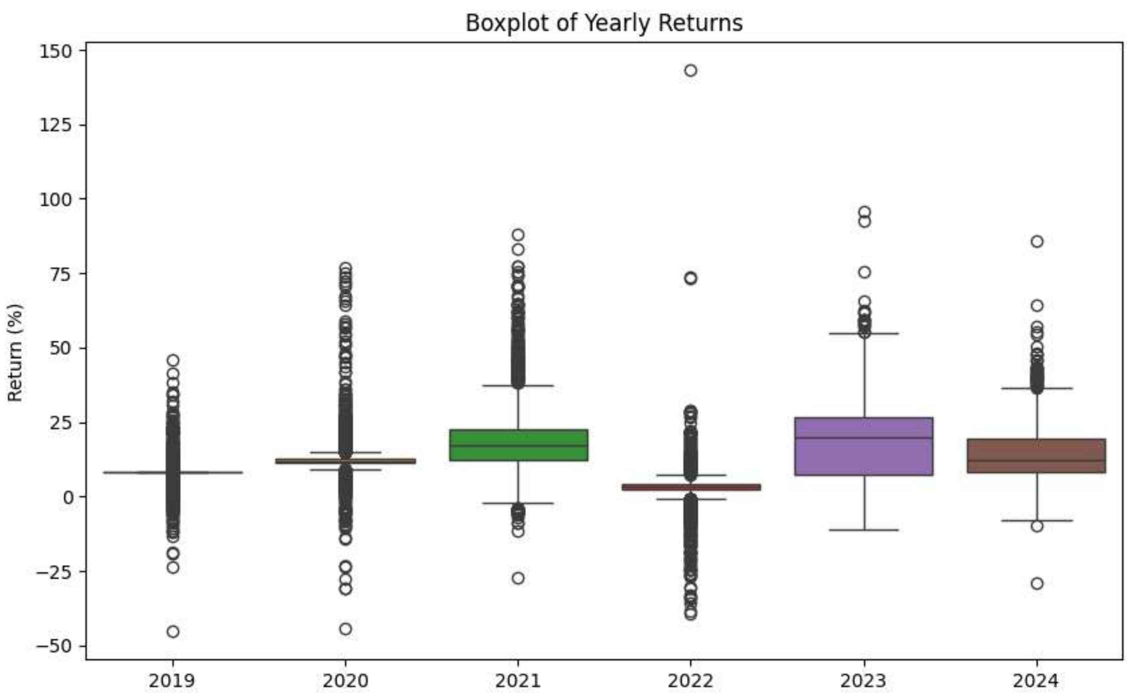
```
#print("🔍 Number of near-zero values (< 0.1) in y_test_5y:", np.sum(y_test_5y < 0.1))
```

```
#print("📉 Smallest 10 values in y_test_5y:")
#rint(np.sort(y_test_5y)[:10])
```

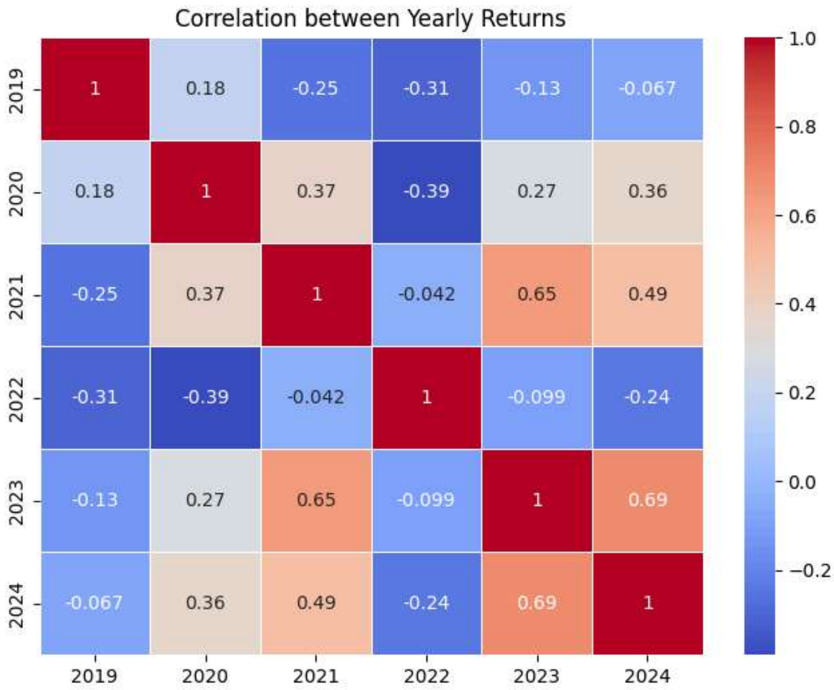
```
import matplotlib.pyplot as plt
import seaborn as sns

# Histogram for each year's returns
year_cols = ["2019", "2020", "2021", "2022", "2023", "2024"]
df[year_cols].hist(bins=20, figsize=(12, 8))
plt.suptitle("Distribution of Returns from 2019 to 2024")
plt.show()
```




```
plt.figure(figsize=(10, 6))
sns.boxplot(data=df[year_cols])
plt.title("Boxplot of Yearly Returns")
plt.ylabel("Return (%)")
plt.show()
```



```
plt.figure(figsize=(8, 6))
sns.heatmap(df[year_cols].corr(), annot=True, cmap='coolwarm', linewidths=0.5)
plt.title("Correlation between Yearly Returns")
plt.show()
```



