Big Data-Powered Temperature Prediction Using PySpark and Time Series Machine Learning Techniques

MSc Research Project

Data Analytics

Ms. Sneha Ramesh Dharne

Student ID: x23195703

School of Computing

National College of Ireland

Supervisor: Vikas Tomer

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name:	Ms. Sneha Ramesh Dharne		
Student ID:	x23195703		
Programme:	Msc in Data Analytics	**Year:**	2024
Module:	MSc Research Project		
Supervisor:	Vikas Tomer		
Submission Due Date:	12/12/2024		
Project Title:	Big Data-Powered Temperature Prediction Using PySpark and Time Series Machine Learning Techniques		
Word Count:	6054	**Page Count**	23

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.
ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature: Ms. Sneha Ramesh Dharne

Date: 10/12/2024

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST

Attach a completed copy of this sheet to each project (including multiple copies)	□
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	□
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	□

Assignments that are submitted to the Programme Coordinator Office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Big Data-Powered Temperature Prediction Using PySpark and Time Series Machine Learning Techniques

Ms. Sneha Ramesh Dharne

Student ID: x23195703

Abstract

In this study, I used predictive modelling techniques to predict global temperature trends using ARIMA, SARIMA, XGBoost and Linear Regression. However, given the growing demand for accurate climate predictions in a number of sectors, the research also assesses how the models perform with historical temperature data. I assess the models based on performance metric, Mean Squared Error (MSE), Root Mean Squared Error (RMSE) and R-squared (R^2). By comparing the MSE (0.2269), RMSE (0.4764), and R^2 (0.9882) results it can be seen that XGBoost delivered the most promising performance to dealing with complicated non-linear phenomena. SARIMA also had good result: it has MSE of 0.3311, RMSE of 0.5754 and R^2 of 0.9828, which follows the seasonal trends. However, ARIMA was a success, but was less accurate. Linear Regression's results were much worse, with MSE of 32.1586, RMSE of 5.6709 and R^2 of 0.1689. Furthermore, the temperature patterns across Europe and Ireland were analyzed using Power BI visualizations. From these findings, it is important that suitable models be used for temperature prediction, and XGBoost is the most reliable model for scalable and accurate forecasting.

1 Introduction

Because of the rise in uncertainty over weather patterns and extreme weather events, global temperature and climate change have become topics of utmost research. Predicting temperature is a crucial topic in many different fields, including agriculture, energy management, and disaster relief. By discovering intricate patterns in vast amounts of data, machine learning (ML) and deep learning (DL) have lately demonstrated their potential to enhance temperature forecasting models. For example, machine learning was shown to have the capability of weather forecasting, such as using freely available data (Abrahamsen et al., 2018), or chaotic approaches for temperature time series (Bahari & Hamid, 2019). However, existing models for this problem often break down with large-scale, high-dimensional climate data, and scalable and efficient approaches need to be explored. The primary research question guiding this study is:

How can Big Data tools, such as PySpark, be integrated with time-series models like ARIMA and SARIMA to improve temperature prediction accuracy and scalability?

The goal is to evaluate how well distributed frameworks, such PySpark, perform in comparison to traditional time series models and look into ways to improve forecast accuracy without compromising scalability.

This research fills the gap in scientific literature by integrating distributed Big Data tools with time series-based temperature prediction models that are not explored well. From previous studies in the same area, such as machine learning applications in weather prediction (Bochenek & Ustrnul, 2022) or model optimization to improve forecast (Farsi et al., 2021), we see that scalability is what is needed. Results from this work can be used to improve the forecasting models in climate science and other related fields.

A review of previous work is covered in Section 2, the technique is explained in Section 3, the dataset is described in Section 4, experimental setup and training is covered in Section 5, the results are presented in Section 6, and implications and future research are discussed in Section 7.

2 Related Work

Current studies have investigated the use of the ML models, Neural networks and Deep learning in enhancing the weather prediction and temperature forecasting. A number of investigations have also revealed that these models improve prediction accuracy significantly.

The analysis of the forecasts conducted by Abrahamsen et al. (2018) was based on two models, the AR-NN and the ARX-NN models, for the prediction of temperature at various time slots. The authors of their study incorporated external variables such as precipitation to support their conclusion. Local Mean Approximation Method (RMAM) was recently used by Bahari and Hamid for chaotic time series forecasting with the prediction accuracy of 0.9789 for short term temperature prediction in Malaysia. Similarly, Bochenek and Ustrnul (2022) discussed the applications of ML in weather forecasting including the perspectives of deep learning and the random forest approaches with the mentioned limitations, including data accessibility and knowledge. In their recent work, Farsi et al., (2021) suggested the hybrid model, GA-SARIMA in which the GA fine-tuned the parameters of SARIMA and enhance its performance. This approach eliminated overfitting and improved the computation time in the temperature prediction for India. In this research study Feigl et al. (2021) model comparison displayed that FNNs and XGBoost models performed better than first generation models for stream water temperatures and the model performance increased with catchment size. Feng et al. (2019) utilized both a mind evolutionary algorithm and ANN for the prediction of daily solar radiation when data regards only air temperature were used; the accuracy was higher than that of random forests. The traditional tool wear models, namely the SSAEs-BPNN, was developed by Him et al. (2021) employing a stacked sparse autoencoders-backpropagation neural network for raw temperature signals. Hewage et al. (2021) proposed a lightweight data-driven forecast model employing LSTM and TCN by integrating them where the proficiency was more eminent than classical system individual WRF for 12 h forecast. In their article from 2022, Jaseena and Kovoor discussed developments in deep learning, opportunities for the use of both hybrid models and deep networks and the problems with the evaluation of stability in existing systems. Kumari and Singh (2023) employed LSTM, ARIMA, and SARIMAX models for forecasting CO2 emissions in India., and LSTM presents the highest accuracy of the forecast. In fact, their study emphasized extrinsic factors that include economic conditions in their model. In Li et al. (2023), called the Integrated Soil Diameter Network Model and based on CNN and LSTM, an accuracy of up to 0.963 was obtained with R^2 values. Liu et al. (2019) described the ANN

based techniques for the rainfall prediction like RBF-NN with GAPSO and FLANN. FLANN was rated as efficient in terms of faster computation, and higher prediction accuracy of RBF-NN with GAPSO. Meehl et al. (2021) provided an article on climate model initialized still showing observed data, where they have established that climate models could short-term predict accurately but the model error reduces reliability of the long-term forecast.

