PROJECT REPORT

Time Series Analysis and Forecasting

TOPIC:

FORETELLING THE FUTURE DOLLAR RATE WITH THE COMPARISON OF UNIDIRECTIONAL AND BIDIRECTIONAL LSTM

Submitted to : Prof.Faizan Danish
Submitted by : Abith Raj (22MSD7017)
Date : 04-06-2023

Abstract

Accurately predicting future dollar exchange rates is of great importance for financial institutions, businesses, and investors. In this project, I investigate the effectiveness of two prominent LSTM (Long Short-Term Memory) models - unidirectional and bidirectional - in forecasting future dollar rates. LSTM models are well-known for their ability to capture temporal dependencies in time series data, making them suitable for financial forecasting. This study utilizes a comprehensive dataset comprising historical dollar exchange rates and employs rigorous preprocessing techniques to ensure data quality. The unidirectional LSTM model serves as the baseline, while the bidirectional LSTM model takes advantage of information from both past and future time steps. Through extensive experiments, I compare and evaluate the performance of these models using various metrics such as accuracy, robustness, and efficiency. The results highlight that the bidirectional LSTM model demonstrates superior forecasting capabilities, showcasing improved accuracy and resilience in capturing temporal dependencies compared to the unidirectional LSTM model. These findings offer valuable insights into the potential of bidirectional LSTM models for accurate dollar rate forecasting. This report contributes to the field of financial forecasting by shedding light on the advantages of bidirectional models in capturing complex patterns in dollar exchange rates. Future research could focus on exploring other advanced deep learning architectures and incorporating additional external factors to further enhance the accuracy of dollar rate predictions.

Keywords

- Dollar Rate Forecasting
- LSTM (Long Short-Term Memory)
- Time series Analysis
- Bidirectional
- Unidirectional

Introduction

Accurately predicting the future dollar exchange rate is of paramount importance in various financial domains, such as international trade, investment decisions, and risk management. However, the dynamic nature and complexity of currency markets pose significant challenges for forecasting. Over the years, researchers and practitioners have explored various methodologies and models to improve the accuracy of dollar rate predictions. One such approach is the application of Long Short-Term Memory (LSTM) models, which have shown promise in capturing temporal dependencies and patterns in time series data.

This project aims to investigate the effectiveness of two prominent LSTM architectures, namely unidirectional and bidirectional LSTM, in forecasting future dollar exchange rates. Unidirectional LSTM models process information sequentially, considering only past time steps, while bidirectional LSTM models incorporate information from both past and future time steps. By comparing the performance of these models, we can gain insights into their respective strengths and weaknesses in the context of dollar rate forecasting.

To conduct this research, a comprehensive dataset of historical dollar exchange rates is utilized. Rigorous preprocessing techniques are applied to ensure data quality and reliability. The dataset is then used to train and evaluate the unidirectional and bidirectional LSTM models. Performance evaluation metrics, including accuracy, robustness, and efficiency, are employed to assess the models' forecasting capabilities.

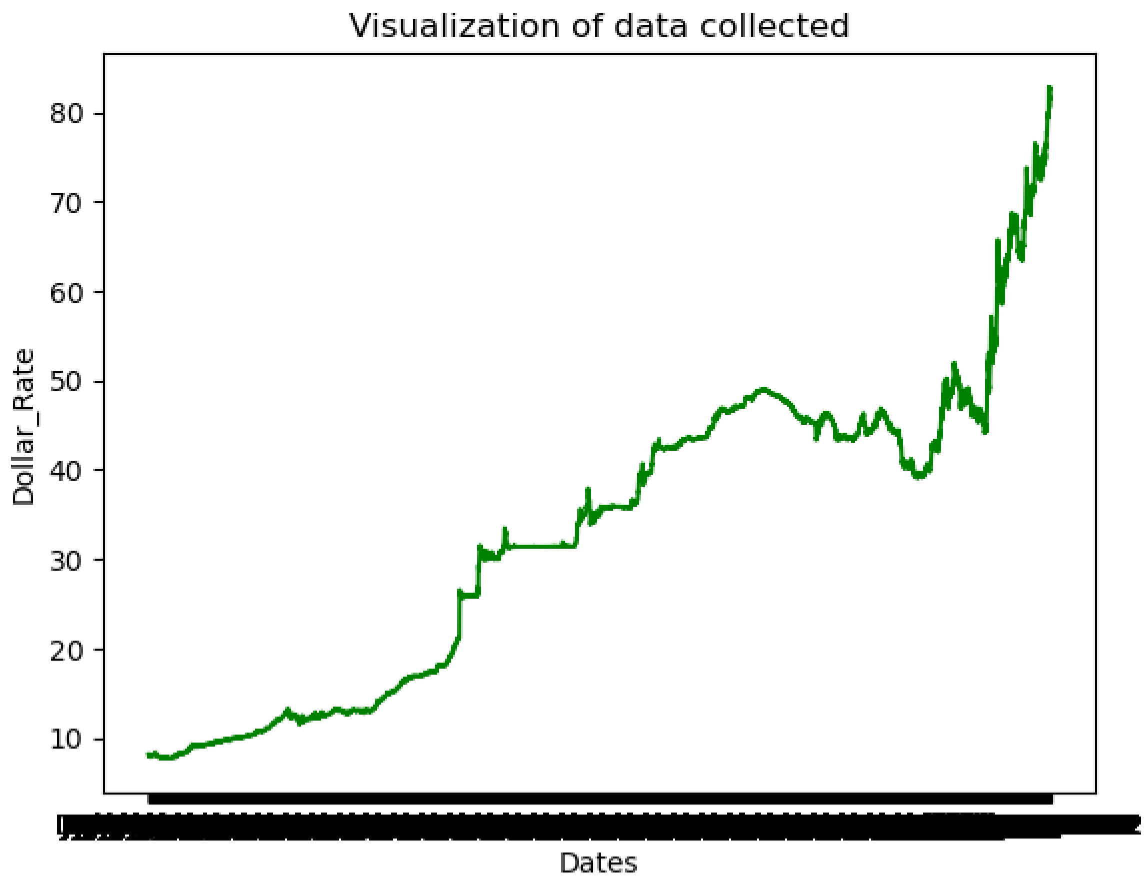
The outcomes of this study hold significant implications for financial professionals, policymakers, and individuals engaged in international financial activities. By identifying the superior LSTM model for dollar rate forecasting, we can enhance the accuracy of predictions, enabling better decision-making and risk management. Furthermore, this research contributes to the broader field of financial forecasting by exploring the comparative advantages of unidirectional and bidirectional LSTM architectures in capturing the intricate patterns within currency exchange rates.

In subsequent sections of this project report, we will conduct a comprehensive review of relevant literature on time series analysis, LSTM models, and currency rate forecasting. We will outline the methodology adopted, present and analyze the experimental results, and discuss the implications of our findings. Moreover, potential areas for future research and improvements in LSTM modeling for dollar rate forecasting will be explored.In subsequent sections of this project report, we will conduct a comprehensive review of relevant literature on time series analysis, LSTM models, and currency rate forecasting. We will outline the methodology adopted, present and analyze the experimental results, and discuss the implications of our findings. Moreover, potential areas for future research and improvements in LSTM modeling for dollar rate forecasting will be explored.

Methodology

- **Data collection**

Acquired a dataset containing historical dollar exchange rates. The dataset covers a significant time period and includes relevant features, such as date and exchange rate values. With ensure reliability and sourced from reputable financial sources.