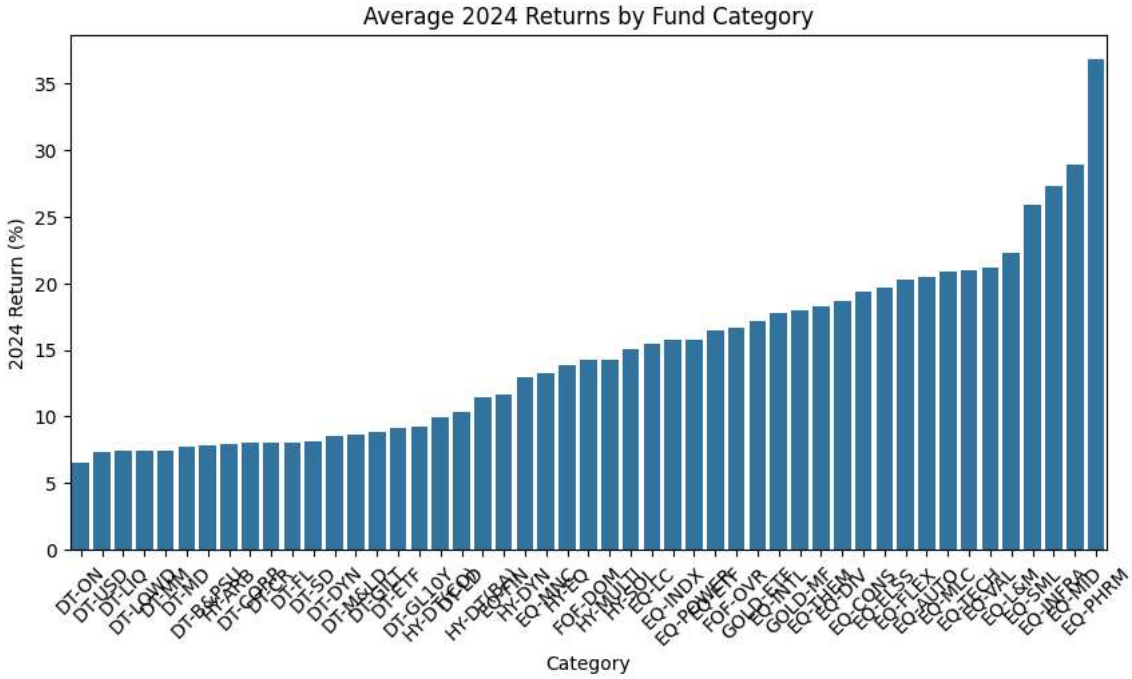

```
# Fund category count
print(df['Category'].value_counts())
```

Show hidden output

```
# Average returns by category
avg_returns = df.groupby('Category')[year_cols].mean()
print(avg_returns)
```

Show hidden output

```
# Visualizing average return by category for 2024
avg_returns_2024 = df.groupby('Category')["2024"].mean().sort_values()


plt.figure(figsize=(10, 5))
sns.barplot(x=avg_returns_2024.index, y=avg_returns_2024.values)
plt.title("Average 2024 Returns by Fund Category")
plt.ylabel("2024 Return (%)")
plt.xticks(rotation=45)
plt.show()
```




```
import pandas as pd
import numpy as np
from sklearn.ensemble import RandomForestRegressor
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
from sklearn.metrics import mean_absolute_error, r2_score, mean_squared_error
```

Start coding or generate with AI.