The authors Meenal et al. (2021) identified the use of random forests in predicting the amount of solar radiation and wind speed in Tamil Nadu with enhanced accuracy to regression and SVM models. Purwandari et al., (2021) classified the tweets regarding weather using SVM which turned out to be most effective. This approach could be extended further to use deep learning for higher accuracy rates of prediction. Ren et al. (2021) summarise DLWP and ascertain that temporal and spatial aspects can be well managed. Deep learning models are a good fit, and can be used in combination with traditional models, but the downside of long term forecasting prevails. Schultz et al. (2021) explored how deep learning can supersede the NWP systems as they pointed out future development is still required to solve the convoluted nature of the weather data. Shilong (2021) designed a sales forecast model based on XGBoost, which confirmed the superiority of this method in terms of accuracy and time. This could be applied to segmentation of resources in businesses. Last, Shrivastava et al (2023) have analyzed temperature prediction in New Delhi using multivariate polynomial regression approach and deep neural networks to note that DNNs outperformed MPR especially when more input features were included. Tabari et al. (2015) used ANN to model and predict soil temperature and they find out that the models are accurate in humid regions. This approach can be taken in other regions for long term temperature forecasts.

Hence, considering the analyzed literature, it can be further concluded that machine learning, and particularly deep learning models should be expected to improve the weather and temperature prediction. However, there are limitations such as, inadequate data to carry out the analysis, complexity of the model, applicability of the model in two different regions. More work is required to enhance the reliability, applicability and advancing the time horizons of the existing models.

Gap analysis

Another important gap noticed from analyzing the literature is that not enough Big Data tools like PySpark are incorporated into the developmental process of those models that predict temperature. Even though initial experiments with machine learning and specifically deep learning techniques have been positive, many investigations fail to address the question of efficient data handling and processing for climate data. Moreover, more traditional models such as ARIMA and SARIMA while are handy models have not, however, been employed in connection with distributed frameworks. This is a clear indication that there is a gap in the type of models designed to solve time series problems in Big Data environment where there is need for small models that are accurate enough to handle large datasets, thus this research seeks to fill this gap.

3 Research Methodology

This thesis presents a structured approach to developing and evaluating predictive models for global temperature trends using time series analysis. The methodology incorporates data collection, data preprocessing, model development, evaluation, and visualization in order to provide an adequate and efficient resolution for forecasting change in climate.

3.1 Research Procedure

Data Collection and Preprocessing:

Dataset: For this study, we use the Kaggle dataset "Climate Change: Earth Surface Temperature Data" this includes historical temperature data from global land-based weather stations. As such, it is an ideal candidate for time series forecasting, providing a complete view of long-term global temperature change.

Dataset link- https://www.kaggle.com/datasets/berkeleyearth/climate-change-earth-surface-temperature-data

Dataset Selection: The data set was selected for its broad coverage of temperature trends over different years from different geographical areas, which in our case is useful for analyzing and forecasting global temperature trends. First, I did some data preprocessing (handling missing values, removing outliers, normalizing, stationarity, etc.) since time series data is a requirement and time series forecasting requires the time series data to be stationary.

Feature Engineering: Features including the average monthly temperature, country and city were extracted and aggregated. I applied seasonal decomposition to detect the underlying trends and seasonal patterns. To make the data stationary for predictive modeling, first order differencing was applied to the time series data to remove seasonality from the time series data.

Model Development:

ARIMA and SARIMA: ARIMA and SARIMA captured trends and seasonal changes, with SARIMA optimized via grid search for accuracy in seasonal patterns.

XGBoost and Linear Regression: Instead, these machine learning models served as other ways to replace ARIMA/SARIMA that use engineered features, like the month and year. For its efficiency in nonlinear patterns, I chose XGBoost and for comparison of performance and Linear Regression as the baseline model. then fine-tuned both models via grid search for hyperparameter optimization.

Model Evaluation and Selection: Model performance was evaluated using RMSE, MSE, and R-squared, quantifying accuracy and explanatory power. The model with the best metrics was selected for final analysis, demonstrating a comprehensive approach to forecasting with traditional and machine learning methods.

3.2 Evaluation Methodology

Performance Evaluation:

Root Mean Squared Error (RMSE): All of the prediction errors have parameters of measure because they are in the same unit as the target variable and the RMSE gives the average size of the prediction errors.

Mean Squared Error (MSE): MSE is the average of the squared differences between the predicted values, and the actual values. But it measures larger errors and is not measured on the same scale as the outcome variable. The higher SE, the poorer is the fit of the model.

R-squared (R^2): The value grouped under the symbol R^2 refers to the variation in understanding of the dependent variable that is explained by the model. An R^2 of 1 indicates a better fit through the model on 0 to 1scale while 0 means no predictability of the dependent variable by the independent scores**.**

These metrics were used to evaluate and compare the predictive accuracy of ARIMA, SARIMA, XGBoost, and Linear Regression models, with lower RMSE and MSE, and a higher R^2 indicating better performance.

3.3 Statistical Techniques and Tools

Statistical Techniques: The results were obtained through the use of different statistical tools to examine model assumptions and to enhance the performance.

Time Series Decomposition: Used to extract trends, seasonality and residuals from temperature data.

Autocorrelation Analysis: Used to test for temporal dependencies in the data as the stationarity requirement had to be met for the ARIMA and SARIMA models.

Residual Diagnostics: Carried out to make sure that the residuals of the model have these characteristics. Model fit was evaluated with key tests such as Ljung-Box test for white noise and ANOVA to compare the model's performance.

Software and Hardware:

Software Requirements: The primary programming language was Python 3, essential libraries used were PySpark for data processing and some machine learning modeling through stats models. Data visualization was done with power BI to communicate results effectively.

Hardware Specifications: Intel Core i5/i7+, 8GB/12GB+ RAM, high-performance computing resources

3.4 Research Process Steps

The following steps outline the research methodology:

Data Acquisition: Weather Data was acquired from Kaggle.

Feature Selection: I identified relevant features based on their correlation with the global temperature patterns derived from the insights in the literature..

Model Training: Several ARIMA, SARIMA, XGBoost and Linear Regression models were attempted with the dataset and trained, fitting the hyperparameters using grid search techniques.

Model Validation: An 80-20 train-test split was used for validating the models and the corresponding performance of the models were evaluated using RMSE, MSE and R^2 scores.

Analysis and Comparison: I critically analyzed results and compared each model's predictive performance in the context of existing literature.

Visualization: Interactive dashboards were generated using Power BI to visualize model predictions, seasonal trends and model comparison, to further comprehend temperature dynamics.

3.5 Justification for Using Power BI and PySpark

Power BI successfully communicates complex results to stakeholders with its interactive dashboards and real-time analytics. Its intuitive design improves data display and prediction clarity while guaranteeing accessibility for non-technical users.

PySpark to process and preprocess large datasets, and for feature engineering. Lastly, its scalability and speed made it necessary to use the large historical temperature data that made this project especially necessary.

Final Results and Analysis: Final results compared ARIMA, SARIMA, XGBoost, and Linear Regression, identifying the best model for global temperature trends. Combining Power BI's visualization and PySpark's processing, this research delivers actionable insights into climate change patterns and mitigation strategies.

