ALLIANCE UNIVERSITY

Private University Estd. in Karnataka State by Act No . 34 of year 2010
Recognized by the University Grants Commission (UGC), New Delhi

School of Advanced Computing

Department of Computer Science and Engineering

B. Tech. CSE, V Sem. DA-A

5CS 1020 Machine Learning

Mini Project Report

Project Title:
Weather Prediction (Temperature)

Group Members:

1. Lagisetty Bhupala Vignesh (2022BCSE07AED283)
2. Saya Preetham (2022BCSE07AED029)
3. Saya Rishi (2022BCSE07AED028)
4. Araveti Vishnu (2022BCSE07AED127)

Date of Submission: 28/10/2024

Table of Contents

Description **Page no.**

---

1. Introduction

- **Objective of the Project:**

Weather prediction is used to predict the current weather situation. The application of physics principles, augmented by a range of statistical and analytical methods, to predict the weather is known as weather forecasting.

Climate forecast, especially temperature determining, plays a pivotal part in different segments, counting horticulture, transportation, vitality, and open security. Exact temperature forecasts help people and organizations arrange and make educated choices. This venture centers on creating a machine learning show to anticipate temperature utilizing relapse strategies. By analyzing authentic climate information and distinguishing designs, the show points to give solid forecasts for future temperature values. Here we use Regression examination, a key concept in machine learning, is well-suited for temperature forecast as it permits us to assess ceaseless values. It sets up connections between autonomous factors (such as mugginess, wind speed, or weight) and the subordinate variable (temperature) based on chronicled information. This venture leverages relapse calculations to construct, assess, and optimize a demonstration for exact temperature estimating.

The goal of this project is to develop a machine learning model using regression techniques to predict accurate temperature based on historical weather datasets.

- **Scope of the Project:**

The primary goal of our project is to develop a user-friendly and accessible weather forecasting system, specifically focusing on temperature predictions. This system aims to provide valuable information to the general public, enabling them to make informed decisions and plan their activities accordingly. By leveraging a regression algorithm, we can effectively handle large volumes of data and deliver accurate temperature forecasts.

---

2. Problem Statement

People may not be aware of the current temperature and may not be able to protect themselves from extreme temperatures. However, by using this application, people can easily access real-time temperature information for their current location.

---

3. Workload Matrix

- Collection of datasets from various sources – S. Rishi
- Literature Review – S. Preetham
- Data Pre-Processing – A.Vishnu
- Model Development, Selecting suitable Algorithm, Predicting, conclusion - L. Bhupala Vignesh

4. Literature Review

1. Profound Learning for Climate Expectation and Climate Informatics:
Title: "Profound learning for precipitation and temperature estimating in climate alter scenarios"
Authors: Ribeiro, M., et al.
Summary: This paper investigates utilizing profound learning models, especially LSTMs and CNNs, for climate and weather-related forecasts beneath shifting climate alter scenarios. It emphasizes capturing non-linear connections in expansive temperature datasets and appears how profound models can be tuned for climate variability.
Published In: IEEE Get to, 2021.

2. Crossover Models for Temperature Prediction:
Title: "A half breed machine learning approach to temperature expectation: Joining ARIMA and LSTM models"
Authors: Wang, Z., et al.
Summary: This paper presents an ARIMA-LSTM cross breed demonstrate, combining straight and non-linear modelling approaches. The ponder illustrates how the ARIMA component

captures drift and regularity, whereas the LSTM demonstrate improves expectation exactness by learning complex designs in leftover data.
Published In: Natural Displaying & Computer program, 2022.

3. Attention-Based Neural Systems for Climate Forecasting:
Title: "Transient Consideration Instruments in LSTM and GRU Systems for Time Arrangement Estimating in Climate Prediction"
Authors: Zhou, F., et al.
Summary: This paper explores consideration components in time arrangement determining for climate, cantering on temperature and stickiness information. The creators highlight how attention-enhanced LSTM and GRU models progress long-term estimating by prioritizing pertinent past time steps.
Published In: Neurocomputing, 2023.

4. Exchange Learning for Temperature Expectation in Data-Scarce Regions:
Title: "Applying exchange learning for temperature expectation in data-scarce situations: A case thinks about in Sub-Saharan Africa"
Authors: Kim, H., and Johnson, M.
Summary: This ponder investigates the adequacy of exchange learning by utilizing pre-trained models adjusted to foresee temperature in locales with constrained information. It grandstands space adjustment methods and highlights changes in temperature forecast accuracy.
Published In: Diary of Connected Meteorology and Climatology, 2021.

5. Comparative Think about of ML Models in Temperature Forecasting:
Title: "Comparing machine learning approaches for brief and medium-term temperature forecasting"
Authors: García, L., et al.
Summary: This paper compares conventional models (ARIMA, SVM) and ML models (Arbitrary Woodland, XGBoost) in temperature estimating. The comes about emphasize outfit learning’s strength over classical approaches, particularly in medium-term predictions.
Published In: Connected Vitality, 2020.

---

5. Data Collection

- **Data Source:**
  Here we used GlobalLandTemperaturesByMajorCity dataset.
- **Data Description:**
  The GlobalLandTemperaturesByMajorCity dataset has historical temperature data of major cities around the world

  Overview:

  Source: This is data is taken from various meteorological sources, including National Oceanic and Atmospheric Administration (NOAA) etc.

Time Frame: This dataset has temperature records from the year 1743 to the last year.

Frequency: Data provides on a monthly or daily basis, but it presented average monthly temperature in this dataset.