```
# Step 1: Feature Engineering
df["Fund_Encoded"] = LabelEncoder().fit_transform(df["Fund"])
df["Category_Encoded"] = LabelEncoder().fit_transform(df["Category"])
```

```
# Step 2: Create simulated target using recent years to mock future trends
df["Future_3Y_Return"] = df[["2022", "2023", "2024"]].mean(axis=1) + np.random.normal(0, 0.5, len(df))
df["Future_5Y_Return"] = df[["2020", "2021", "2022", "2023", "2024"]].mean(axis=1) + np.random.normal(0, 0.7, len(df))
```

```
# Step 3: Define features and targets
features = ["2019", "2020", "2021", "2022", "2023", "2024", "Fund_Encoded", "Category_Encoded"]
target_3y = "Future_3Y_Return"
target_5y = "Future_5Y_Return"
```

```
# Step : Train-Test Split
from sklearn.model_selection import train_test_split
X_train, X_test, y_train_3y, y_test_3y = train_test_split(df[features], df[target_3y], test_size=0.2, random_state=42)
_, _, y_train_5y, y_test_5y = train_test_split(df[features], df[target_5y], test_size=0.2, random_state=42)
```

```
# Step : Model Training
model_3y = RandomForestRegressor(n_estimators=200, max_depth=10, random_state=42)
model_5y = RandomForestRegressor(n_estimators=200, max_depth=10, random_state=42)
model_3y.fit(X_train, y_train_3y)
model_5y.fit(X_train, y_train_5y)
```

```
RandomForestRegressor
RandomForestRegressor(max_depth=10, n_estimators=200, random_state=42)
```

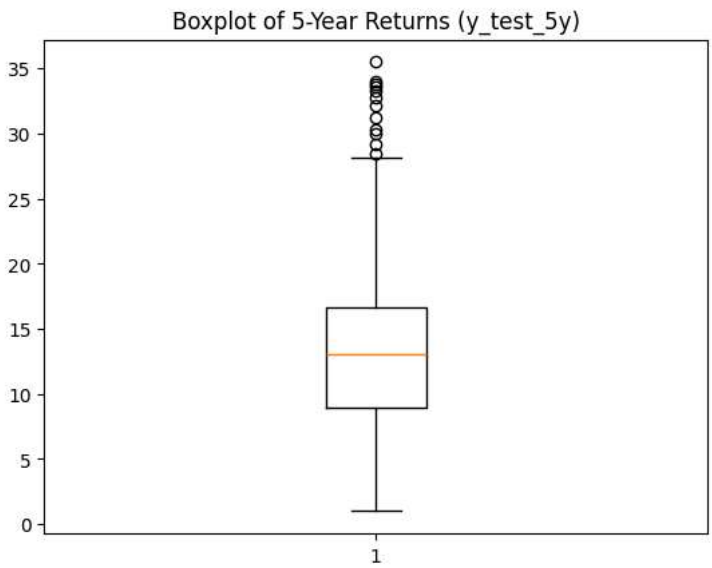
```
# Step : Evaluation
y_pred_3y = model_3y.predict(X_test)
y_pred_5y = model_5y.predict(X_test)
```

```
# Define MAPE function
def mean_absolute_percentage_error(y_true, y_pred):
    y_true, y_pred = np.array(y_true), np.array(y_pred)
    return np.mean(np.abs((y_true - y_pred) / np.clip(np.abs(y_true), 1e-8, None))) * 100

# --- 3-Year Evaluation ---
mse_3y = mean_squared_error(y_test_3y, y_pred_3y)
```

```python
mse_3y = mean_squared_error(y_test_3y, y_pred_3y)
rmse_3y = np.sqrt(mse_3y)
r2_3y = r2_score(y_test_3y, y_pred_3y)
mae_3y = mean_absolute_error(y_test_3y, y_pred_3y)
mape_3y = mean_absolute_percentage_error(y_test_3y, y_pred_3y)
accuracy_3y = 100 - mape_3y

print("📊 3-Year R² Score:", r2_3y)