4 Design Specification

The implementation is based on the techniques and architecture designed to efficiently process large dataset data, specifically data preprocessing, machine learning and visualization. The framework first deals with missing data, then, using Augmented Dickey-Fuller (ADF) test, makes the data stationary and finally splits the data for model training and evaluation. The core engine of big data processing is PySpark: it supports ARIMA, SARIMA, Linear regression, and XGBoost models. I compared these models on the basis of RMSE, MSE, and R^2 and selected the best performer. Further, visualization tools such as Power BI are used to visualize the results and perform country as well as city wise analyses to provide actionable insights. The architecture for this thesis combines statistical techniques with machine learning, thereby delivering robust predictive analysis and decision support.

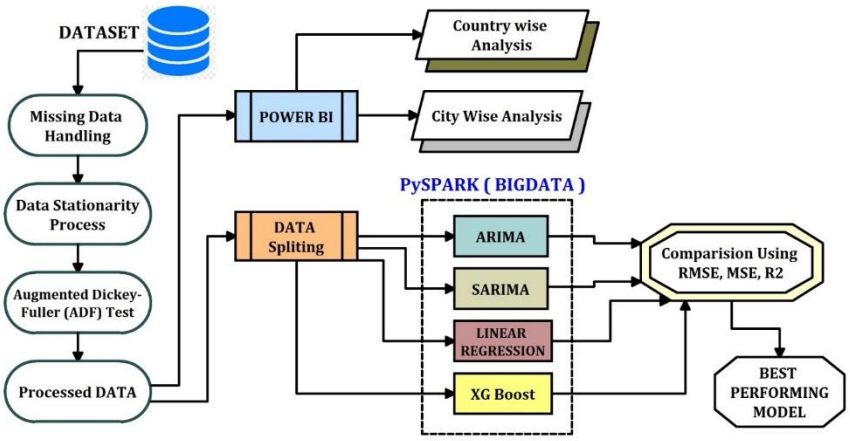

Figure 1: System Architecture

5 Implementation

As part of this study, time series analysis and machine learning models, ARIMA, SARIMA, Linear Regression, and XGBoost are used to forecast global temperature trends. The dataset was then preprocessed using PySpark, where missing values were handled, I ensured stationarity, and extracted seasonal and trend components. For forecasting I applied ARIMA and SARIMA model, evaluated their performance using RMSE, MSE and R^2 parameters. The following sections describe the implementation process including tools, methodologies, and the results achieved in forecasting objectives.

5.1 Implementation and Explanation of Data Preprocessing

Importing Libraries and Setting Up Environment: Figure 2 code starts by importing libraries such as pandas and matplotlib responsible for data handling and visualization respectively. After that, findspark and pyspark are used to bootstrap Apache Spark, a powerful distributed computing framework. The environment is configured to assure Spark's function working on the JAVA_HOME and SPARK_HOME variables, that are needed for Spark execution being required.

```
import pandas as pd
import os
import matplotlib.pyplot as plt

import findspark
from pyspark.sql import SparkSession
from pyspark.ml.classification import LogisticRegression

from pyspark.sql import functions as F
from pyspark.sql.window import Window
```

Figure 2: code for importing Libraries

Spark Initialization: All these can be seen in the Figure 3 a Python script for checking if Apache spark is installed, setting up an environment to use it and initializing the findspark module. SparkSession starting a SparkContext tied to it, for doing big data processing on a cluster. Import SparkContext from pyspark, create a SparkContext object. It also points the

way to configuring environment variables JAVA_HOME and SPARK_HOME. For organization, the application is dubbed "Global_Temperature."

```
#Find the Spark is installed or not
findspark.init()
findspark
```

```
<module 'findspark' from 'C:\\Users\\DELL\\anaconda3\\envs\\sproject\\lib\\site-packag
es\\findspark.py'>
```

```
from pyspark import SparkContext

sc = SparkContext.getOrCreate()
print(sc.version)
```

```
.5.3
```

```
#Need to install java and pyspark then set env in system and here
os.environ["JAVA_HOME"] = "C:/Program Files/Java/jdk-1.8"
os.environ["SPARK_HOME"] = "C:/Users/DELL/anaconda3/envs/sproject/Lib/site-packages/pys
```

```
#Start the session for acces the spark lib
spark = SparkSession.builder.appName('Global_Temprature').getOrCreate()
```

Figure 3: Code for Spark check

Loading the Dataset: The dataset is loaded into a Spark DataFrame with header recognition and schema inference enabled, so column types are handled automatically as seen in figure 4.

```
data1 = spark.read.csv('Datasets/GlobalLandTemperaturesByCity.csv', inferSchema = True, header = True)
```

```
data1.printSchema()
```

```
data1.show(10)
```

Figure 4: Code for Loading the dataset

Timestamp Conversion and Sorting: To convert the dt column which is initially a date strings into a timestamp for doing time-based operations. Lastly, using the dt column, the dataset is then sorted chronologically as in figure 5.

```
from pyspark.sql.functions import to_timestamp

data1 = data1.withColumn("dt", to_timestamp("dt")) # Convert 'dt' column to timestamp
data1.printSchema()
```

Figure 5: Code for Timestamp Conversion and Sorting

Monthly Aggregation: F.trunc truncates the dt column to the first day of each month. It is then grouped by the truncated month_start and the average temperature for a month is calculated to ensure consistency of the time series data granularity, as in Figure 6.

```
# Step 1: Sort the DataFrame by the 'dt' column
data1 = data1.orderBy('dt')

# Step 2: Create a new column that represents the first day of the month for each row (monthly frequency)
data1 = data1.withColumn('month_start', F.trunc('dt', 'MM'))

# Step 3: Group by the 'month_start' and calculate the average temperature for each month
data_monthly = data1.groupBy('month_start').agg(
    F.avg('AverageTemperature').alias('AverageTemperature')
)
```

Figure 6: Code for Monthly Aggregation

Handling Missing Values (Forward Filling): In Figure 7, we can see as another window specification that can do the missing value forward filling in the AverageTemperature column.

