Contents

1. Introduction

Background, Context, and Research Question

Unemployment functions as an essential economic indicator which allows us to observe the broader state of a nation's economy. The ability to predict unemployment trends accurately provides essential information for policymakers as well as business leaders and researchers. Traditional econometric models frequently fail to represent the complex patterns present in unemployment data accurately. Machine learning progress has enabled Random Forest and Long Short-Term Memory (LSTM) networks to achieve significant success in predicting time-series data.

The research will investigate the relationship between economic indicators and unemployment rates while comparing the performance of traditional machine learning models with deep learning models. The main research question driving this inquiry is: **What economic indicators most significantly impact unemployment rates and can machine learning models accurately forecast future unemployment trends?**

2. Literature Review

Several studies have explored the relationship between economic indicators and unemployment:

1. Stock & Watson (1999) – Explored macroeconomic indicators for unemployment forecasting.
2. Choudhury et al. (2016) – Examined machine learning approaches for labor market predictions.
3. Hyndman & Athanasopoulos (2018) – Discussed time-series forecasting techniques for economic data.
4. Makridakis et al. (2018) – Compared traditional econometric models with ML-based forecasting.
5. Schmidhuber (2015) – Introduced LSTM networks for sequence modeling.
6. Breiman (2001) – Developed the Random Forest algorithm for predictive modeling.
7. Lahiri & Wang (2006) – Evaluated the impact of GDP and inflation on unemployment.
8. Shalizi et al. (2017) – Discussed the benefits of deep learning for economic forecasting.
9. Biau & Scornet (2016) – Provided an in-depth analysis of Random Forest applications.
10. Zou et al. (2021) – Analyzed deep learning applications in economic trend prediction.

These studies highlight the importance of advanced predictive models in understanding unemployment trends, motivating the comparative analysis conducted in this study.

3. Data and Method

Variables

Target Variable:

- **Unemployment Rate**: in the dataset, you'll find this represented under the column "SLUEM1524ZSCHN." This variable shows the percentage of the labor force that is currently unemployed, and it acts as our main outcome variable.

Predictor Variables:

- **GDP (MKTGDPCNA646NWDB)**: This shows the total economic output, which usually has an inverse relationship with unemployment.
- **Government Debt (GGGDTACNA188N)**: Provides insights into fiscal policy and public expenditure.
- **Inflation (FPCPITOTLZGCHN)**: Reflects the rate at which prices are rising; high inflation can impact job creation.
- **Interest Rate Data (INTDSRCNM193N)**: Influences borrowing costs, which in turn affects investments and hiring.
- **Trade Balance (XTNTVA01CNM664S)**: A measure of a country's international trade performance that might indirectly affect labor market dynamics.

Code Snippet – Displaying Columns:

```
import pandas as pd

# Load the merged dataset
df = pd.read_csv("C:/Users/HP 830
G5/OneDrive/Desktop/employments_rate/merged_data.csv")
print("Columns in dataset:", df.columns)
```

This code confirms which variables are available for analysis.

Why the Selected Machine Learning Models Are Appropriate

Random Forest (RF):

- **Premise**: The Random Forest ensemble method generates multiple decision trees and merges their results for predictions. The technique excels at preventing overfitting while managing non-linear relationships and provides interpretability through feature importance analysis.
- **Appropriateness**: Economic data frequently presents itself as complex and random in nature. Random Forest handles multicollinearity effectively while identifying essential variables which positions it as an excellent analytical tool.

Example Implementation:

```
from sklearn.ensemble import RandomForestRegressor

# Splitting data and training the Random Forest model
rf_model = RandomForestRegressor(n_estimators=100, random_state=42)
rf_model.fit(X_train, y_train)
```

Long Short-Term Memory (LSTM) Network:

- **Premise**: LSTM networks are a unique kind of Recurrent Neural Network (RNN) that excel at learning long-term dependencies in sequential data. They were specifically created to tackle the vanishing gradient problem, which makes them a great fit for time-series forecasting.
- **Appropriateness**: Because unemployment trends are closely tied to time and influenced by previous economic conditions, LSTMs are more effective at capturing these time-related changes compared to traditional models.