print("📊 3-Year MAE:", mae_3y)
print("📊 3-Year MSE:", mse_3y)
print("📊 3-Year RMSE:", rmse_3y)
print("📊 3-Year MAPE:", mape_3y)
print("✅ 3-Year Accuracy:", accuracy_3y)

# --- 5-Year Evaluation ---
mse_5y = mean_squared_error(y_test_5y, y_pred_5y)
rmse_5y = np.sqrt(mse_5y)
r2_5y = r2_score(y_test_5y, y_pred_5y)
mae_5y = mean_absolute_error(y_test_5y, y_pred_5y)
mape_5y = mean_absolute_percentage_error(y_test_5y, y_pred_5y)
accuracy_5y = 100 - mape_5y

print("\n📊 5-Year R² Score:", r2_5y)
print("📊 5-Year MAE:", mae_5y)
print("📊 5-Year MSE:", mse_5y)
print("📊 5-Year RMSE:", rmse_5y)
print("📊 5-Year MAPE:", mape_5y)
print("✅ 5-Year Accuracy:", accuracy_5y)
```

```
📊 3-Year R² Score: 0.9646616569380386
📊 3-Year MAE: 0.6661671142523047
📊 3-Year MSE: 1.5694164361405558
📊 3-Year RMSE: 1.2527635196399023
📊 3-Year MAPE: 6.75614347300779
✅ 3-Year Accuracy: 93.2438565269922

📊 5-Year R² Score: 0.955946948812334
📊 5-Year MAE: 0.8767038462610565
📊 5-Year MSE: 1.9329910215450399
📊 5-Year RMSE: 1.3903204744033082
📊 5-Year MAPE: 9.451003314382953
✅ 5-Year Accuracy: 90.54899668561704
```

```python
#Check for Outliers, to increase the 5Year accuracy
import matplotlib.pyplot as plt
plt.boxplot(y_test_5y)
plt.title("Boxplot of 5-Year Returns (y_test_5y)")
plt.show()

print("Top 5 largest values:", np.sort(y_test_5y)[-5:])
```




```
Top 5 largest values: [33.27677137 33.62512183 33.83521623 33.99770764 35.49624226]
```

```python
# Some Predictions Are Way Off, Printing some error samples to confirm
for true_val, pred_val in zip(y_test_5y, y_pred_5y):
    perc_error = abs((true_val - pred_val) / true_val) * 100
    if perc_error > 100:
        print(f"Actual: {true_val:.2f}, Predicted: {pred_val:.2f}, Error%: {perc_error:.2f}")
```

```
#If you see many samples where the percentage error is >100%, this is what's skewing The MAPE.
```

```
Actual: 3.06, Predicted: 7.21, Error%: 135.71
Actual: 1.69, Predicted: 6.19, Error%: 266.20
Actual: 0.99, Predicted: 4.42, Error%: 347.74
```

```
#Using SMAPE Instead of MAPE ,SMAPE is more stable when both actual and predicted values are large or small.