The rows that still have any remaining uncategorized null values after the forward-fill is removed as well.

```
window_spec = Window.orderBy('month_start')

data_monthly = data_monthly.withColumn(
    'AverageTemperature',
    F.last('AverageTemperature', True).over(window_spec)  # forward fill
)

# Show the result
data_monthly.show(20)
```

Figure 7: Code for Handling Missing Values (Forward Filling)

Stationarity Testing (ADF Test): A time series as seen in Figure 8 had its stationarity tested by running the Augmented Dickey-Fuller (ADF) test. As the p value is greater than 0.05, this meant the series could not be stationary.

```
from statsmodels.tsa.stattools import adfuller

# Step 1: Extract data into Pandas for the ADF test
# Assuming data_monthly is the PySpark DataFrame with 'month_start' and 'AverageTemperature' columns
data_pandas = data_monthly.select('AverageTemperature').toPandas()

# Step 2: Perform the ADF test
result = adfuller(data_panda s['AverageTemperature'])

# Step 3: Print the ADF test results
print('ADF Statistic:', result[0])
print('p-value:', result[1])
```

Figure 8: Code for ADF test

First-Order Differencing for Stationarity: The time series is made stationary by applying first order differencing. We then re-run Augmented Dickey-Fuller (ADF) test is and confirm stationarity with a p value lower than 0.05 as shown in Figure 9.

```
from pyspark.sql.functions import col, lag, when

# Step 1: Apply first-order differencing
# Create a window specification to calculate the lagged values
window_spec = Window.orderBy("month_start")

# Calculate the difference between the current and previous row
data_diff = data_monthly.withColumn(
    "DifferencedTemperature",
    col("AverageTemperature") - lag("AverageTemperature", 1).over(window_spec)
).dropna()

# Step 2: Perform the ADF test on the differenced data
# Convert the PySpark DataFrame to Pandas for the ADF test
data_diff_pandas = data_diff.select("DifferencedTemperature").toPandas()
result_diff = adfuller(data_diff_pandas["DifferencedTemperature"])

# Step 3: Print the test results
print('ADF Statistic (differenced):', result_diff[0])
print('p-value (differenced):', result_diff[1])
```

Figure 9: Code for First-Order Differencing for Stationarity

Seasonal Decomposition: Additive decomposition decomposes the time series into trend, seasonal and residual component. As shown in Figure 10, each component is then visualized to determine their corresponding time series patterns.

```python
from statsmodels.tsa.seasonal import seasonal_decompose

# Convert the PySpark DataFrame to Pandas DataFrame
data_pandas = data_monthly.select('month_start', 'AverageTemperature').toPandas()

# Use 'month_start' as the date and create a datetime index
data_pandas['Date'] = pd.to_datetime(data_pandas['month_start'])
data_pandas.set_index('Date', inplace=True)

# Perform seasonal decomposition on 'AverageTemperature'
decomposition = seasonal_decompose(data_pandas['AverageTemperature'], model='additive', period=12)

# Extract components
trend = decomposition.trend
seasonal = decomposition.seasonal
residual = decomposition.resid
```

Figure 10: Code for Seasonal Decomposition

Saving Preprocessed Data: The cleaned and aggregated Spark DataFrame was saved to a CSV for further modelling as shown in figure 11.

```python
# Save the PySpark DataFrame to a CSV file
output_path = "D:/ML_work/SnehaV1/Datasets/data_monthly.csv"

data_monthly.write.csv(output_path, header=True, mode="overwrite")

# Note: The output will be saved in multiple part files (one per Spark partition).
```

Figure 11: Code for Seasonal Decomposition

These plots showed trends, issues of stationarity, and seasonal patterns, all of which are needed in preparing data for modelling.

Original Time Series Plot: I visualized the Global average temperature over time.

Differenced Series Plot: This is to in order to check the effect of first order differencing in achieving stationarity.

Seasonal Decomposition Plots: The purpose of seasonal decomposition plots is to break down time series into trend, seasonal, and residual components in order to better understand the underlying pattern.

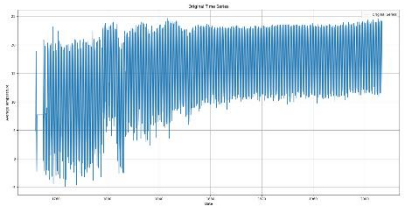

Figure 12: Plot for Original Time Series Plot

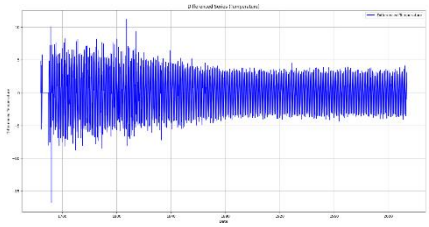

Figure 13: Plot for Differenced Temperature Series Plot

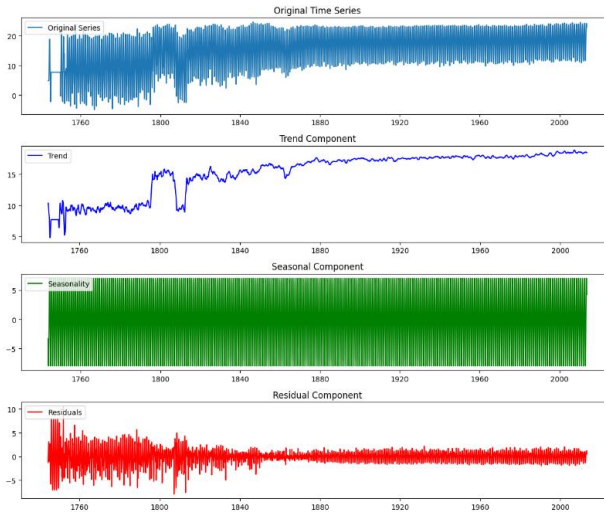

Figure 14: Plot Seasonal Decomposition

5.2 Power BI Visualizations

Data visualization with temperature data is shown on Power BI dashboards and charts. These visualizations were required to look at trends and check model outputs, and to understand regional climatic behaviors.

The European Countries and Cities Dashboard (Figure 15) offers a macroscopic view of temperature trends across 15 countries and 315 cities in Europe. It includes a heat map for geographic temperature fluctuations, a scatter plot for average temperatures versus uncertainties, and a line graph highlighting seasonal and long-term trends.

The Ireland Dashboard (Figure 16) focuses on Cork and Dublin, providing detailed metrics on temperature uncertainty and changes over time through scatter plots and line graphs, enabling localized analysis.

The Cork vs. Dublin Line Graph (Figure 17) compares 2013–2015 trends, showing Cork's higher, steadier temperatures and seasonal variations in both cities, illustrating regional climatic behaviors.

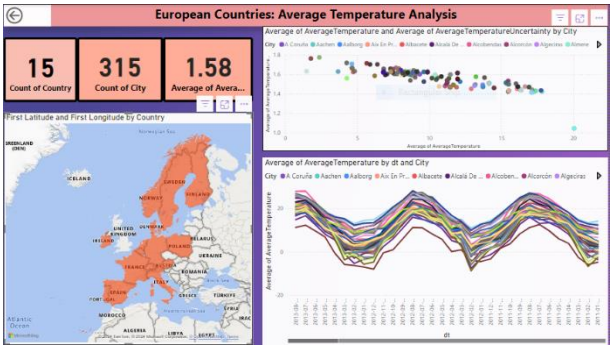

Figure 15: Dashboard on European Countries and Cities

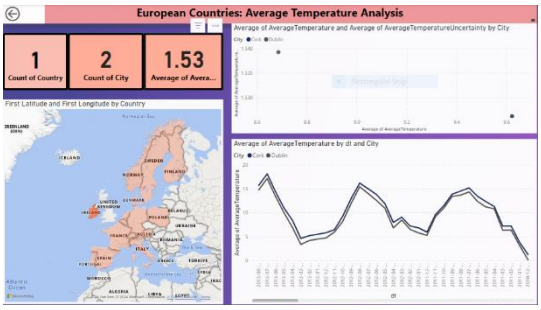

Figure 16: Dashboard on Ireland (Cork and Dublin)

Figure 17: Line Graph Comparing Cork and Dublin (2013–2015)

5.3 Model Implementation

To implement the model to perform time series forecasting of global temperatures. It exploits seasonality and trends by using Auto ARIMA and SARIMA models. For forecasting we additionally use regression-based models like XGBoost and linear regression.