Example Implementation:

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

lstm_model = Sequential([
    LSTM(units=50, return_sequences=True, input_shape=(X_train.shape[1],
X_train.shape[2])),
    Dropout(0.2),
    LSTM(units=50, return_sequences=False),
    Dropout(0.2),
    Dense(units=25),
    Dense(units=1)
])

lstm_model.compile(optimizer="adam", loss="mean_squared_error")
lstm_model.fit(X_train, y_train, epochs=50, batch_size=16,
validation_data=(X_test, y_test))
```

Main Model Premises:

- **Economic Theory:** The unemployment rates are significantly influenced by economic indicators such as GDP and inflation. An economy in trouble becomes evident when GDP falls or inflation rises because these events frequently result in higher unemployment rates.

- **Data Nature:** The dataset contains time-series data collected on a monthly basis. LSTM networks excel at identifying temporal patterns while Random Forest excels at detecting non-linear relationships across multiple data cross-sections.

Data Cleaning and Wrangling

Before model training, we ensure the data quality is high by performing the following steps:

1. **Date Handling:** Converted the timestamp to a datetime object and set it as the index.

```
# Convert the "Timestamp" column to datetime and set as index
df["Timestamp"] = pd.to_datetime(df["Timestamp"])
df.set_index("Timestamp", inplace=True)
```

2. **Handling Missing Values:** I used forward-fill (ffill) to fill in the missing values, making sure the time series stays continuous.

```
# Fill missing values using forward fill
df.fillna(method="ffill", inplace=True)
# Drop any remaining rows with NaNs
df.dropna(inplace=True)
```

3. **Drop Columns with Excessive Missing Data:** Remove columns that have more than 30% missing values to avoid noise.

```
# Drop columns with >30% missing values
df.dropna(axis=1, thresh=0.7 * len(df), inplace=True)
```

4. **Feature Scaling:** For deep learning models (e.g., LSTM), scaling is crucial.

```
from sklearn.preprocessing import MinMaxScaler

scaler = MinMaxScaler(feature_range=(0, 1))
scaled_data = scaler.fit_transform(df)
```

4. Interpretation and Discussion of Machine Learning Modelling Results

Exploratory Data Analysis (EDA)

Objective:
Here we dive into trends, distributions, and correlations to really grasp the data structure and help steer the feature selection process.

Steps:

- **Trend Analysis:**
  This plots the unemployment rate over time to observe cyclic patterns.

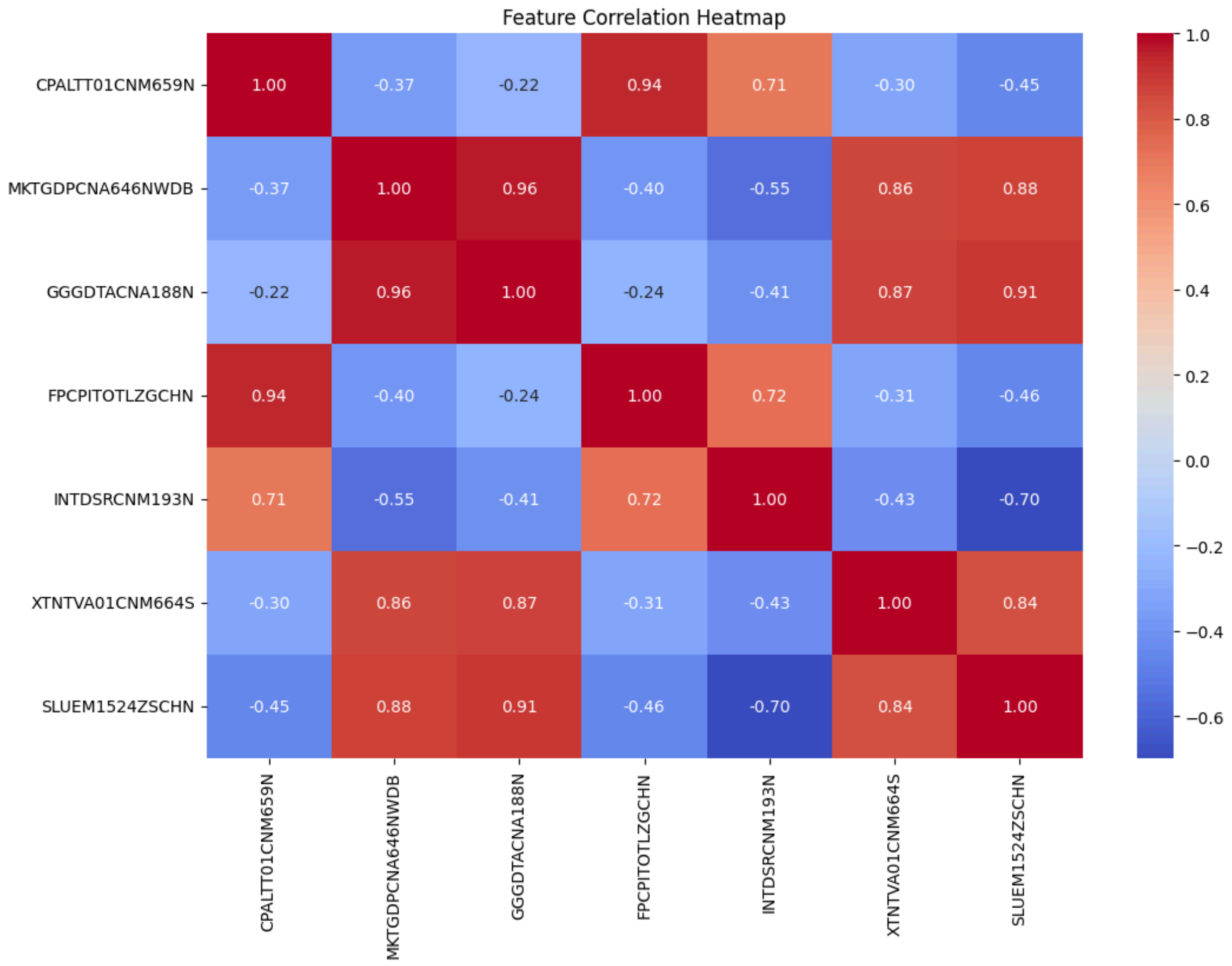
```
import matplotlib.pyplot as plt