def smape(y_true, y_pred):
    y_true, y_pred = np.array(y_true), np.array(y_pred)
    denominator = (np.abs(y_true) + np.abs(y_pred)) / 2.0
    diff = np.abs(y_true - y_pred) / np.clip(denominator, 1e-8, None)
    return np.mean(diff) * 100

smape_val = smape(y_test_5y, y_pred_5y)
print("📊 SMAPE for 5-Year Model:", smape_val)
print("✅ SMAPE-based for 5-year Accuracy:", 100 - smape_val)
```

Show hidden output

```
# Define SMAPE function, for better results.
def mean_absolute_percentage_error(y_true, y_pred):
    y_true, y_pred = np.array(y_true), np.array(y_pred)
    return np.mean(np.abs((y_true - y_pred) / np.clip(np.abs(y_true), 1e-8, None))) * 100

#Using SMAPE Instead of MAPE ,SMAPE is more stable when both actual and predicted values are large or small
def smape(y_true, y_pred):
    y_true, y_pred = np.array(y_true), np.array(y_pred)
    denominator = (np.abs(y_true) + np.abs(y_pred)) / 2.0
    diff = np.abs(y_true - y_pred) / np.clip(denominator, 1e-8, None)
    return np.mean(diff) * 100

# --- 5-Year Model Metrics ---
mse_5y = mean_squared_error(y_test_5y, y_pred_5y)
rmse_5y = np.sqrt(mse_5y)
mae_5y = mean_absolute_error(y_test_5y, y_pred_5y)
r2_5y = r2_score(y_test_5y, y_pred_5y)
mape_5y = mean_absolute_percentage_error(y_test_5y, y_pred_5y)
smape_5y = smape(y_test_5y, y_pred_5y)
accuracy_5y = 100 - smape_5y  # Use SMAPE for 5Y accuracy

# --- Print Results ---

print("📊 5-Year R² Score:", r2_5y)
print("📊 5-Year MAE:", mae_5y)
print("📊 5-Year MSE:", mse_5y)
print("📊 5-Year RMSE:", rmse_5y)
print("📊 5-Year MAPE:", mape_5y)
print("📊 5-Year SMAPE:", smape_5y)
print("✅ 5-Year Accuracy:", accuracy_5y)
```

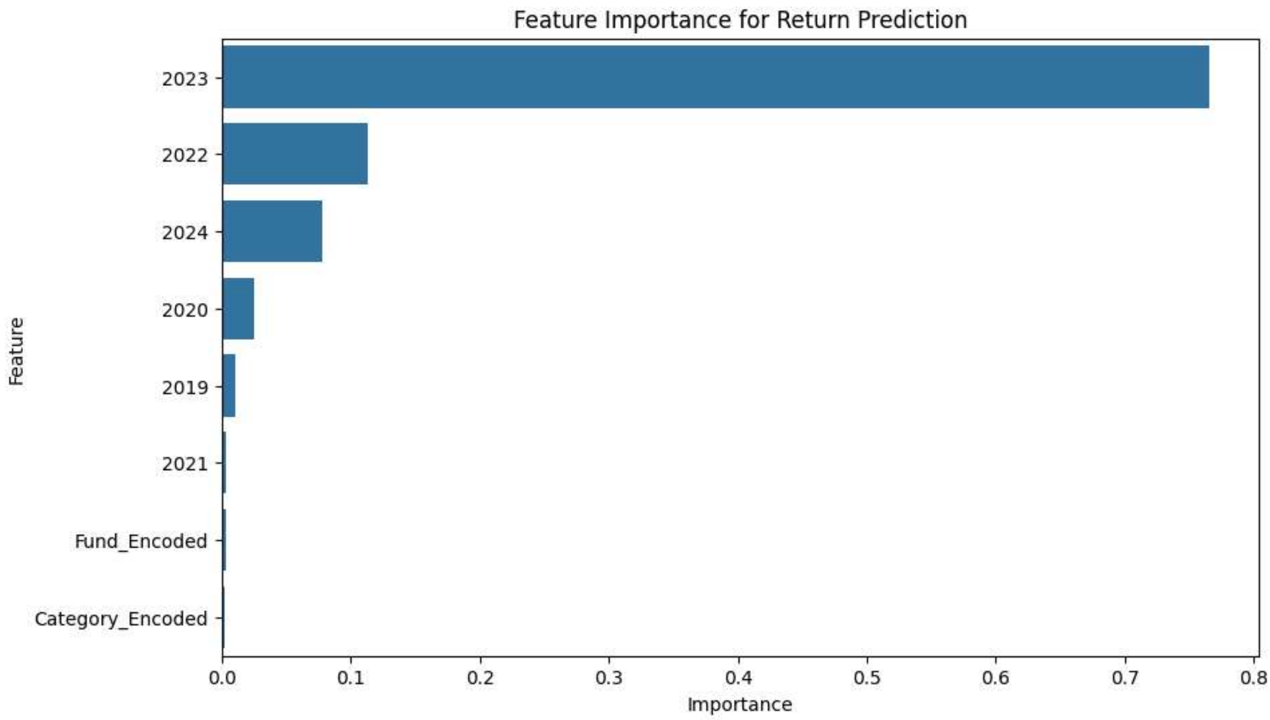
```
📊 5-Year R² Score: 0.955946948812334
📊 5-Year MAE: 0.8767038462610565
📊 5-Year MSE: 1.9329910215450399
📊 5-Year RMSE: 1.3903204744033082