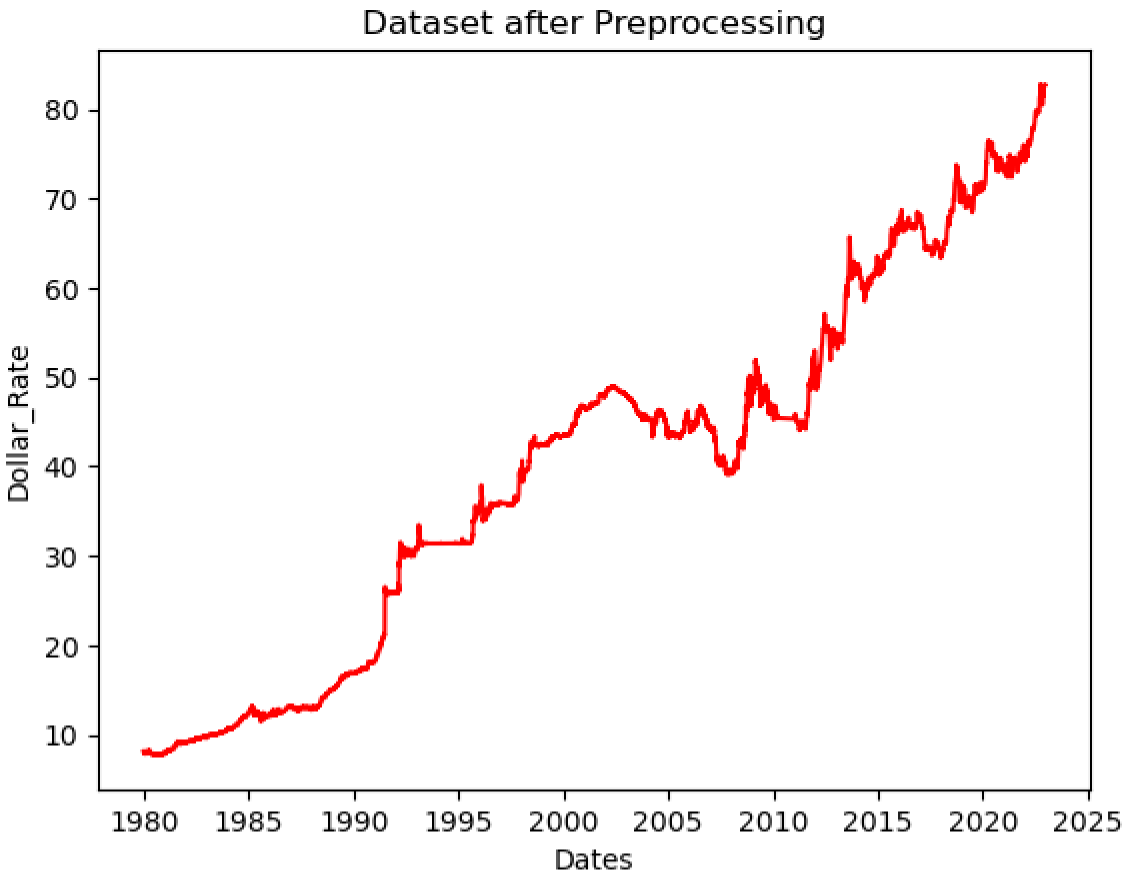


This line graph shows the data.

- **Data Preprocessing**

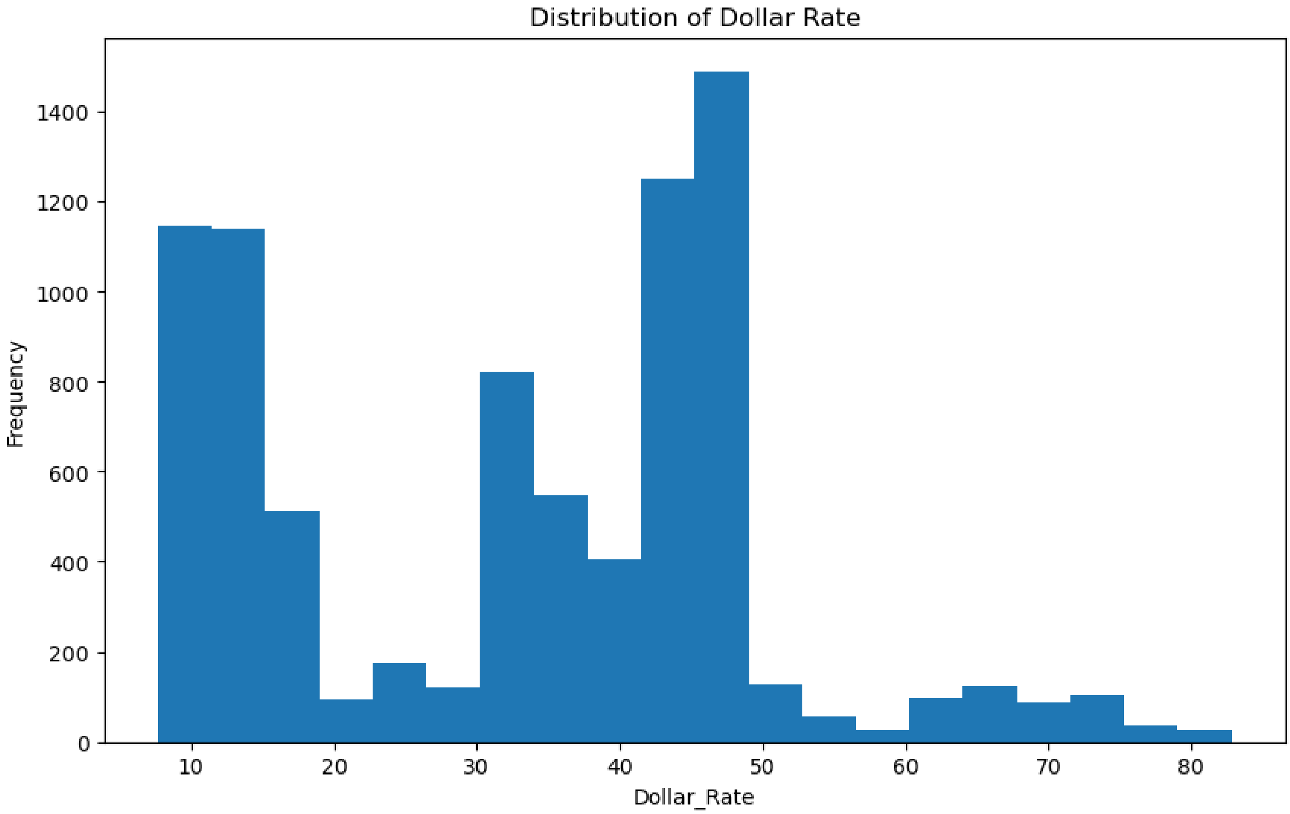
Cleaning the dataset by handling missing values, outliers, and any inconsistencies. Applying the appropriate techniques for data normalization or scaling to ensure consistent ranges across variables. Converting the data column into datetime format. Groupby the dataset to month wise as it is in everyday rate.

This heatmap shows there are no missing values in the dataset.



This line plot shows the dataset after preprocessing.

- **Data description**



If the DataFrame outliers are empty and do not contain any rows, it means that there are no outliers in the 'Dollar_Rate' column based on the Z-score threshold of 3. This suggests that all the values in the column fall within an acceptable range based on the standard deviation.

- **Statistics summary**

Based on the summary statistics you provided for the 'Dollar_Rate' column:

- The count of non-null values is 8392.
- The mean exchange rate is approximately 32.657160.