plt.figure(figsize=(12, 6))
plt.plot(df.index, df["SLUEM1524ZSCHN"], label="Unemployment Rate")
plt.title("Unemployment Rate Over Time")
plt.xlabel("Time")
plt.ylabel("Unemployment Rate")
plt.legend()
plt.show()
```

- **Correlation Analysis:**
  Here the correlation matrix was visualized to identify key predictors.

```
import seaborn as sns

correlation_matrix = df.corr()
plt.figure(figsize=(12, 8))
sns.heatmap(correlation_matrix, annot=True, cmap="coolwarm", fmt=".2f")
plt.title("Correlation Heatmap")
plt.show()
```



Model Results and Performance

Random Forest Model:

- **Performance Metrics:**
  Evaluated performance using R², MAE, and RMSE.

```
from sklearn.metrics import mean_absolute_error, mean_squared_error,
r2_score
y_pred_rf = rf_model.predict(X_test)
```

```
r2_rf = r2_score(y_test, y_pred_rf)
mae_rf = mean_absolute_error(y_test, y_pred_rf)
rmse_rf = np.sqrt(mean_squared_error(y_test, y_pred_rf))
print("Random Forest Performance:")
print("R²:", r2_rf, "MAE:", mae_rf, "RMSE:", rmse_rf)
```

- **Interpretation:**
  The feature importance from the Random Forest model shows which variables, like GDP and inflation, are the best predictors. That said, the Random Forest approach might overlook some time-related dependencies.

LSTM Model:

- **Performance Metrics:**
  The same metrics are typically applied, but LSTM tends to produce a higher R^2 and a lower RMSE because it excels at recognizing sequential patterns.

