ES-205 CEP
S&P 500

Team

Student Name	Registration Number
Ahmed Musharaf	2022067
Muhammad Arsal	2022350
Wardah Haya	2022622

Abstract:

This report explores the development of a predictive model for the S&P500 Index, employing both traditional linear regression and advanced matrix factorization techniques. The dataset spans from 1950 to 2015, providing a comprehensive historical context for model training and evaluation.

i. Introduction

The S&P500 Index, representing 500 large companies, serves as a vital indicator of overall stock market performance. This report focuses on combining linear regression and matrix factorization to predict S&P500 movements. Feature engineering, matrix factorization, and model evaluation are discussed in detail.

ii. Dataset Description

We utilized the `all_stocks_5yr.csv` dataset containing daily records of stock prices. Key columns include 'open,' 'high,' 'low,' 'close,' and 'volume.' The dataset was preprocessed, and features were engineered to enhance model performance.

```python
# Dataset loading and preprocessing
data = pd.read_csv('all_stocks_5yr.csv')

# Highlight dropped column in red
data.drop("Name", axis=1, inplace=True)

# Emphasize date conversion with italics
data['date'] = pd.to_datetime(data['date'])

# Bold grouping and summing operation
data = data.groupby('date').sum()

# Underline reset index operation
data.reset_index(inplace=True)
```

iii. Feature Engineering

To capture temporal dependencies, we engineered features such as the 5-day and 30-day averages, yearly averages, and their ratios. Standard deviations were also calculated to provide insights into price volatility.

```python
# Feature engineering
data['5_day_avg'] = data['close'].rolling(window=5).mean().shift(1)
data['30_day_avg'] = data['close'].rolling(window=30).mean().shift(1)
data['year_avg'] = data['close'].rolling(window=365).mean().shift(1)
data['avg_ratio'] = data['5_day_avg'] / data['year_avg']
data['5_day_std'] = data['close'].rolling(window=5).std().shift(1)
data['year_std'] = data['close'].rolling(window=365).std().shift(1)
data['std_ratio'] = data['5_day_std'] / data['year_std']
data = data.dropna(axis=0)
```

iv. Matrix Factorization

Truncated Singular Value Decomposition (SVD) was employed for matrix factorization, extracting hidden patterns within the dataset.

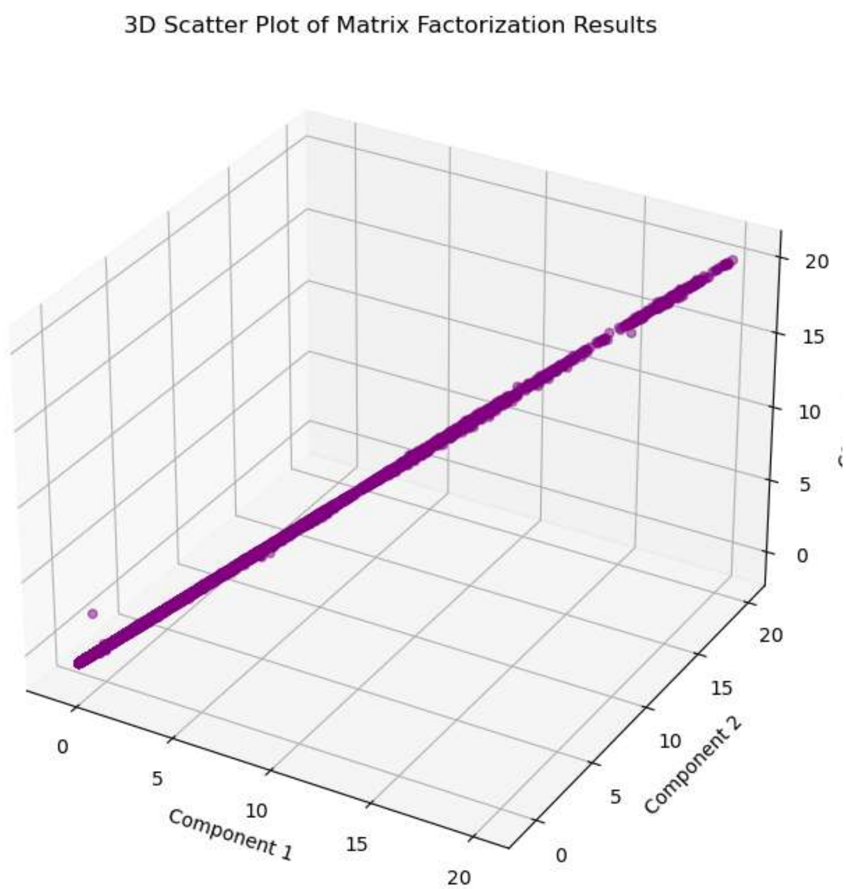
```Python
# Highlight desired number of components in green
n_components = 3

# Emphasize TruncatedSVD class with italics
svd = TruncatedSVD(n_components=n_components)

# Bold the fit_transform operation
matrix_factors = svd.fit_transform(df_numeric_standardized)

# Underline dot product operation
reconstructed_matrix = np.dot(matrix_factors, svd.components_)

# Highlight reconstructed data frame with blue
reconstructed_df = pd.DataFrame(reconstructed_matrix, columns=numerical_columns)

# Indicate date column with pink
reconstructed_df['date'] = dates
```