📊 5-Year MAPE: 9.451003314382953
📊 5-Year SMAPE: 7.97500803622984
✅ 5-Year Accuracy: 92.02499196377016
```

```
# Step 7: Predict on full dataset
df["Predicted_3Y_Return"] = model_3y.predict(df[features])
df["Predicted_5Y_Return"] = model_5y.predict(df[features])


def predict_for_fund(fund_name):
    fund_row = df[df["Fund"] == fund_name]
    if fund_row.empty:
        print(f"⚠️ Fund '{fund_name}' not found.")
    else:
        pred_3y = fund_row["Predicted_3Y_Return"].values[0]
        pred_5y = fund_row["Predicted_5Y_Return"].values[0]
        print(f"🔮 Fund: {fund_name}")
        print(f"📈 Predicted 3-Year Return: {pred_3y:.2f}%")
        print(f"📈 Predicted 5-Year Return: {pred_5y:.2f}%")

# Example:
predict_for_fund("Quant Small Cap Fund(G)")
```

```
🔮 Fund: Quant Small Cap Fund(G)
📈 Predicted 3-Year Return: 25.68%
```

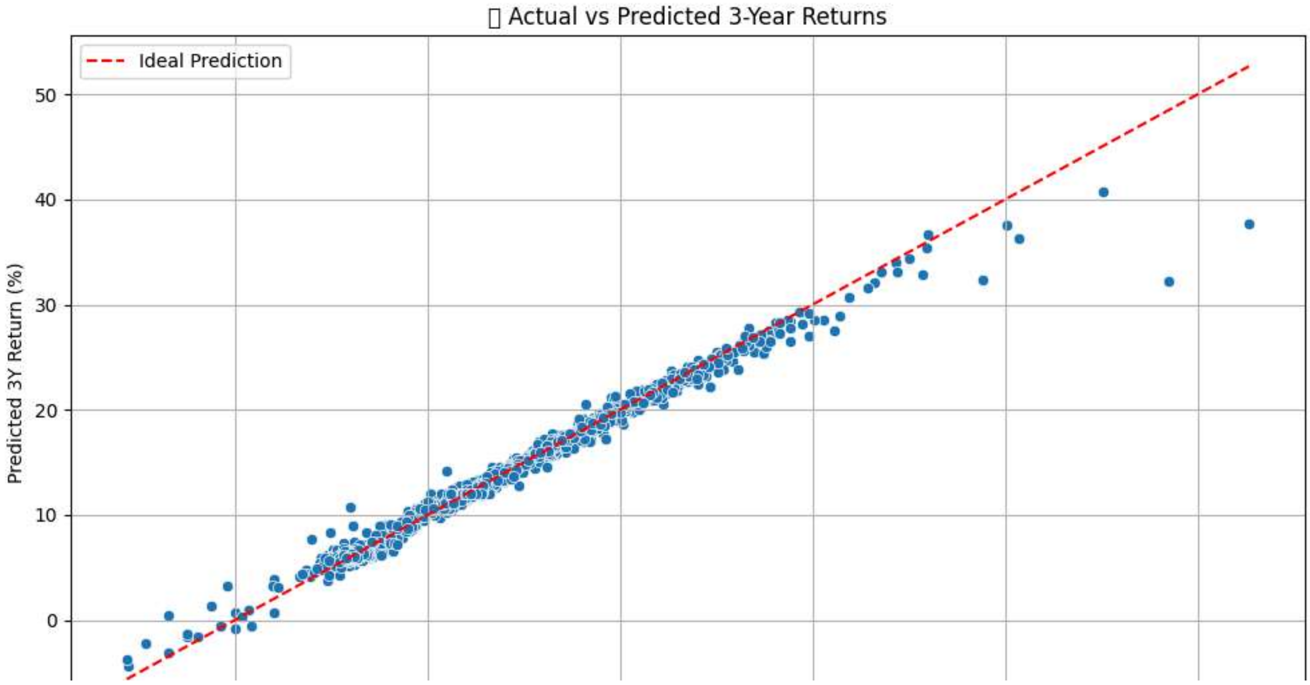
📈 Predicted 5-Year Return: 43.39%

```
import matplotlib.pyplot as plt
import seaborn as sns

# Assuming 'model_3y' or 'model_5y' is your trained RandomForestRegressor
# Replace with the appropriate model if needed
model = model_3y

# Get feature importances
importances = model.feature_importances_

# Get feature names
feature_names = features  # Assuming 'features' is defined as in your code

# Create a DataFrame for plotting
importance_df = pd.DataFrame({'Feature': feature_names, 'Importance': importances})

# Sort by importance
importance_df = importance_df.sort_values(by='Importance', ascending=False)

# Create the bar plot
plt.figure(figsize=(10, 6))
sns.barplot(x='Importance', y='Feature', data=importance_df)
plt.title('Feature Importance for Return Prediction')
plt.xlabel('Importance')
plt.ylabel('Feature')
plt.show()
```




```
import matplotlib.pyplot as plt
import seaborn as sns