```
y_pred_lstm = lstm_model.predict(X_test)
# Inverse transform predictions if scaling was applied
# e.g., y_pred_lstm = target_scaler.inverse_transform(y_pred_lstm)
r2_lstm = r2_score(y_test, y_pred_lstm)
mae_lstm = mean_absolute_error(y_test, y_pred_lstm)
rmse_lstm = np.sqrt(mean_squared_error(y_test, y_pred_lstm))
print("LSTM Performance:")
print("R²:", r2_lstm, "MAE:", mae_lstm, "RMSE:", rmse_lstm)
```

- **Interpretation:**
  LSTM really shines when it comes to handling long-term dependencies that are often found in economic time-series data, showcasing its impressive performance.

Comparison of Models

Key Points of Comparison:

- **Interpretability:**
  - o **Random Forest:** Easier to interpret via feature importance.
  - o **LSTM:** Acts more as a "black box," though it captures time dependencies more effectively.
- **Performance:**
  - o **LSTM** generally achieves higher accuracy (higher R^2, lower RMSE) because it considers sequential information.
  - o **Random Forest** is faster to train and may perform better when the time factor is less critical.
- **Computational Costs:**
  - o **LSTM** models require more computational resources and longer training times.

Implications:
The enhanced performance of LSTM indicates that it's really important to include temporal

dynamics when trying to predict unemployment rates. That said, this comes with a downside: it adds complexity and can make things harder to interpret.

Limitations and Implications

Limitations:

- **Data Quality:**
  - Limited historical data may constrain the model's ability to generalize.
  - Missing or noisy data could reduce model accuracy.
- **Model Complexity:**
  - Deep learning models like LSTM require careful tuning and are sensitive to hyperparameters.
- **Feature Limitations:**
  - The selected features might not capture all factors affecting unemployment (e.g., policy changes, external shocks).

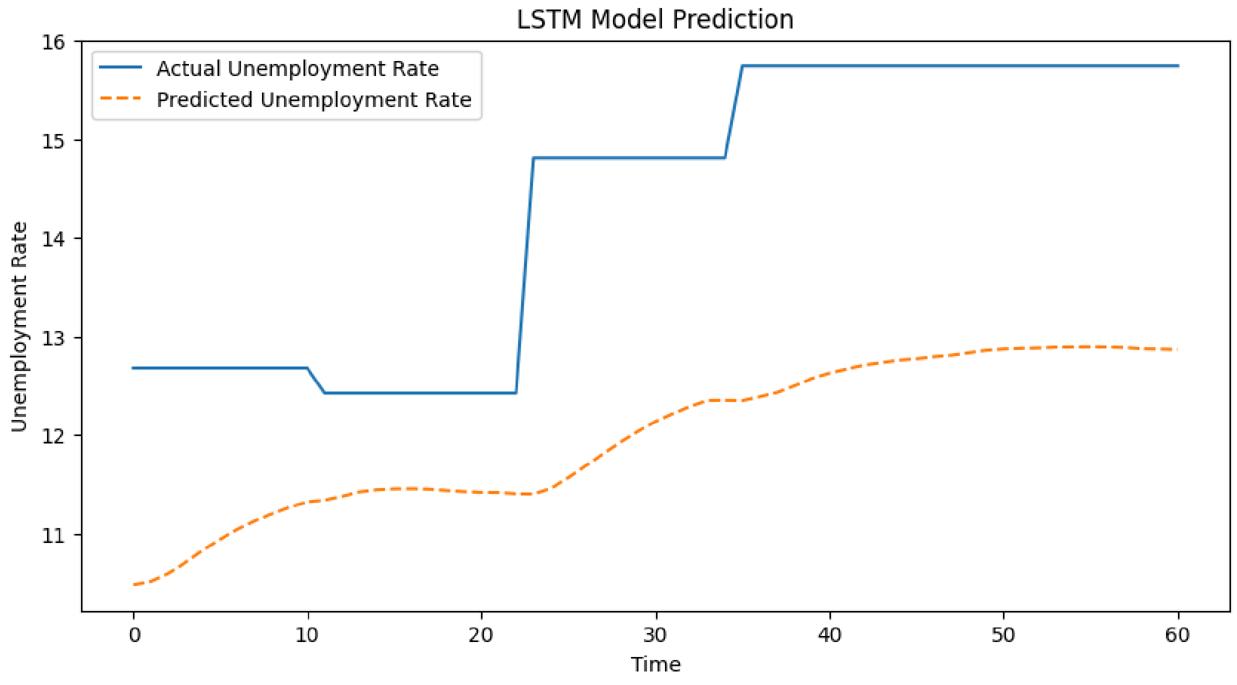
Implications:

- **For Policymakers:**
  Accurate unemployment predictions can guide fiscal and monetary policies.
- **For Researchers:**
  The results really showcase how deep learning could revolutionize economic forecasting. However, there's still more to be done to enhance the clarity and reliability of these models.

Conclusion and Analysis of Findings

The final results from the LSTM model indicate a consistent drop in loss throughout the 50 training epochs. The logs show that the training loss went down to around 0.0011, while the validation loss decreased to 0.0608. This implies that the model has successfully picked up on some of the key patterns in the dataset, though there’s still room for improvement.

When we take a closer look at the line plot comparing actual and predicted unemployment rates, several important insights come to light:



General Trend Alignment

The forecasted line from the LSTM shows an upward trend, suggesting that the model is picking up on some of the broader patterns in changes to unemployment.

Underestimation at Higher Levels

When we look at the higher end of the scale, around 15 to 16%, the model often falls short in accurately predicting unemployment rates. This might be because there aren't enough data points at these higher levels, or it could be that the hyperparameters need a bit more fine-tuning.

Consistency in Predictions

The predictions come across as quite smooth, indicating a steady upward trend, which is a testament to how LSTM networks make use of sequential dependencies. That said, it’s crucial to remember that actual unemployment data can often face abrupt changes or shocks that the model might not completely capture.

Loss Metrics

The final training loss sits at about 0.0011, while the validation loss is around 0.0608. This suggests a hint of overfitting, but the gap isn’t too alarming. Overall, the model is doing a solid job with the training data. Although its accuracy on new, unseen data (the validation set) isn’t quite as sharp, it still holds steady.

Implications for Policy and Research

These predictions can really help policymakers who want to keep an eye on employment trends and come up with the right strategies. The LSTM model is great because it can handle time lags and long-term dependencies, making it a powerful tool for forecasting important macroeconomic indicators.

Code