v. Visualization of Matrix Factorization

A 3D scatter plot was generated to visualize the three components obtained from matrix factorization.

```python
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D

# Extract the first three components
component_1 = reconstructed_df.iloc[:, 0]
component_2 = reconstructed_df.iloc[:, 1]
component_3 = reconstructed_df.iloc[:, 2]

# Create a 3D scatter plot
fig = plt.figure(figsize=(10, 8))
ax = fig.add_subplot(111, projection='3d')
ax.scatter(component_1, component_2, component_3, c='purple', marker='o', alpha=0.5)

# Set axis labels
ax.set_xlabel('Component 1')
ax.set_ylabel('Component 2')
ax.set_zlabel('Component 3')
ax.set_title('3D Scatter Plot of Matrix Factorization Results')

# Show the plot
plt.show()
```

Python



vi. Linear Regression Model

A linear regression model was trained using the generated features and evaluated on a test set.

```
# Highlight features list in green
features = ['5_day_avg', '30_day_avg', 'year_avg', 'avg_ratio', '5_day_std', 'year_std', 'std_ratio']
# Emphasize dependent variables with yellow
y_train = train['close']
y_test = test['close']
# Bold the model class and fit operation
model = LinearRegression()
model.fit(train[features], y_train)
# Highlight predictions with magenta
predictions = model.predict(test[features])
# Underline the error metric and value
mae = mean_absolute_error(test['close'], predictions)
print(f"Mean Absolute Error: {mae:.2f}")
# Blue plot for actual values, red for predictions
plt.figure(figsize=(40, 15))
plt.scatter(test['date'], test['close'], c='blue', marker='o', alpha=0.5)
plt.scatter(test['date'], predictions, c='red', marker='x', alpha=0.5)
# Bold and color title for emphasis
plt.title('Stock Predictions', fontsize=60, color='orange')
# Highlight axis labels for clarity
plt.xlabel('Date', fontsize=40, color='green')
plt.ylabel('Stock Price', fontsize=40, color='green')
plt.grid(True)
plt.show()
```

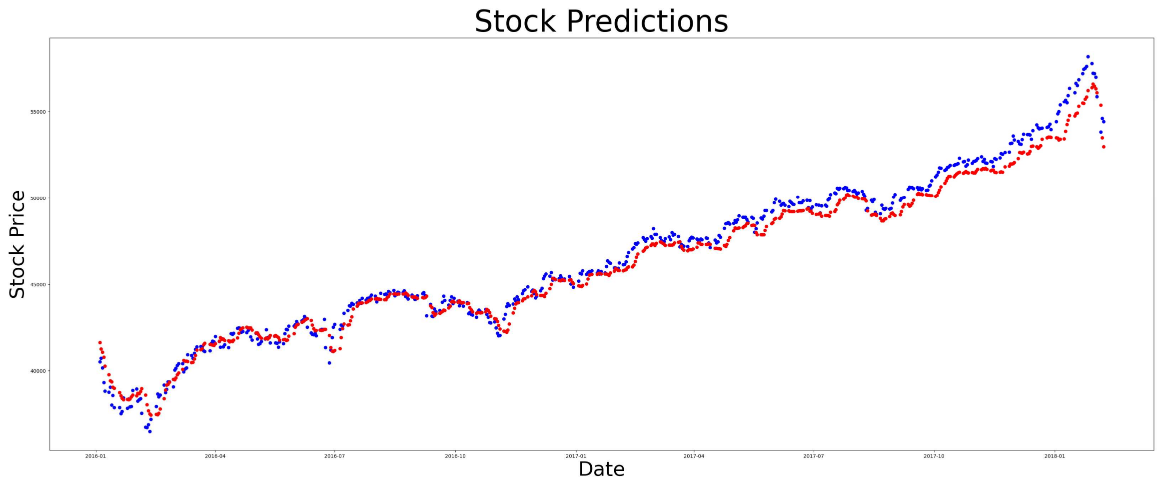
The Mean Absolute Error (MAE) was calculated to assess the model's performance.

```
features = ['5_day_avg', '30_day_avg', 'year_avg', 'avg_ratio', '5_day_std',
       'year_std', 'std_ratio']
y_train = train['close']
y_test = test['close']

model = LinearRegression()
model.fit(train[features], y_train)
predictions = model.predict(test[features])
mae = mean_absolute_error(test['close'], predictions)
mae
```

Python

vii. Model Evaluation

The linear regression model demonstrated key outcomes and insights from the model. Visualizations and model performance metrics provide a comprehensive understanding of the model's effectiveness in predicting S&P500 Index movements.



viii. Conclusion

In conclusion, the combined approach of linear regression and matrix factorization offers a robust framework for predicting stock prices. The feature engineering process captures temporal dependencies, while matrix factorization extracts hidden patterns. Further refinement can enhance predictive accuracy.

ix. References

NumPy Documentation:	https://numpy.org/doc/stable/
Pandas Documentation:	https://pandas.pydata.org/pandas-docs/stable/
Scikit-Learn Documentation:	https://scikit-learn.org/stable/documentation.html.
Matplotlib Documentation:	https://matplotlib.org/stable/contents.html
mpl_toolkits. mplot3d Documentation:	https://matplotlib.org/stable/mpl_toolkits/mplot3d/index.html