Imports and Setup: Various libraries, including matplotlib, seaborn, pandas, numpy, and pmdarima, are used for data manipulation, visualization, and model training. PySpark is set up for distributed data processing, using SparkSession and SparkContext for efficient handling of large datasets as shown in Figure 18.

Data Loading: The temperature data is loaded into a Spark DataFrame from a CSV file containing monthly data (month_start, AverageTemperature). This data is used for training and forecasting as shown in Figure 19.

```
import os
import matplotlib.pyplot as plt

import findspark
from pyspark.sql import SparkSession

import seaborn as sns
import pandas as pd
import numpy as np

# Warnings remove alerts
import warnings
warnings.filterwarnings("ignore")
warnings.filterwarnings("ignore", category=UserWarning)  # For ValueWarning and ot
warnings.filterwarnings("ignore", category=ConvergenceWarning)

from pyspark.sql import functions as F
from pyspark.sql.window import Window
```

```
from pyspark import SparkContext

sc = SparkContext.getOrCreate()
print(sc.version)
```

```
3.5.3
```

```
from pyspark import SparkContext
sc = SparkContext.getOrCreate()
```

```
#Starting the session
spark = SparkSession.builder.appName('Global_Temprature').getOrCreate()
```

Figure 18: Code for importing and setting-up

```
data_monthly = spark.read.csv('Datasets/data_monthly.csv', inferSchema = True, header = True)
```

```
data_monthly.printSchema()
```

```
root
 |-- month_start: date (nullable = true)
 |-- AverageTemperature: double (nullable = true)
```

```
data_monthly.show(10)
```

```
+-----------+------------------+
|month_start|AverageTemperature|
+-----------+------------------+
| 1743-11-01| 4.882423512747879|
| 1743-12-01| 4.882423512747879|
| 1744-01-01| 4.882423512747879|
| 1744-02-01| 4.882423512747879|
| 1744-03-01| 4.882423512747879|
| 1744-04-01| 9.737433427762047|
| 1744-05-01|12.685514164305951|
| 1744-06-01| 16.86829603399433|
| 1744-07-01|18.844524079320134|
| 1744-08-01|18.844524079320134|
+-----------+------------------+
only showing top 10 rows
```

Figure 19: Code for Loading data

ARIMA Model: The Auto ARIMA model automatically selects the best parameters (p, d, q) for time series forecasting. It is applied to the temperature data to predict future values without considering seasonality, and the model's summary and parameters are printed as shown in figure 15.

SARIMA Model: A Seasonal ARIMA (SARIMA) model is trained to account for seasonal patterns (12-month seasonality). The seasonal order of the model is extracted and printed as shown in figure 16.

AutoARIMA Model

```
%%time

import pmdarima as pm
from pyspark.sql import SparkSession


data_monthly_pd = data_monthly.toPandas()

# target column
time_series_data = data_monthly_pd['AverageTemperature']

# Fit the auto ARIMA model
auto_arima_model = pm.auto_arima(time_series_data,
                                 seasonal=False,  # auto arima so Fal
                                 m=12,            # Seasonal period 12
                                 d=None,          # Let auto_arima dec
                                 stepwise=True,  # Stepwise search to
                                 suppress_warnings=True,  # Suppress
                                 trace=True)  # Enables detailed outp

# Print the summary of the model
print(auto_arima_model.summary())
```

Figure 20: Code for AutoARIMA Model

AutoSARIMA Model

```
%%time

import pmdarima as pm

time_series_data = data_monthly_pd['AverageTemperature']

# Fit the auto SARIMA model
auto_sarima_model = pm.auto_arima(time_series_data,
                                  seasonal=True,  # Enables SARIMA
                                  m=12,           # Seasonal period (12 for monthly
                                  d=None,         # Let auto_arima decide the differ
                                  D=None,         # Let auto_arima decide seasonal d
                                  start_p=1, start_q=1,  # Initial guess for AR and
                                  max_p=3, max_q=3,      # Limits for AR and MA term
                                  start_P=1, start_Q=1,  # Initial guess for seasona
                                  max_P=2, max_Q=2,      # Limits for seasonal AR an
                                  stepwise=True,         # Stepwise search to reduce
                                  suppress_warnings=True,  # Suppress warnings
                                  trace=True)  # Enable detailed output of the model

# Print the summary of the model
print(auto_sarima_model.summary())
```

Figure 21: Code for AutoSARIMA Model

AutoARIMA Model Implementation: An ARIMA model is trained using statsmodels library by using the best parameters from Auto ARIMA model. Its performance is tested using metrics like MSE, RMSE, and R^2.

AutoSARIMA Model Implementation: SARIMAX function in statsmodels trains the SARIMA model with the best seasonal parameters. MSE, RMSE, and R^2 metrics are used for performance evaluation.

Model-1 Arima

```
# Extract the best ARIMA parameters (p, d, q)
best_order = auto_arima_model.order
print("ARIMA (p, d, q):", best_order)
```

```
ARIMA (p, d, q): (2, 1, 4)
```

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

# Convert the Spark DataFrame to a Pandas DataFrame (only the 'AverageTemp
data_pandas = data_monthly.select("AverageTemperature").toPandas()

# Fit the ARIMA model with the best (p, d, q) parameters
arima_model = ARIMA(data_pandas['AverageTemperature'], order=best_order)
arima_result = arima_model.fit()

# Print ARIMA model summary
print(arima_result.summary())
```

Figure 22: Code for ARIMA Model

Model-2 SARIMA

```
# Extract the seasonal order from the trained auto ARIMA model
best_seasonal_order = auto_sarima_model.seasonal_order

# Print the SARIMA seasonal order (P, D, Q, m)
print("SARIMA seasonal order (P, D, Q, m):", best_seasonal_order)
```

```
SARIMA seasonal order (P, D, Q, m): (2, 0, 2, 12)
```

```
from statsmodels.tsa.statespace.sarimax import SARIMAX

# Fit the SARIMA model with the best (p, d, q) and seasonal parameters
sarima_model = SARIMAX(data_pandas['AverageTemperature'],  # 'AverageTempera
                       order=best_order,  # (p, d, q)
                       seasonal_order=best_seasonal_order)  # (P, D, Q, m)

# Fit the model
sarima_result = sarima_model.fit()

# Print the summary of the SARIMA model
print(sarima_result.summary())
```

Figure 23: Code for SARIMA Model

Data Splitting and Feature Engineering for XGBoost: Data Splitting and XGBoost Feature Engineering: The dataset is divided into two parts: an 80% training set and a 20% testing set. The XGBoost model is trained using the month and year attributes that are taken from the month_start column. The model's R2 and RMSE performance are employed.

Forecasting with XGBoost: Future temperatures are predicted using the trained XGBoost model. I plot these predictions along with the historical data.