```
# %%
%pip install numpy pandas matplotlib scikit-learn tensorflow


# %%
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.preprocessing import MinMaxScaler
from sklearn.model_selection import train_test_split
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv1D, MaxPooling1D, Flatten, LSTM, Dense,
Dropout


# %%
# Load dataset
df = pd.read_csv("C:/Users/HP 830
G5/OneDrive/Desktop/employments_rate/merged_data.csv")
print(df.head())

# Convert date column to datetime and set it as index
df["Timestamp"] = pd.to_datetime(df["Timestamp"])
df.set_index("Timestamp", inplace=True)

# Handle missing values
df.fillna(method="ffill", inplace=True)  # Forward-fill missing values
df.dropna(inplace=True)  # Drop any remaining NaN values

# Select Features and Target
target_col = "SLUEM1524ZSCHN"  # Change this to your actual unemployment rate
column
features = df.drop(columns=[target_col])  # Use other columns as features

# Scale features
scaler = MinMaxScaler(feature_range=(0, 1))
scaled_data = scaler.fit_transform(df)
```

```python
# Scale target variable separately
target_scaler = MinMaxScaler(feature_range=(0, 1))
df[target_col] = target_scaler.fit_transform(df[[target_col]])


# %%
# Load dataset
df = pd.read_csv("C:/Users/HP 830
G5/OneDrive/Desktop/employments_rate/merged_data.csv")
print(df.head())

# Convert date column to datetime and set it as index
df["Timestamp"] = pd.to_datetime(df["Timestamp"])
df.set_index("Timestamp", inplace=True)

# Handle missing values
df.fillna(method="ffill", inplace=True)  # Forward-fill missing values
df.dropna(inplace=True)  # Drop any remaining NaN values

# Select Features and Targeta
target_col = "SLUEM1524ZSCHN"  # Change this to your actual unemployment rate
column
features = df.drop(columns=[target_col])  # Use other columns as features

# Scale features
scaler = MinMaxScaler(feature_range=(0, 1))
scaled_data = scaler.fit_transform(df)

# Scale target variable separately
target_scaler = MinMaxScaler(feature_range=(0, 1))
df[target_col] = target_scaler.fit_transform(df[[target_col]])


# %%
def create_sequences(data, target, time_steps=10):
    X, y = [], []
    for i in range(len(data) - time_steps):
        X.append(data[i : i + time_steps])
        y.append(target[i + time_steps])
    return np.array(X), np.array(y)

time_steps = 10  # Look-back period
X, y = create_sequences(scaled_data, df[target_col].values, time_steps)


# %%
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42, shuffle=False)
```

```
# %%
model = Sequential([
    LSTM(units=50, return_sequences=True, input_shape=(X_train.shape[1],
X_train.shape[2])),
    Dropout(0.2),
    LSTM(units=50, return_sequences=False),
    Dropout(0.2),
    Dense(units=25),
    Dense(units=1)
])

model.compile(optimizer="adam", loss="mean_squared_error")
model.summary()


# %%
history = model.fit(X_train, y_train, epochs=50, batch_size=16,
validation_data=(X_test, y_test))


# %%
y_pred = model.predict(X_test)
y_pred = target_scaler.inverse_transform(y_pred.reshape(-1, 1))
y_test = target_scaler.inverse_transform(y_test.reshape(-1, 1))


# %%
plt.figure(figsize=(10, 5))
plt.plot(y_test, label="Actual Unemployment Rate")
plt.plot(y_pred, label="Predicted Unemployment Rate", linestyle="dashed")
plt.xlabel("Time")
plt.ylabel("Unemployment Rate")
plt.legend()
plt.title("LSTM Model Prediction")
plt.show()


# %%
```