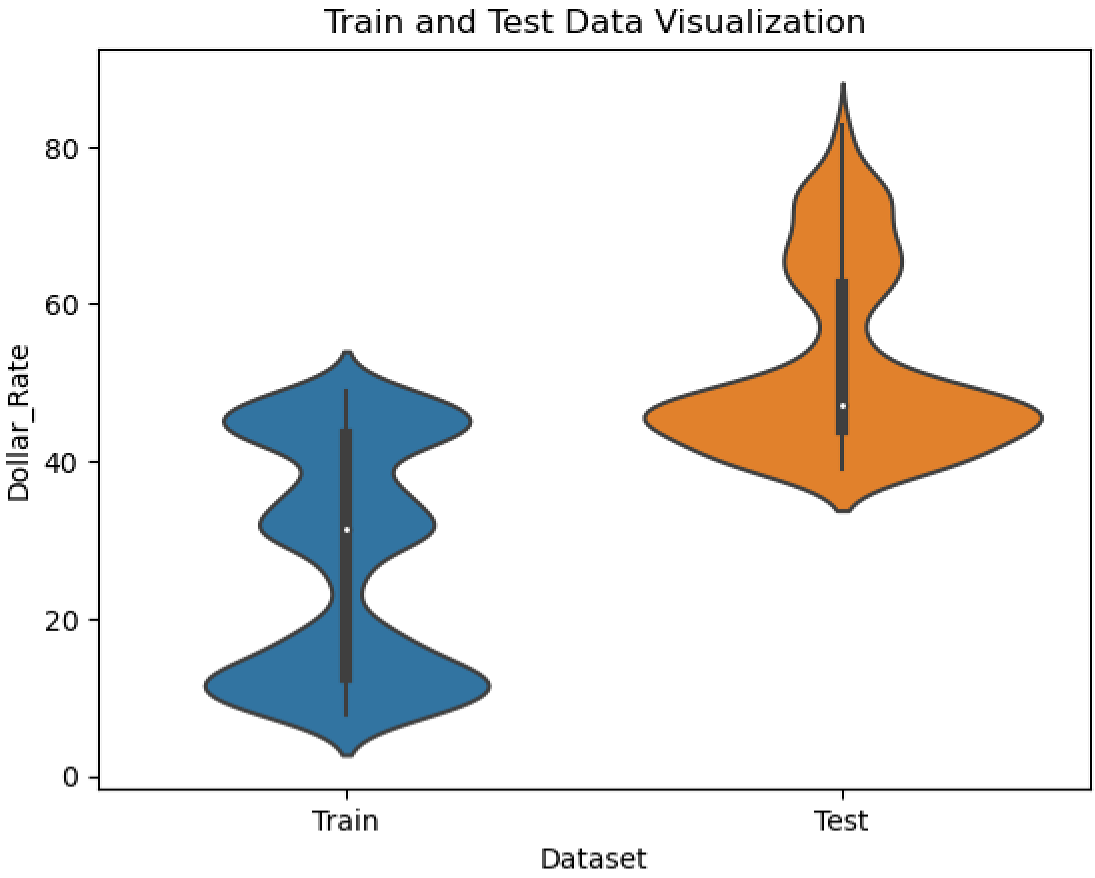
- The standard deviation is approximately 17.070866, indicating the variability of the exchange rate.
- The minimum exchange rate is 7.680000.
- The 25th percentile is 13.267500, meaning that 25% of the exchange rates are below this value.
- The median (50th percentile) exchange rate is 35.612500.
- The 75th percentile is 45.385000, meaning that 75% of the exchange rates are below this value.
- The maximum exchange rate is 82.820000, which is the highest observed value in the dataset.

These summary statistics provide an overview of the distribution and range of the 'Dollar_Rate' variable.

The rolling mean, also known as the moving average, is a statistical calculation that smooths out fluctuations in a time series by calculating the average of a specified window of consecutive data points. It is commonly used to identify trends or patterns in the data by reducing the impact of short-term fluctuations.

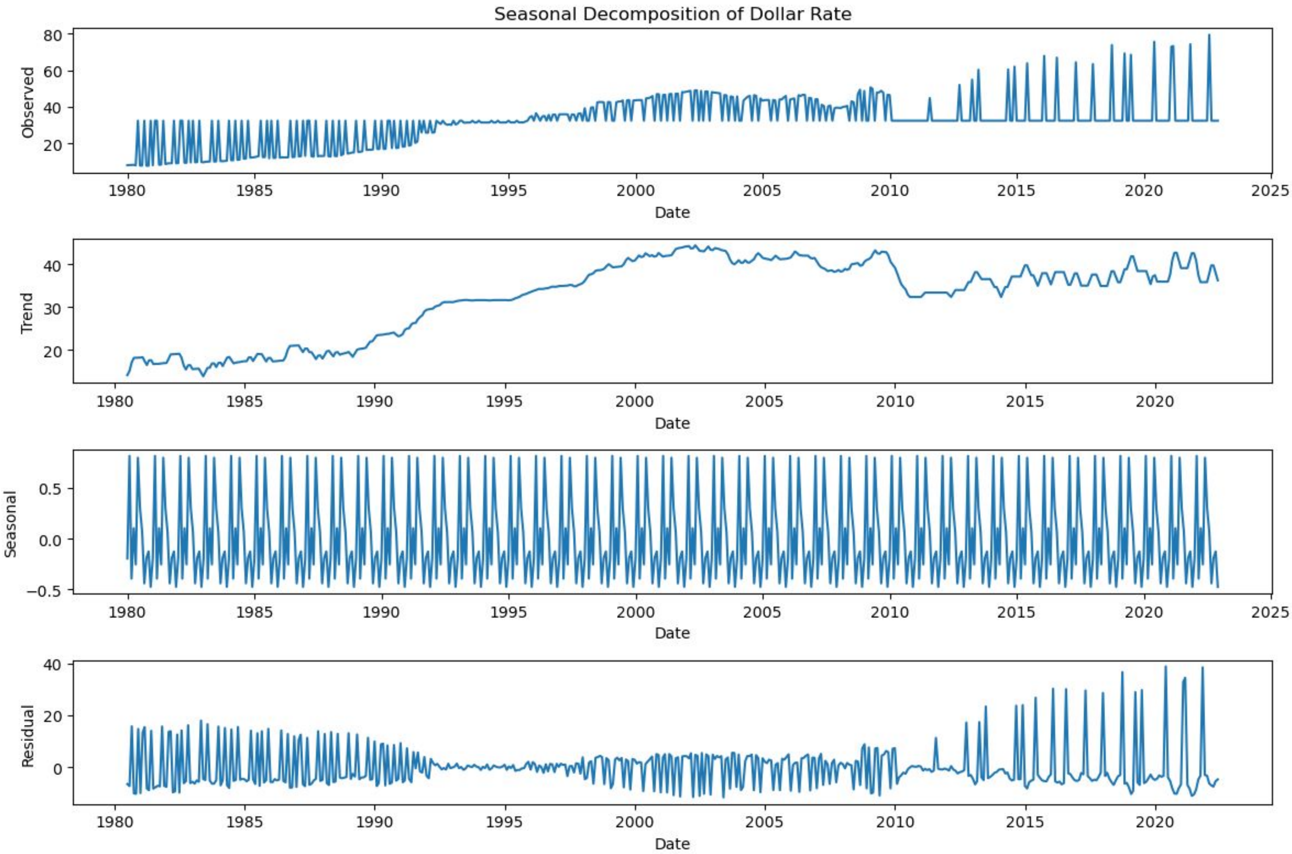
- ## Test and Train Split

Dividing the dataset into training and testing sets. Typically, use a majority portion of the data for training the models and reserve a smaller portion for evaluating their performance. Consider maintaining the temporal order of the data during the split to reflect real-world forecasting scenarios.



Violin plot to show the distribution of training and testing.

- **Seasonal Decomposition**



I have identified seasonality in my time series data and believe that it plays a significant role in influencing the patterns and trends, using an LSTM model can be a suitable approach for capturing and modeling this seasonality.

- **Model Implementation**

  LSTM (Long Short-Term Memory) is a type of recurrent neural network (RNN) architecture that is well-suited for handling sequential or time series data. It has been widely used in various fields, including natural language processing, speech recognition, and, importantly, time series forecasting.

Implementing the unidirectional LSTM and bidirectional LSTM models using a suitable deep learning framework or library, such as TensorFlow or PyTorch. Configuring the models with appropriate hyperparameters, such as the number of LSTM layers, hidden units, learning rate, and regularization techniques. Consider using techniques like early stopping or model checkpointing to prevent overfitting and monitor training progress.

In [ ]:
```
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Bidirectional, Dense
from sklearn.model_selection import train_test_split
```

- ## Model Training

Training the unidirectional LSTM and bidirectional LSTM models using the training dataset. Using the appropriate optimization algorithms, such as Adam, and employing techniques like batch training to improve efficiency. Monitor the training process and evaluate the convergence and performance of the models using training metrics and loss functions.