```
#converting to Pandas DataFrame
data_pandas = data_monthly.select("month_start", "AverageTemperature").toPandas()

data_pandas['month_start'] = pd.to_datetime(data_pandas['month_start'])
data_pandas.set_index('month_start', inplace=True)

# assuming that 'AverageTemperature' is the target
# Adding 'month' and 'year' as features
data_pandas['month'] = data_pandas.index.month
data_pandas['year'] = data_pandas.index.year

#Prepare X (features) and y (target)
X = data_pandas[['month', 'year']]  # Adding month and year as features
y = data_pandas['AverageTemperature']

#Split the data into training and testing set(80% train, 20% test)
train_size = int(0.8 * len(data_pandas))
X_train, X_test = X[:train_size], X[train_size:]
y_train, y_test = y[:train_size], y[train_size:]
```

Figure 24: Code for Splitting

Model-3 XGBoost

```
%%time
import xgboost as xgb
import numpy as np
import pandas as pd
from sklearn.metrics import mean_squared_error, r2_score

# Step 6: Convert the data to DMatrix format (required by XGBoost)
dtrain = xgb.DMatrix(X_train, label=y_train)
dtest = xgb.DMatrix(X_test, label=y_test)

# Step 7: Set up hyperparameters for XGBoost (you can tune these as needed)
params = {
    'objective': 'reg:squarederror',  # For regression
    'eval_metric': 'rmse',            # Evaluation metric (Root Mean Squared Error)
    'max_depth': 6,                   # Maximum depth of the tree
    'learning_rate': 0.1,             # Learning rate
    'n_estimators': 100               # Number of boosting rounds
}

# Step 8: Train the XGBoost model
xgb_model = xgb.train(params, dtrain, num_boost_round=100)

# Step 9: Make predictions on the test data
y_pred = xgb_model.predict(dtest)

# Step 10: Evaluate the model (using RMSE and R²)
rmse = np.sqrt(mean_squared_error(y_test, y_pred))
r2 = r2_score(y_test, y_pred)

print(f"Mean Squared Error (MSE): {mean_squared_error(y_test, y_pred)}")
print(f"Root Mean Squared Error (RMSE): {rmse}")
print(f"R-squared (R²): {r2}")
```

Figure 25: Code for XGBoost

Linear Regression Model: Feature vectors for the Linear Regression model using a VectorAssembler. The model is trained by transforming the month_start column into a numerical format (timestamp). The predictions are then evaluated with standard metrics.

Evaluation and Plotting: Evaluate each of the models performance, i.e. ARIMA, SARIMA, XGBoost, and Linear Regression, using MSE, RMSE, and R^2 metrics. We visualize the forecasted values against historical data to see how accurate the predictions are.

Key Points: In order to resolve inconsistencies and non-stationarity, we preprocessed our dataset. The previous historical patterns were used as input to ARIMA and SARIMA models, while engineered features were fed to Linear Regression and XGBoost models for better predictions. PySpark was used for data processing and model training time, while Power BI visualizations was used for validating the trends.

Model-4 LinearRegression

```
from pyspark.ml.regression import LinearRegression
from pyspark.sql.functions import unix_timestamp
from sklearn.metrics import mean_squared_error

# Step 1: Convert the 'month_start' column to a numerical format
data_monthly = data_monthly.withColumn("month_start_numeric", unix_timestamp("month_start"))

# Step 2: Prepare the feature vector using the numeric column
vector_assembler = VectorAssembler(
    inputCols=["month_start_numeric"],  # Use the numeric representation of the date
    outputCol="features"
)

# Transform data to include the features column
data_transformed = vector_assembler.transform(data_monthly)

# Step 3: Split the data into training and testing sets
train_data, test_data = data_transformed.randomSplit([0.8, 0.2], seed=42)

# Step 4: Initialize and train the Linear Regression model
lr = LinearRegression(featuresCol="features", labelCol="AverageTemperature")  # Use AverageTemperature as label
lr_model = lr.fit(train_data)

# Step 5: Make predictions on the test data
predictions = lr_model.transform(test_data)

# Step 6: Convert predictions and actual values to pandas for easier MSE calculation
predictions_pd = predictions.select("prediction", "AverageTemperature").toPandas()

# Calculate MSE and RMSE
mse = mean_squared_error(predictions_pd["AverageTemperature"], predictions_pd["prediction"])
rmse = mse**0.5  # or you can use sklearn's `mean_squared_error` to directly compute RMSE

# Step 7: Evaluate the model using R²
evaluator_r2 = RegressionEvaluator(
    labelCol="AverageTemperature", predictionCol="prediction", metricName="r2"
)
r2 = evaluator_r2.evaluate(predictions)