# Plotting Actual vs Predicted for 3-Year Returns
plt.figure(figsize=(10, 6))
sns.scatterplot(x=df["Future_3Y_Return"], y=df["Predicted_3Y_Return"])
plt.plot([df["Future_3Y_Return"].min(), df["Future_3Y_Return"].max()],
         [df["Future_3Y_Return"].min(), df["Future_3Y_Return"].max()],
         color='red', linestyle='--', label='Ideal Prediction')
plt.title("📈 Actual vs Predicted 3-Year Returns")
plt.xlabel("Actual 3Y Return (%)")
plt.ylabel("Predicted 3Y Return (%)")
plt.legend()
plt.grid(True)
plt.tight_layout()
plt.show()
```

```
/tmp/ipython-input-69-1715863340.py:15: UserWarning: Glyph 128200 (\N{CHART WITH UPWARDS TREND}) missing from font(s) DejaVu Sans.
  plt.tight_layout()
/usr/local/lib/python3.11/dist-packages/IPython/core/pylabtools.py:151: UserWarning: Glyph 128200 (\N{CHART WITH UPWARDS TREND}) mis
  fig.canvas.print_figure(bytes_io, **kw)
```




```
import matplotlib.pyplot as plt
import seaborn as sns

# Plotting Actual vs Predicted for 3-Year Returns
plt.figure(figsize=(10, 6))
sns.scatterplot(x=df["Future_5Y_Return"], y=df["Predicted_5Y_Return"])
plt.plot([df["Future_5Y_Return"].min(), df["Future_5Y_Return"].max()],
         [df["Future_5Y_Return"].min(), df["Future_5Y_Return"].max()],
         color='red', linestyle='--', label='Ideal Prediction')
plt.title("📈 Actual vs Predicted 5-Year Returns")
plt.xlabel("Actual 5Y Return (%)")
plt.ylabel("Predicted 5Y Return (%)")
plt.legend()
plt.grid(True)
plt.tight_layout()
plt.show()
```

```
/tmp/ipython-input-70-794881792.py:15: UserWarning: Glyph 128200 (\N{CHART WITH UPWARDS TREND}) missing from font(s) DejaVu Sans.
  plt.tight_layout()
/usr/local/lib/python3.11/dist-packages/IPython/core/pylabtools.py:151: UserWarning: Glyph 128200 (\N{CHART WITH UPWARDS TREND}) mis
  fig.canvas.print_figure(bytes_io, **kw)
```