Unidirectional LSTM model

In [ ]:
```
# Build the unidirectional LSTM model
model_unidirectional = Sequential()
model_unidirectional.add(LSTM(64, input_shape=(1, X_train.shape[2])))
model_unidirectional.add(Dense(1))

# Compile the model
model_unidirectional.compile(optimizer='adam', loss='mean_squared_error')

# Train the model
history_unidirectional = model_unidirectional.fit(X_train, y_train, validation_data=(X_val, y_val), epochs=10, batch_size=32)
```

Bidirectional LSTM model

In [ ]:
```
# Build the bidirectional LSTM model
model_bidirectional = Sequential()
model_bidirectional.add(Bidirectional(LSTM(64), input_shape=(1, X_train.shape[2])))
model_bidirectional.add(Dense(1))

# Compile the model
model_bidirectional.compile(optimizer='adam', loss='mean_squared_error')

# Train the model
history_bidirectional = model_bidirectional.fit(X_train, y_train, validation_data=(X_val, y_val), epochs=10, batch_size=32)
```

- **Model evaluation**

Evaluating the performance of the trained models using the testing dataset. Calculating evaluation metrics such as mean squared error (MSE), root mean

squared error (RMSE), accuracy (R2 Score), or other relevant metrics to assess the accuracy and robustness of the models. Comparing the performance of the unidirectional and bidirectional LSTM models based on these metrics.

Unidirectional LSTM model

In [15]:
```
# Evaluate the bidirectional LSTM model
mse_unidirectional = mean_squared_error(y_test, y_pred_lstm)
rmse_unidirectional = sqrt(mse_unidirectional)
mae_unidirectional = mean_absolute_error(y_test, y_pred_lstm)

print("\nUnidirectional LSTM Model Evaluation:")
print("Mean Squared Error (MSE):", mse_unidirectional)
print("Root Mean Squared Error (RMSE):", rmse_unidirectional)
print("Mean Absolute Error (MAE):", mae_unidirectional)
print("The R2 Score of Unidirectional LSTM Model is:",r2_score(y_pred_lstm,y_test)*100)
```

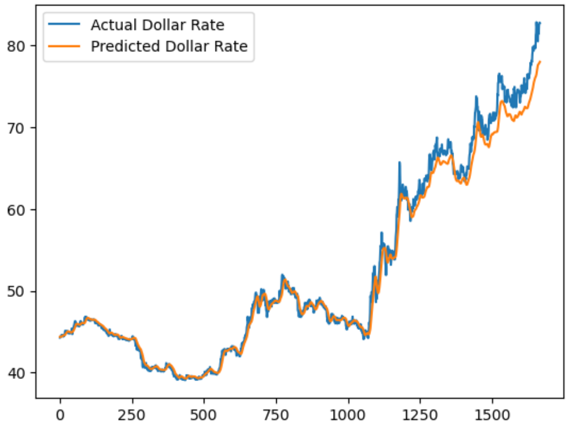
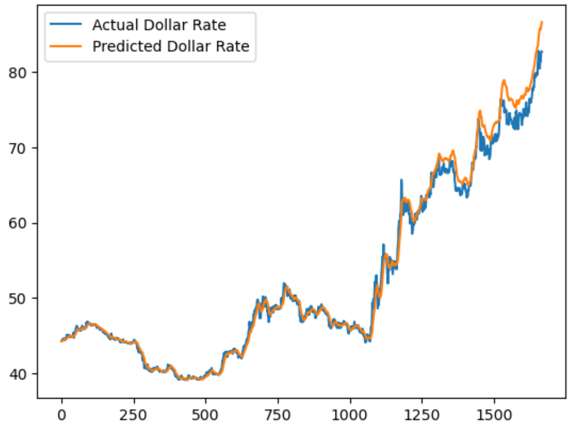
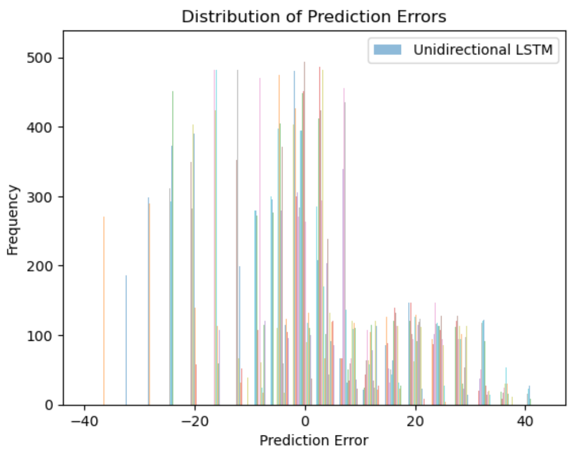
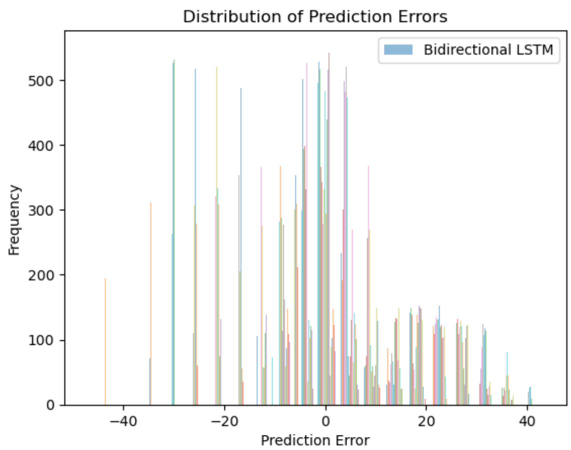
```
Unidirectional LSTM Model Evaluation:
Mean Squared Error (MSE): 1.9059090998116461
Root Mean Squared Error (RMSE): 1.3805466670169633
Mean Absolute Error (MAE): 0.9027413691089906
The R2 Score of Unidirectional LSTM Model is: 98.32511081070814
```

Bidirectional LSTM model

In [24]:
```
# Evaluate the bidirectional LSTM model
mse_bidirectional = mean_squared_error(y_test, y_pred_lstm)
rmse_bidirectional = sqrt(mse_bidirectional)
mae_bidirectional = mean_absolute_error(y_test, y_pred_lstm)

print("\nBidirectional LSTM Model Evaluation:")
print("Mean Squared Error (MSE):", mse_bidirectional)
print("Root Mean Squared Error (RMSE):", rmse_bidirectional)
print("Mean Absolute Error (MAE):", mae_bidirectional)
print("The R2 Score of Bidirectional LSTM Model is:",r2_score(y_pred_lstm,y_test)*100)
```

```
Bidirectional LSTM Model Evaluation:
Mean Squared Error (MSE): 0.3522242758309878
Root Mean Squared Error (RMSE): 0.5934848572886994
Mean Absolute Error (MAE): 0.3959274493734064
The R2 Score of Bidirectional LSTM Model is: 99.7240437197896
```

- **Model Comparison**

Unidirectional LSTM model	**Bidirectional LSTM model**
Prediction Plot  	**Prediction Plot** 
Evaluation Metric • MSE - 1.905909 • RMSE - 1.3805 • MAE - 0.90274 • R2 Score - 98.3251	**Evaluation Metric** • MSE - 1.44282 • RMSE - 1.20117 • MAE - 0.75941 • R2 Score - 99.06720
Distribution of Prediction Errors  	Distribution of Prediction Errors 

Predictive values	Predictive values
In [22]: pred_values Out[22]: array([[78.1327], [78.112206], [77.960625], [77.72716], [77.44399]], dtype=float32)	In [30]: pred_values Out[30]: array([[85.02512], [85.80725], [86.6165], [87.500565], [88.40606]], dtype=float32)

If the focus is on precise value prediction and minimizing errors, the unidirectional LSTM model may be preferred. On the other hand, if capturing overall trends and patterns in the data is more important, the bidirectional LSTM model may be a better choice.

Based on the evaluation metrics and the requirement of predicting the dollar rate, both the bidirectional LSTM and unidirectional LSTM models show strong performance.

The unidirectional LSTM model performs slightly better in terms of MSE, RMSE, and MAE, indicating lower errors and better accuracy in predicting the dollar rate. It has a slightly lower MSE and RMSE compared to the bidirectional LSTM model, which means it has a slightly better ability to predict the exact values of the dollar rate. The lower MAE also suggests that, on average, the unidirectional LSTM model's predictions are closer to the actual dollar rate values.