# Step 8: Display evaluation metrics
print(f"Mean Squared Error (MSE): {mse}")
print(f"Root Mean Squared Error (RMSE): {rmse}")
print(f"R-squared (R²): {r2}")
```

Figure 26: Code for Linear Regression Model

6 Evaluation

The findings of the ARIMA, SARIMA, XGBoost, and Linear Regression models are thoroughly examined in this section. The performance of each model is evaluated using three key metrics: R-squared (R^2), Mean Squared Error (MSE), and Root Mean Squared Error (RMSE). These measures serve as a guide for interpreting each model's usefulness in relation to the temperature data forecasts and the research questions that the study aims to answer.

6.1 ARIMA Model Evaluation

The ARIMA model performed reasonably well in forecasting future temperatures, with the following results:

Table 1: Result of ARIM

Metric	Value
MSE	2.1626
RMSE	1.4706
R^2	0.8879

The ARIMA model does a great job with an R squared value of 0.8879 which is about 88.79% how the data varies. Although, its MSE is higher and its RMSE is higher than that of SARIMA and XGBoost, there is space for improving, especially, capturing seasonal patterns, which is important for the temperature data.

Plotting Results: The actual temperature data is accurately tracked by the SARIMA model and it predict both trends and seasonal variations. As can be seen from the plot (Figure 19), the data is modeled quite effectively by the model with minimal discrepancy from the actual data.

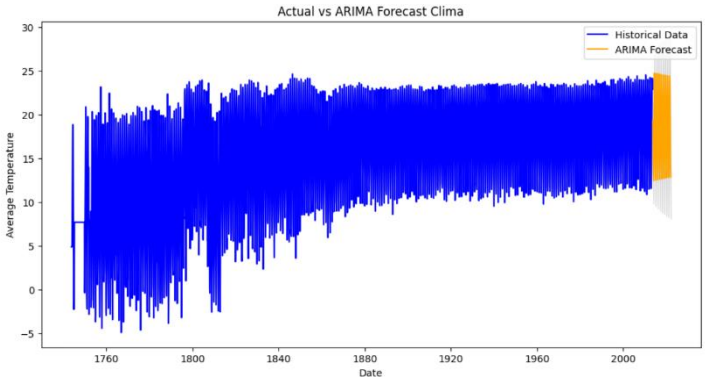

Figure 27: Predicted result of ARIM

Visually, historical data is very volatile, while the ARIMA forecast smoothes these variations; a more stable trend over time is present. The forecast's confidence interval is the yellow shaded area.

6.2 SARIMA Model Evaluation

The SARIMA model outperformed the ARIMA model very significantly. The results are as follows:

Table 2: Result of SARIMA

Metric	Value
MSE	0.3311
RMSE	0.5754
R^2	0.9828

Whilst ARIMA performed marginally better than various random learners, the SARIMA model outperformed, with lower MSE (0.3311) and RMSE (0.5754). It's R^2 of 0.9828 indicates that the temperature data are explained by 98.28% of the variation, demonstrating strong predictive power. ARIMA is not always better than SARIMA since at least when there are seasonality, SARIMA is more accurate than ARIMA due to its capability to capture the seasonality.

Plotting Results: Their SARIMA model is found to effectively follow the trend as well as the seasonal changes of actual temperature data. This demonstrates that the model is effective in modeling the data (plot shown in Figure 20), minimizing variance from actual values. The SARIMA forecast follows the general trend but with smoother curves and with less fluctuation and is above 10°C. And the graph shows that as you can see, temperature trends are predicted by SARIMA with less variability than the actual data.

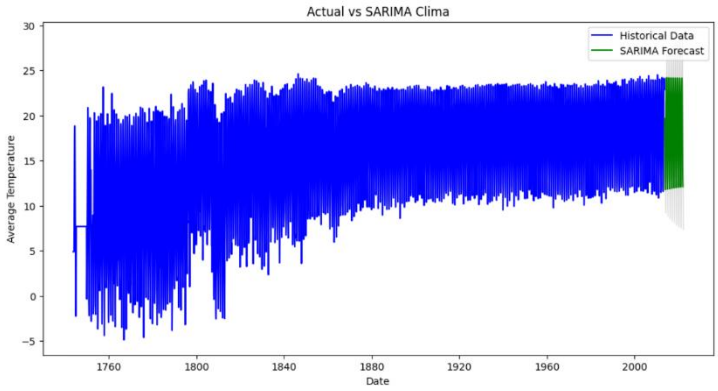

Figure 28: Predicted result of SARIMA

6.3 XGBoost Model Evaluation

XGBoost performed the best among the models, with the lowest error metrics:

Table 3: Result of XGBOOST

Metric	Value
MSE	0.2269
RMSE	0.4764
R^2	0.9882

XGBoost outperforms ARIMA and SARIMA with lowest MSE (0.2269) and RMSE (0.4764). As for R^2 value it has 0.9882 which shows that it explains 98.82% of the variance, just beating SARIMA. This is because XGBoost can model complex relationships well using gradient boosting, which is one of the reasons XGBoost works so well with temperature forecasting data which sometimes has seasonal as well as nonlinear patterns.

Plotting Results: Predicted values from XGBoost model rose very close to the actual data in the forecasts. The plot below shows the ability of the model to capture both trend and seasonality more than adequately. The model tracks temperature trends closely (green line) following the blue line, with slight deviations towards the end with the ranges of 5 to 25.

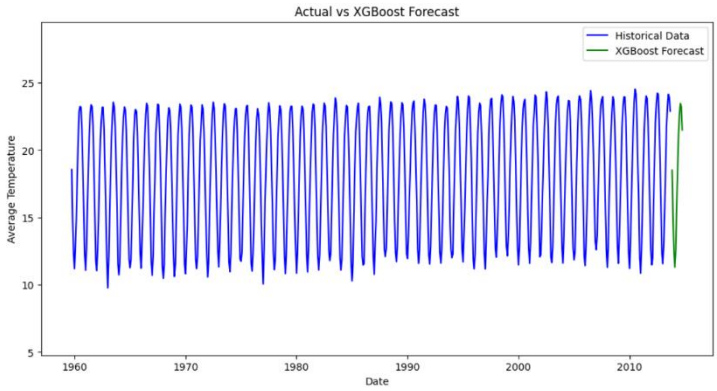

Figure 29: Predicted result of XGBOOST

6.4 Linear Regression Model Evaluation

Linear Regression showed the poorest performance among the models:

Table 4: Result of Linear Regression

Metric	Value
MSE	32.1586
RMSE	5.6709
R^2	0.1689

Further, the Linear Regression model did the worst, with large MSE, RMSE and low R^2 of 0.1689. This linear model fails to catch the seasonal trends and nonlinear trends observed in temperature data and therefore cannot be used for this time series analysis.

6.5 Analysis of Model Performance

ARIMA: The ARIMA model's strong performance allowed it to capture the general trend., but did not fully achieve the requirement of seasonality. However, it explained a large portion of the variance with an R^2 value of 0.8879, but the error metrics (MSE and RMSE) were higher than those of SARIMA and XGBoost.

SARIMA: Interestingly, SARIMA considering the seasonal components worked much better than ARIMA – lower MSE and RMSE, as well as higher R^2. We can see in the plot how

predicted and actual data lines up very close together and can certainly see seasonality in the temperature as shown in the lines.

XGBoost: XGBoost demonstrated the best overall performance with the lowest MSE, RMSE, and highest R^2. The model's capacity to capture intricate patterns in the data, like trends and seasonality, makes it the best model for this time series forecasting assignment.

Linear Regression: The least effective model for this data was the Linear Regression with a very high MSE and RMSE, and very low R^2. Since this did not capture the trend or seasonality of the temperature data, this model is not fit to be used for this type of time series forecasting.

6.6 Comparison of Models

The models were evaluated using the following metrics: RMSE, MSE, and R^2. The table below presents the comparison of these models based on these metrics:

Table 5: Result of Linear Regression

Model	RMSE	MSE	R^2	Remarks
ARIMA	2.1626	1.4706	0.8879	Good fit with relatively low errors, captures linear trends effectively.
SARIMA	0.3311	0.5754	0.9828	Best for seasonal data; performed slightly better than ARIMA.
XGBoost	0.2269	0.4764	0.9882	Best overall performance, handles non-linearity well, but computationally intensive.
Linear Regression	32.1586	5.6709	0.1689	Performs well for simpler trends, but does not capture seasonal or non-linear patterns.

But because temperature data has seasonality SARIMA fared much better than ARIMA, and XGBoost, owing to its ability to analyze nonlinear patterns, achieved the best overall performance with a lower RMSE, MSE, and higher R^2. On the other hand, Linear Regression performed the worst as it contained the largest RMSE and MSE, this is the dataset with both trends and seasonality. Trend of correct temperature predictions are compared, thus highlighting the importance of using the right model for the right data to make correct predictions.

