Time Series Forecasting for Portfolio Management Optimization

1. Introduction

This documentation outlines the implementation details of the Time Series Forecasting for Portfolio Management Optimization project. The primary objective is to leverage historical financial data to predict market trends, optimize asset allocation, and enhance portfolio performance for GMF Investments.

2. Data Preprocessing and Exploration

2.1 Data Extraction

- Used **YFinance API** to extract historical data for **Tesla (TSLA), Vanguard Total Bond Market ETF (BND), and S&P 500 ETF (SPY)**.
- The dataset includes **adjusted closing prices, trading volumes, and daily returns**.
- Python libraries used: `yfinance`, `pandas`, `numpy`.

Code Implementation:

```
import yfinance as yf
import pandas as pd

# Define assets
assets = ['TSLA', 'BND', 'SPY']

# Fetch historical data
start_date = '2015-01-01'
end_date = '2025-02-28'
data = yf.download(assets, start=start_date, end=end_date)['Adj Close']
data.to_csv('financial_data.csv')
```

2.2 Data Cleaning

- Checked for missing values and handled them using **linear interpolation**.
- Converted date columns to datetime format.
- Normalized data using Min-Max scaling.

Code Implementation:

```
data = pd.read_csv('financial_data.csv', index_col=0, parse_dates=True)
data.interpolate(method='linear', inplace=True)
data = (data - data.min()) / (data.max() - data.min())
```

2.3 Exploratory Data Analysis (EDA)

- **Visualized trends** using Matplotlib and Seaborn.
- **Computed daily returns** to analyze volatility.
- **Applied rolling mean & standard deviation** to assess fluctuations.
- **Detected outliers** using the Interquartile Range (IQR) method.
- **Seasonality analysis** performed using Statsmodels.

Code Implementation:

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

data.pct_change().dropna().plot(figsize=(12,6), title="Daily Returns")
plt.show()
```

3. Time Series Forecasting Model Development

3.1 Model Selection

Three models were implemented and compared:

- **ARIMA (AutoRegressive Integrated Moving Average)**
- **SARIMA (Seasonal ARIMA)**
- **LSTM (Long Short-Term Memory Network)**

3.2 Model Training and Evaluation

- **Splitting the dataset** (80% Training, 20% Testing).
- **Hyperparameter tuning** using Grid Search (for ARIMA/SARIMA) and optimizing neural network architecture (for LSTM).
- **Evaluation Metrics**:
    - Mean Absolute Error (MAE)
    - Root Mean Squared Error (RMSE)
    - Mean Absolute Percentage Error (MAPE)

Code Implementation (ARIMA Model):

```
from statsmodels.tsa.arima.model import ARIMA
```

```
# Train ARIMA Model
model = ARIMA(data['TSLA'], order=(5,1,0))
model_fit = model.fit()
forecast = model_fit.forecast(steps=30)
```

Code Implementation (LSTM Model):

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense

# Create LSTM Model
model = Sequential()
model.add(LSTM(units=50, return_sequences=True, input_shape=(30, 1)))
model.add(LSTM(units=50))
model.add(Dense(units=1))
model.compile(optimizer='adam', loss='mean_squared_error')
```

4. Forecasting Future Market Trends

- **Generated 6-12 month forecasts** for Tesla stock using the best-performing model.
- **Plotted confidence intervals** to show expected price range.
- **Identified key market opportunities and risks** based on forecasted trends.

Code Implementation (Forecasting Visualization):

```
plt.figure(figsize=(12,6))
plt.plot(data.index, data['TSLA'], label="Actual Prices")
plt.plot(forecast, label="Forecasted Prices")
plt.fill_between(forecast.index, forecast - 0.1*forecast, forecast + 0.1*forecast, color='b',
alpha=0.2)
plt.legend()
plt.show()
```

5. Portfolio Optimization Based on Forecast

5.1 Portfolio Construction

- Created a portfolio using **TSLA, BND, and SPY**.
- **Extended forecasting models to BND and SPY**.
- Consolidated forecasted data into a single dataframe.

5.2 Optimization Process

- **Calculated annual returns** using log returns.
- **Computed covariance matrix** to analyze asset correlation.
- **Implemented Markowitz Portfolio Optimization** to maximize the Sharpe Ratio.

Code Implementation (Sharpe Ratio Maximization):

```
import numpy as np
from scipy.optimize import minimize

def portfolio_volatility(weights, cov_matrix):
    return np.sqrt(np.dot(weights.T, np.dot(cov_matrix, weights)))

def sharpe_ratio(weights, returns, cov_matrix, risk_free_rate=0.02):
    return -((np.dot(weights, returns) - risk_free_rate) / portfolio_volatility(weights, cov_matrix))

# Optimizing portfolio allocation
num_assets = len(assets)
initial_weights = [1/num_assets] * num_assets
bounds = [(0,1) for _ in range(num_assets)]
constraints = {'type': 'eq', 'fun': lambda w: np.sum(w) - 1}
result = minimize(sharpe_ratio, initial_weights, args=(expected_returns, covariance_matrix),
bounds=bounds, constraints=constraints)
```

5.3 Results and Adjustments

- Increased allocation to **BND** to mitigate **Tesla's high volatility**.
- **Portfolio Sharpe Ratio increased**, ensuring optimal risk-adjusted returns.
- Visualized **cumulative returns and risk-adjusted performance**.

6. Conclusion

- Successfully **extracted, cleaned, and analyzed** financial data.
- Developed **time series forecasting models** with high accuracy.
- **Optimized asset allocation** for enhanced portfolio performance.
- **Provided actionable investment recommendations** based on model insights.

7. Recommendations

- **Implement real-time market monitoring** for dynamic asset rebalancing.
- **Experiment with hybrid models** (ARIMA + LSTM) to improve accuracy.
- **Develop automated trading strategies** based on forecasted trends.

8. Tools and Technologies Used

- **Programming Language**: Python
- **Libraries**: pandas, numpy, matplotlib, seaborn, statsmodels, scikit-learn, tensorflow, keras, yfinance, scipy.optimize
- **Data Source**: YFinance API
- **Optimization Techniques**: Sharpe Ratio Maximization, Risk Management Strategies

---

Prepared by: Amanuel Legesse
Program: 10 Academy AI Mastery Training
Date: March 4, 2025