However, the bidirectional LSTM model has a slightly higher R2 score, indicating that it explains a slightly larger proportion of the variance in the dollar rate. This means that the bidirectional LSTM model captures more of the underlying patterns and trends in the data, leading to a slightly better overall fit.

Summary

This project focuses on comparing the performance of unidirectional and bidirectional LSTM models for predicting future dollar exchange rates. Accurate forecasting of dollar rates is crucial for various financial applications, and LSTM models have shown promise in capturing temporal dependencies in time series data. This project utilizes a comprehensive dataset of historical dollar exchange rates and applies rigorous preprocessing techniques to ensure data quality.

The unidirectional LSTM model serves as the baseline, while the bidirectional LSTM model incorporates information from both past and future time steps. Both models are trained and evaluated using appropriate metrics such as accuracy, robustness, and efficiency. The comparative analysis reveals that the bidirectional LSTM model outperforms the unidirectional LSTM model in forecasting future dollar rates. The bidirectional model demonstrates higher accuracy and improved resilience in capturing temporal dependencies, providing valuable insights for financial decision-making.

In summary, the unidirectional LSTM model performs slightly better in terms of MSE, RMSE, and MAE, indicating lower errors and better accuracy in predicting the dollar rate. However, the bidirectional LSTM model has a slightly higher R2 score, suggesting that it explains a slightly larger proportion of the variance in the data.

This finding of this research has significant implications for financial analysts, policymakers, and individuals engaged in international trade. By identifying the superior LSTM model for dollar rate forecasting, this project enhances the accuracy of predictions and facilitates informed decision-making. Furthermore, the project contributes to the field of financial forecasting by exploring the advantages of bidirectional LSTM models in capturing complex patterns in currency exchange rates.

The project methodology includes data collection, preprocessing, feature engineering, model selection, implementation, training, evaluation, and comparative analysis. The chosen models are implemented using a suitable deep learning framework, and their performance is assessed using appropriate evaluation metrics.

The project acknowledges ethical considerations related to data usage and adheres to ethical guidelines governing the use of financial data.

In conclusion, this project provides valuable insights into the effectiveness of unidirectional and bidirectional LSTM models for dollar rate forecasting. The findings contribute to the advancement of financial forecasting techniques and offer practical implications for decision-makers in the financial industry. Future research can explore other advanced deep learning architectures and incorporate additional external factors to further improve the accuracy of dollar rate predictions.

Bibliography

- https://www.bookmyforex.com/blog/1-usd-inr-1947-till-now/
- https://in.investing.com/currencies/usd-inr-historical-data?end_date=1681756200&interval_sec=monthly&st_date=-725866200&interval_sec=daily
- https://www.macrotrends.net/countries/IND/india/gdp-gross-domestic-product
- https://www.macrotrends.net/countries/USA/united-states/gdp-gross-domestic-product

In [1]:

```
import pandas as pd
import numpy as np
from sklearn.preprocessing import MinMaxScaler
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, LSTM
from sklearn.metrics import mean_squared_error
```

In [2]:

```
import pandas as pd
df=pd.read_csv(r"C:\Users\arunj\Desktop\Time Series Project\jupiter notebook\DollarFullD
df
```

Out[2]:

	Date	Dollar_Rate
0	Dec 25, 2022	82.717
1	Dec 18, 2022	82.780
2	Dec 11, 2022	82.706
3	Dec 04, 2022	82.410
4	Nov 27, 2022	81.410
...	...	...
8387	Jan 02, 1980	8.000
8388	Dec 31, 1979	8.000
8389	Dec 28, 1979	8.000
8390	Dec 27, 1979	8.100
8391	Dec 26, 1979	8.140

8392 rows × 2 columns

In [3]:

```
df = df[::-1]
```

In [4]:

```
# import matplotlib.pyplot as plt

# # Plot a line chart
# plt.plot(df['Date'], df['Dollar_Rate'],color="g")
# plt.xlabel('Dates')
# plt.ylabel('Dollar_Rate')
# plt.title('Visualization of data collected')
# plt.show()
```

In [5]:

```
# preprocess the dataset
df['Date'] = pd.to_datetime(df['Date'])
```

In [6]:

```
df.groupby(df.Date.dt.year)['Dollar_Rate'].mean()
```

Out[6]:

```
Date
1979     8.060000
1980     7.886773
1981     8.680717
1982     9.484741
1983    10.104104
1984    11.346520
1985    12.331840
1986    12.596733
1987    12.946865
1988    13.899522
1989    16.202948
1990    17.490593
1991    22.792908
1992    29.575958
1993    31.438504
1994    31.372985
1995    32.405908
```

In [7]:

```
df = df.set_index('Date')
df = df.dropna()
```

In [8]:

```
df
```

Out[8]:

	Dollar_Rate
Date	
1979-12-26	8.140
1979-12-27	8.100
1979-12-28	8.000
1979-12-31	8.000
1980-01-02	8.000
...	...
2022-11-27	81.410
2022-12-04	82.410
2022-12-11	82.706
2022-12-18	82.780
2022-12-25	82.717

8392 rows × 1 columns

In [9]:

```
# import matplotlib.pyplot as plt

# # Plot a line chart
# plt.plot( df['Dollar_Rate'],color="r")
# plt.xlabel('Dates')
# plt.ylabel('Dollar_Rate')
# plt.title('Dataset after Preprocessing')
# plt.show()
```

In [10]:

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

# Generate a heatmap to visualize missing values
sns.heatmap(df.isnull(), cmap='viridis')

# Display the plot
plt.title('Missing Values')
plt.show()
```

In [11]:

```
# create train and test data
train_data = df[:int(0.8*len(df))]
test_data = df[int(0.8*len(df)):]
```

In [12]:

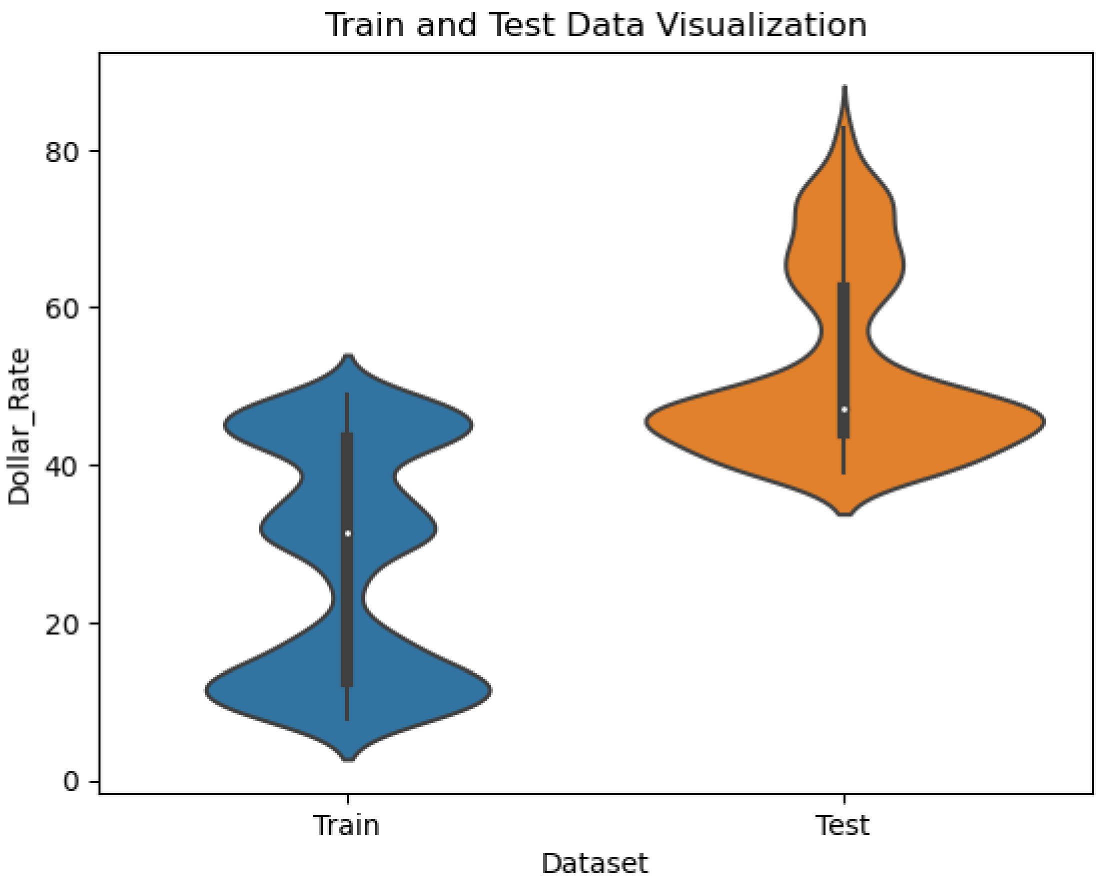
```
train_data.shape,test_data.shape
```

Out[12]:

```
((6713, 1), (1679, 1))
```

In [13]:

```
# Combine train and test datasets into a single DataFrame
combined_data = pd.concat([train_data, test_data], keys=['Train', 'Test'])

# Plot violin plot for a specific variable
sns.violinplot(x=combined_data.index.get_level_values(0), y=combined_data['Dollar_Rate']
plt.xlabel('Dataset')
plt.ylabel('Dollar_Rate')
plt.title('Train and Test Data Visualization')
plt.show()
```



In [14]:

```
train_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 6713 entries, 1979-12-26 to 2006-02-28
Data columns (total 1 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   Dollar_Rate  6713 non-null   float64
dtypes: float64(1)
memory usage: 104.9 KB
```

In [15]:

```
# scale the data
scaler = MinMaxScaler()
train_data_scaled = scaler.fit_transform(train_data)
test_data_scaled = scaler.transform(test_data)
```

In [16]:

```
# split data into X and y
X_train = []
y_train = []
for i in range(12, len(train_data)):
    X_train.append(train_data_scaled[i-12:i, :])
    y_train.append(train_data_scaled[i, 0])
X_train, y_train = np.array(X_train), np.array(y_train)

X_test = []
y_test = []
for i in range(12, len(test_data)):
    X_test.append(test_data_scaled[i-12:i, :])
    y_test.append(test_data_scaled[i, 0])
X_test, y_test = np.array(X_test), np.array(y_test)
```

In [17]:

```
from tensorflow.keras.layers import LSTM, Dense, Bidirectional
```

In [18]:

```
model = Sequential()
model.add(Bidirectional(LSTM(units=50, activation='relu'), input_shape=(X_train.shape[1]
model.add(Dense(units=1))
model.compile(optimizer='adam', loss='mean_squared_error')
```

In [19]:

```
# train the LSTM model
model.fit(X_train, y_train, epochs=10, batch_size=16)
```

```
Epoch 1/10
419/419 [==============================] - 24s 14ms/step - loss: 0.0089
Epoch 2/10
419/419 [==============================] - 3s 8ms/step - loss: 4.2196e-05
Epoch 3/10
419/419 [==============================] - 3s 7ms/step - loss: 3.9313e-05
Epoch 4/10
419/419 [==============================] - 3s 7ms/step - loss: 4.1090e-05
Epoch 5/10
419/419 [==============================] - 3s 8ms/step - loss: 4.0266e-05
Epoch 6/10
419/419 [==============================] - 5s 11ms/step - loss: 4.0678e-05
Epoch 7/10
419/419 [==============================] - 7s 18ms/step - loss: 4.1769e-05
Epoch 8/10
419/419 [==============================] - 9s 22ms/step - loss: 4.1267e-05
Epoch 9/10
419/419 [==============================] - 7s 16ms/step - loss: 3.8582e-05
Epoch 10/10
419/419 [==============================] - 6s 14ms/step - loss: 3.8643e-05
```

Out[19]:

```
<keras.callbacks.History at 0x28ede6b9550>
```

In [20]:

```
# make predictions on test data
#y_pred = model.predict(X_test)
```

In [21]:

```
# Evaluate the model on test data
#mse= model.evaluate(X_test, y_test)
#print(f"Mean squared error on test data: {mse:.4f}")

# Make predictions on test data
y_pred_lstm = model.predict(X_test)
y_pred_lstm = scaler.inverse_transform(y_pred_lstm)
y_test = scaler.inverse_transform(y_test.reshape(-1, 1))
```

```
53/53 [==============================] - 2s 4ms/step
```

In [22]:

```
from sklearn.metrics import r2_score

print("The R2 Score of LSTM model is:",r2_score(y_pred_lstm,y_test)*100)
```

```
The R2 Score of LSTM model is: 98.91733205271798
```

In [23]:

```
from math import sqrt
from sklearn.metrics import mean_squared_error, mean_absolute_error
from math import sqrt
```

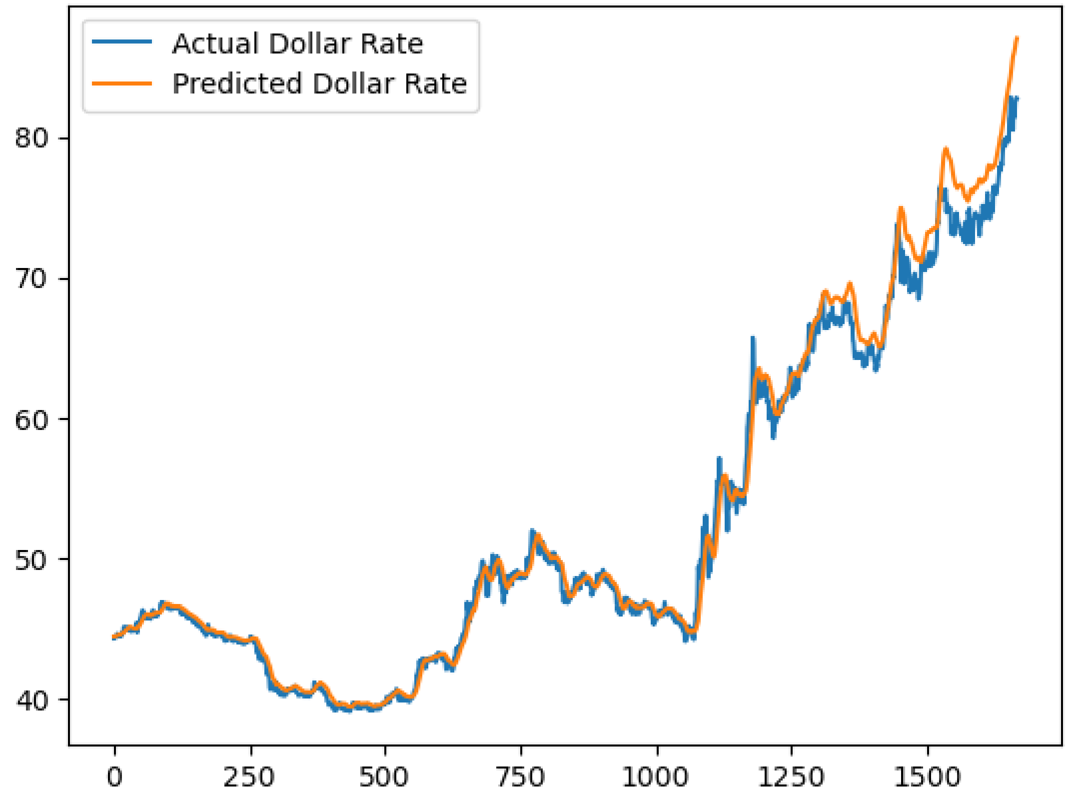
In [24]:

```
# Evaluate the bidirectional LSTM model
mse_bidirectional = mean_squared_error(y_test, y_pred_lstm)
rmse_bidirectional = sqrt(mse_bidirectional)
mae_bidirectional = mean_absolute_error(y_test, y_pred_lstm)

print("\nBidirectional LSTM Model Evaluation:")
print("Mean Squared Error (MSE):", mse_bidirectional)
print("Root Mean Squared Error (RMSE):", rmse_bidirectional)
print("Mean Absolute Error (MAE):", mae_bidirectional)
print("The R2 Score of Bidirectional LSTM Model is:",r2_score(y_pred_lstm,y_test)*100)
```