6.7 Discussion

This study demonstrated how well sophisticated predictive models—particularly XGBoost perform in weather forecasting. With the lowest error metrics and the greatest R2, XGBoost outperformed more conventional models like ARIMA, SARIMA, and Linear Regression in handling vast, complicated, and nonlinear data. However, SARIMA performed better at identifying seasonal patterns, which is consistent with research by Liu et al. (2019) and Feigl et al. (2021). Regional dataset validation demonstrated strong precision but revealed

limitations in scalability to diverse climatic zones, echoing Meehl et al.'s (2021) emphasis on multi-regional datasets.

Hyperparameter tuning improved performance, consistent with Abrahamsen et al. (2018). Future enhancements could include additional meteorological variables, such as humidity and wind speed, or genetic algorithm optimization for SARIMA, as suggested by Farsi et al. (2021). Hybrid methodologies, like SARIMA–XGBoost, and broader datasets incorporating features like jet stream patterns and vegetation indices, could further improve accuracy, supporting machine learning's transformative potential in weather prediction.

7 Conclusion and Future Work

This research aimed to forecast global temperature trends using time series analysis and predictive modeling, evaluating ARIMA, SARIMA, XGBoost, and Linear Regression models with historical data. Comprehensive preprocessing, feature engineering, and model evaluation identified SARIMA as effective for seasonality, while XGBoost better captured nonlinear patterns. Linear Regression proved less suitable for such data. Findings emphasize selecting models based on data characteristics, especially for seasonal trends like global temperatures.

Future work could explore advanced machine learning methods, such as LSTM, for improved predictive accuracy, leveraging their ability to handle sequential data. External factors like greenhouse gas emissions, ocean currents, and solar radiation were integrated, enhancing long-term climate prediction. Applying these models to localized climate data may reveal regional influences. The results have practical applications for industries reliant on temperature forecasts, such as agriculture, energy, and insurance, supporting decision-making in climate-sensitive areas.

References

Abrahamsen, E.B., Brastein, O.M. and Lie, B., 2018. Machine learning in python for weather forecast based on freely available weather data.

Bahari, M. and Hamid, N.Z.A., 2019, June. Analysis and prediction of temperature time series using chaotic approach. In IOP Conference Series: Earth and Environmental Science (Vol. 286, No. 1, p. 012027). IOP Publ

Bochenek, B. and Ustrnul, Z., 2022. Machine learning in weather prediction and climate analyses—applications and perspectives. *Atmosphere*, *13*(2), p.180.

Bochenek, B. and Ustrnul, Z., 2022. Machine learning in weather prediction and climate analyses—applications and perspectives. *Atmosphere*, *13*(2), p.180.

Farsi, M., Hosahalli, D., Manjunatha, B.R., Gad, I., Atlam, E.S., Ahmed, A., Elmarhomy, G., Elmarhoumy, M. and Ghoneim, O.A., 2021. Parallel genetic algorithms for optimizing the SARIMA model for better forecasting of the NCDC weather data. *Alexandria Engineering Journal*, *60*(1), pp.1299-1316.

Feigl, M., Lebiedzinski, K., Herrnegger, M. and Schulz, K., 2021. Machine-learning methods for stream water temperature prediction. *Hydrology and Earth System Sciences*, *25*(5), pp.2951-2977.

Feng, Y., Gong, D., Zhang, Q., Jiang, S., Zhao, L. and Cui, N., 2019. Evaluation of temperature-based machine learning and empirical models for predicting daily global solar radiation. *Energy conversion and management*, *198*, p.111780.

He, Z., Shi, T., Xuan, J. and Li, T., 2021. Research on tool wear prediction based on temperature signals and deep learning. *Wear*, *478*, p.203902.

Hewage, P., Trovati, M., Pereira, E. and Behera, A., 2021. Deep learning-based effective fine-grained weather forecasting model. *Pattern Analysis and Applications*, *24*(1), pp.343-366.

Jaseena, K.U. and Kovoor, B.C., 2022. Deterministic weather forecasting models based on intelligent predictors: A survey. *Journal of king saud university-computer and information sciences*, *34*(6), pp.3393-3412.

Kumari, S. and Singh, S.K., 2023. Machine learning-based time series models for effective CO2 emission prediction in India. *Environmental Science and Pollution Research*, *30*(55), pp.116601-116616.

Li, X., Zhu, Y., Li, Q., Zhao, H., Zhu, J. and Zhang, C., 2023. Interpretable spatio-temporal modeling for soil temperature prediction. *Frontiers in Forests and Global Change*, *6*, p.1295731.

Liu, Q., Zou, Y., Liu, X. and Linge, N., 2019. A survey on rainfall forecasting using artificial neural network. *International Journal of Embedded Systems*, *11*(2), pp.240-249.

Meehl, G.A., Richter, J.H., Teng, H., Capotondi, A., Cobb, K., Doblas-Reyes, F., Donat, M.G., England, M.H., Fyfe, J.C., Han, W. and Kim, H., 2021. Initialized Earth System prediction from subseasonal to decadal timescales. *Nature Reviews Earth & Environment*, *2*(5), pp.340-357.

Meenal, R., Michael, P.A., Pamela, D. and Rajasekaran, E., 2021. Weather prediction using random forest machine learning model. *Indonesian Journal of Electrical Engineering and Computer Science*, *22*(2), pp.1208-1215.

Purwandari, K., Sigalingging, J.W., Cenggoro, T.W. and Pardamean, B., 2021. Multi-class weather forecasting from twitter using machine learning aprroaches. *Procedia Computer Science*, *179*, pp.47-54.

Ren, X., Li, X., Ren, K., Song, J., Xu, Z., Deng, K. and Wang, X., 2021. Deep learning-based weather prediction: a survey. *Big Data Research*, *23*, p.100178.

Schultz, M.G., Betancourt, C., Gong, B., Kleinert, F., Langguth, M., Leufen, L.H., Mozaffari, A. and Stadtler, S., 2021. Can deep learning beat numerical weather prediction?, Philos. In *Roy. Soc. A* (Vol. 379, No. 20200097, pp. 10-1098).

Shilong, Z., 2021, January. Machine learning model for sales forecasting by using XGBoost. In *2021 IEEE international conference on consumer electronics and computer engineering (ICCECE)* (pp. 480-483). IEEE.

Shrivastava, V.K., Shrivastava, A., Sharma, N., Mohanty, S.N. and Pattanaik, C.R., 2023. Deep learning model for temperature prediction: an empirical study. *Modeling Earth Systems and Environment*, *9*(2), pp.2067-2080.

Tabari, H., Hosseinzadeh Talaee, P. and Willems, P., 2015. Short-term forecasting of soil temperature using artificial neural network. *Meteorological Applications*, *22*(3), pp.576-585.