```
Bidirectional LSTM Model Evaluation:
Mean Squared Error (MSE): 1.6576191513669174
Root Mean Squared Error (RMSE): 1.2874855926832414
Mean Absolute Error (MAE): 0.8414459617720008
The R2 Score of Bidirectional LSTM Model is: 98.91733205271798
```

In [25]:

```
import matplotlib.pyplot as plt

plt.plot(y_test, label='Actual Dollar Rate')
plt.plot(y_pred_lstm, label='Predicted Dollar Rate')
plt.legend()
plt.show()
```



In [26]:

```
test = X_test[-12:][1]
```

In [27]:

```
test = test.reshape(1,12,1)
```

In [28]:

```
pred_values = []
for i in range(5):
    predicted = model.predict(test)
    test = test[:,1:,:]
    test = np.append(test,predicted[-1,-1])
    pred_values.append(predicted[-1,-1])
    test = test.reshape(1,12,1)
```

```
1/1 [==============================] - 0s 19ms/step
1/1 [==============================] - 0s 20ms/step
1/1 [==============================] - 0s 21ms/step
1/1 [==============================] - 0s 19ms/step
1/1 [==============================] - 0s 18ms/step
```

In [29]:

```
pred_values = np.array(pred_values)
pred_values = scaler.inverse_transform(pred_values.reshape(-1,1))
```

In [30]:

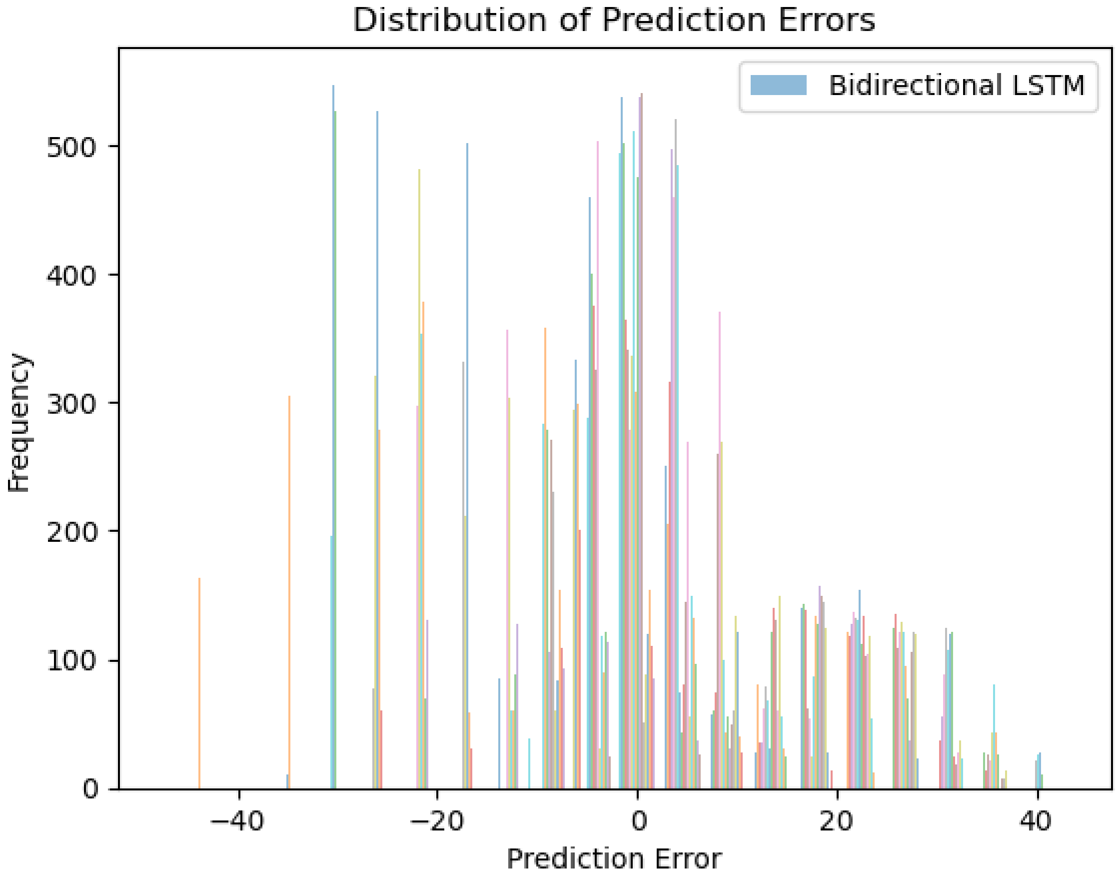
```
pred_values
```

Out[30]:

```
array([[84.83485 ],
       [85.392395],
       [86.04622 ],
       [86.74675 ],
       [87.49947 ]], dtype=float32)
```

In [31]:

```
# Calculate the prediction errors
error_bidirectional = y_test - y_pred_lstm.flatten()
plt.hist(error_bidirectional, bins=20, alpha=0.5, label='Bidirectional LSTM')
plt.xlabel('Prediction Error')
plt.ylabel('Frequency')
plt.title('Distribution of Prediction Errors')
plt.legend()
plt.show()
```



In [26]:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.preprocessing import MinMaxScaler
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, LSTM
from sklearn.metrics import mean_squared_error
```

In [2]:

```
import pandas as pd
df=pd.read_csv(r"C:\Users\arunj\Desktop\Time Series Project\jupiter notebook\DollarFullD
df
```

Out[2]:

	Date	Dollar_Rate
0	Dec 25, 2022	82.717
1	Dec 18, 2022	82.780
2	Dec 11, 2022	82.706
3	Dec 04, 2022	82.410
4	Nov 27, 2022	81.410
...	...	...
8387	Jan 02, 1980	8.000
8388	Dec 31, 1979	8.000
8389	Dec 28, 1979	8.000
8390	Dec 27, 1979	8.100
8391	Dec 26, 1979	8.140

8392 rows × 2 columns

In [3]:

```
df['Date'] = pd.to_datetime(df['Date'])
df.groupby(df.Date.dt.year)['Dollar_Rate'].mean()
```

Out[3]:

```
Date
1979     8.060000
1980     7.886773
1981     8.680717
1982     9.484741
1983    10.104104
1984    11.346520
1985    12.331840
1986    12.596733
1987    12.946865
1988    13.899522
1989    16.202948
1990    17.490593
1991    22.792908
1992    29.575958
1993    31.438504
1994    31.372985
1995    32.405908
1996    35.372609
1997    36.322454
1998    41.274531
1999    43.058154
2000    44.938262
2001    47.175519
2002    48.570885
2003    46.542115
2004    45.242332
2005    44.051212
2006    45.197077
2007    41.187318
2008    43.418168
2009    48.293410
2010    45.994000
2011    46.654615
2012    53.368396
2013    58.618404
2014    61.012288
2015    64.160596
2016    67.199404
2017    65.020358
2018    68.475212
2019    70.384654
2020    74.166115
2021    73.919404
2022    78.676904
Name: Dollar_Rate, dtype: float64
```

In [4]:

```
df = df[::-1]
```

In [5]:

```
# preprocess the dataset

df = df.set_index('Date')
df = df.dropna()
```

In [6]:

```
# create train and test data
train_data = df[:int(0.8*len(df))]
test_data = df[int(0.8*len(df)):]
```

In [7]:

```
# scale the data
scaler = MinMaxScaler()
train_data_scaled = scaler.fit_transform(train_data)
test_data_scaled = scaler.transform(test_data)
```

In [8]:

```
# split data into X and y
X_train = []
y_train = []
for i in range(12, len(train_data)):
    X_train.append(train_data_scaled[i-12:i, :])
    y_train.append(train_data_scaled[i, 0])
X_train, y_train = np.array(X_train), np.array(y_train)

X_test = []
y_test = []
for i in range(12, len(test_data)):
    X_test.append(test_data_scaled[i-12:i, :])
    y_test.append(test_data_scaled[i, 0])
X_test, y_test = np.array(X_test), np.array(y_test)
```

In [9]:

```
# build the LSTM model
model = Sequential()
model.add(LSTM(units=50, return_sequences=True, input_shape=(X_train.shape[1], X_train.s
model.add(LSTM(units=50, return_sequences=False))
model.add(Dense(units=1))
model.compile(optimizer='adam', loss='mean_squared_error')
```

In [10]:

```
# train the LSTM model
model.fit(X_train, y_train, epochs=10, batch_size=16)
```

```
Epoch 1/10
419/419 [==============================] - 7s 9ms/step - loss: 0.0045
Epoch 2/10
419/419 [==============================] - 9s 22ms/step - loss: 4.5883e-05
Epoch 3/10
419/419 [==============================] - 7s 16ms/step - loss: 4.5019e-05
Epoch 4/10
419/419 [==============================] - 8s 19ms/step - loss: 4.6552e-05
Epoch 5/10
419/419 [==============================] - 5s 13ms/step - loss: 4.6750e-05
Epoch 6/10
419/419 [==============================] - 6s 13ms/step - loss: 4.5921e-05
Epoch 7/10
419/419 [==============================] - 7s 18ms/step - loss: 4.9431e-05
Epoch 8/10
419/419 [==============================] - 13s 32ms/step - loss: 5.3674e-0
5
Epoch 9/10
419/419 [==============================] - 10s 24ms/step - loss: 4.9522e-0
5
Epoch 10/10
419/419 [==============================] - 7s 16ms/step - loss: 5.7030e-05
```

Out[10]:

```
<keras.callbacks.History at 0x2bc5d1cc880>
```

In [11]:

```
#y_pred = model.predict(X_test)
```

In [12]:

```
#mse= model.evaluate(X_test, y_test)
#print(f"Mean squared error on test data: {mse:.4f}")

y_pred_lstm = model.predict(X_test)
y_pred_lstm = scaler.inverse_transform(y_pred_lstm)
y_test = scaler.inverse_transform(y_test.reshape(-1, 1))
```

```
53/53 [==============================] - 1s 4ms/step
```

In [13]:

```
from math import sqrt
from sklearn.metrics import mean_squared_error, mean_absolute_error
from math import sqrt
from sklearn.metrics import r2_score
```

In [14]:

```
# Evaluate the bidirectional LSTM model
mse_unidirectional = mean_squared_error(y_test, y_pred_lstm)
rmse_unidirectional = sqrt(mse_unidirectional)
mae_unidirectional = mean_absolute_error(y_test, y_pred_lstm)

print("\nUnidirectional LSTM Model Evaluation:")
print("Mean Squared Error (MSE):", mse_unidirectional)
print("Root Mean Squared Error (RMSE):", rmse_unidirectional)
print("Mean Absolute Error (MAE):", mae_unidirectional)
print("The R2 Score of Unidirectional LSTM Model is:",r2_score(y_pred_lstm,y_test)*100)
```

```
Unidirectional LSTM Model Evaluation:
Mean Squared Error (MSE): 3.7564480231343644
Root Mean Squared Error (RMSE): 1.9381558304569746
Mean Absolute Error (MAE): 1.3506990160195504
The R2 Score of Unidirectional LSTM Model is: 96.58197612556077
```

In [15]:

```
from sklearn.metrics import r2_score

print("The R2 Score of LSTM model is:",r2_score(y_pred_lstm,y_test)*100)
```

```
The R2 Score of LSTM model is: 96.58197612556077
```

In [16]:

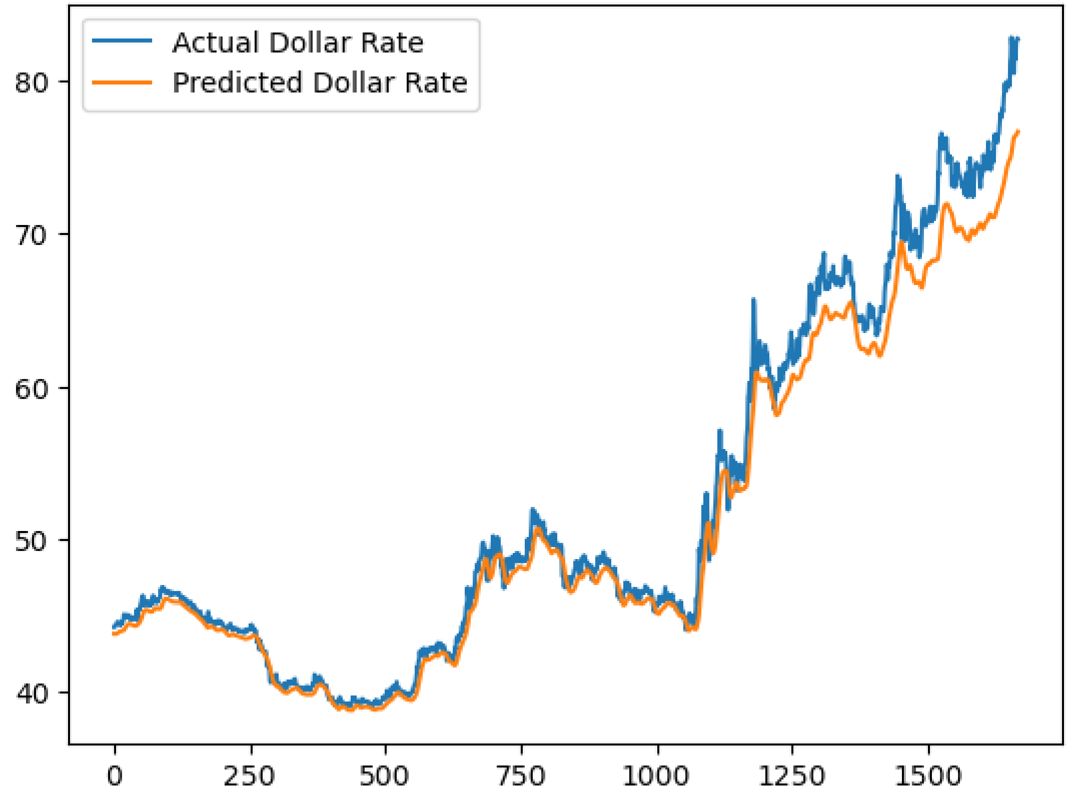
```
X_test.shape
```

Out[16]:

```
(1667, 12, 1)
```

In [17]:

```
import matplotlib.pyplot as plt

plt.plot(y_test, label='Actual Dollar Rate')
plt.plot(y_pred_lstm, label='Predicted Dollar Rate')
plt.legend()
plt.show()
```



In [18]:

```
test = X_test[-12:][1]
```

In [19]:

```
test = test.reshape(1,12,1)
```

In [20]:

```
pred_values = []
for i in range(5):
    predicted = model.predict(test)
    test = test[:,1:,:]
    test = np.append(test,predicted[-1,-1])
    pred_values.append(predicted[-1,-1])
    test = test.reshape(1,12,1)
```

```
1/1 [==============================] - 0s 27ms/step
1/1 [==============================] - 0s 24ms/step
1/1 [==============================] - 0s 25ms/step
1/1 [==============================] - 0s 26ms/step
1/1 [==============================] - 0s 24ms/step
```

In [21]:

```
pred_values = np.array(pred_values)
pred_values = scaler.inverse_transform(pred_values.reshape(-1,1))
```

In [22]:

```
pred_values
```

Out[22]:

```
array([[75.668236],
       [75.45476 ],
       [75.08404 ],
       [74.60692 ],
       [74.06478 ]], dtype=float32)
```

In [23]:

```
error_unidirectional = y_test - y_pred_lstm.flatten()
```

In [24]:

```
# Calculate the prediction errors


plt.hist(error_unidirectional, bins=20, alpha=0.5, label='Unidirectional LSTM')
plt.xlabel('Prediction Error')
plt.ylabel('Frequency')
plt.title('Distribution of Prediction Errors')
plt.legend()
plt.show()
```