Key Features:

1. City
2. Country
3. Latitude
4. Longitude
5. Data
6. AverageTemperature
7. AverageTemperatureUncertainty

It is used for Climate Analysis, Data Visualization, Comparative Studies, Forecasting etc.

---

6. Data Preprocessing

- **Handling Missing Values:**
  In the dataset if there are missing values in the dataset, we need to fill which are represented as NaN, Blank space etc. So, to fill the missing values their different ways to fill.

  1. By deleting the rows.
     df.dropna()
  2. By imputation methods like.
     - Mean, Median, Mode
       df.fillna(df.mean(), inplace=True)
       df.fillna(df.median(), inplace=True)
       df[column].fillna(df[column].mode()[0], inplace=True)
     - Interpolation Techniques
       df.interpolate(method='linear', inplace=True)
       df.interpolate(method='quadratic', inplace=True)

  We used mean and mode imputation method

```
df.isnull().sum()
```

	0
dt	0
AverageTemperature	11002
AverageTemperatureUncertainty	11002
City	0
Country	0
Latitude	0
Longitude	0

dtype: int64

```
df['AverageTemperature'].fillna(df['AverageTemperature'].mean(),inplace=True)
df['AverageTemperatureUncertainty'].fillna(df['AverageTemperatureUncertainty'].mode()[0],inplace=True)
df.isnull().sum()
```

```
<ipython-input-7-12694e983668>:1: FutureWarning: A value is trying to be set on a copy of a DataFrame or Series through chained assign
The behavior will change in pandas 3.0. This inplace method will never work because the intermediate object on which we are setting va

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method({col: value}, inplace=True)' or df[col] = df[col].


  df['AverageTemperature'].fillna(df['AverageTemperature'].mean(),inplace=True)
<ipython-input-7-12694e983668>:2: FutureWarning: A value is trying to be set on a copy of a DataFrame or Series through chained assign
The behavior will change in pandas 3.0. This inplace method will never work because the intermediate object on which we are setting va

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method({col: value}, inplace=True)' or df[col] = df[col].


  df['AverageTemperatureUncertainty'].fillna(df['AverageTemperatureUncertainty'].mode()[0],inplace=True) # Corrected the column name a
```

	0
dt	0
AverageTemperature	0
AverageTemperatureUncertainty	0
City	0
Country	0
Latitude	0
Longitude	0

dtype: int64

- **Feature Scaling/Normalization:**

```
from sklearn.preprocessing import MinMaxScaler

# Assuming your DataFrame is named 'df' and has columns 'Latitude' and 'Longitude'

# Function to convert latitude and longitude to numerical values
def convert_to_numeric(value):
    """Converts latitude/longitude string to numeric value.
    """
    direction = value[-1]  # Get the last character (N, S, E, W)
    numeric_part = float(value[:-1])  # Get the numeric part
    if direction in ('S', 'W'):
        numeric_part *= -1  # Apply negative sign for South and West
    return numeric_part

# Applying the conversion to Latitude and Longitude columns
df['Latitude'] = df['Latitude'].apply(convert_to_numeric)
df['Longitude'] = df['Longitude'].apply(convert_to_numeric)

# Now you can scale the data
scaler = MinMaxScaler()
df[['Latitude_scaled', 'Longitude_scaled']] = scaler.fit_transform(df[['Latitude', 'Longitude']])
```

```
[ ] from sklearn.preprocessing import MinMaxScaler

# Initializing the MinMaxScaler
scaler = MinMaxScaler()

# Applying Min-Max scaling to 'AverageTemperature' column
df['AverageTemperature_scaled'] = scaler.fit_transform(df[['AverageTemperature']])

# Scaling other features like Latitude and Longitude if needed
df[['Latitude_scaled', 'Longitude_scaled']] = scaler.fit_transform(df[['Latitude', 'Longitude']])

# Check results
print(df[['AverageTemperature', 'AverageTemperature_scaled']].head())
```

```
   AverageTemperature  AverageTemperature_scaled
0              26.704                   0.822012
1              27.434                   0.833233
2              28.101                   0.843486
3              26.140                   0.813343
4              25.427                   0.802383
```

- **Encoding Categorical Variables:**

```
from sklearn.preprocessing import LabelEncoder
label_encoder = LabelEncoder()
df['City_Label'] = label_encoder.fit_transform(df['City'])

# One-Hot Encoding
df_one_hot = pd.get_dummies(df, columns=['City'], prefix='City')
df_one_hot.head()
```

	dt	AverageTemperature	AverageTemperatureUncertainty	Country	Latitude	Longitude	Latitude_scaled	Longitude_scaled	AverageTemperature_scaled	City_Label	...	City_São Paulo	City_
0	1849-01-01	26.704	1.435	Côte D'Ivoire	5.63	-3.23	0.442733	0.426908	0.822012	0	...	False	
1	1849-02-01	27.434	1.362	Côte D'Ivoire	5.63	-3.23	0.442733	0.426908	0.833233	0	...	False	
2	1849-03-01	28.101	1.612	Côte D'Ivoire	5.63	-3.23	0.442733	0.426908	0.843486	0	...	False	
3	1849-04-01	26.140	1.387	Côte D'Ivoire	5.63	-3.23	0.442733	0.426908	0.813343	0	...	False	
4	1849-05-01	25.427	1.200	Côte D'Ivoire	5.63	-3.23	0.442733	0.426908	0.802383	0	...	False	

5 rows × 110 columns

- **Train-Test Split:**

```
# Assuming 'AverageTemperature' is the correct target variable name
X=df[['City']]
y=df['AverageTemperature']

# Perform the train-test split
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

print("Training set features:")
X_train
print("Testing set features:")
X_test
```

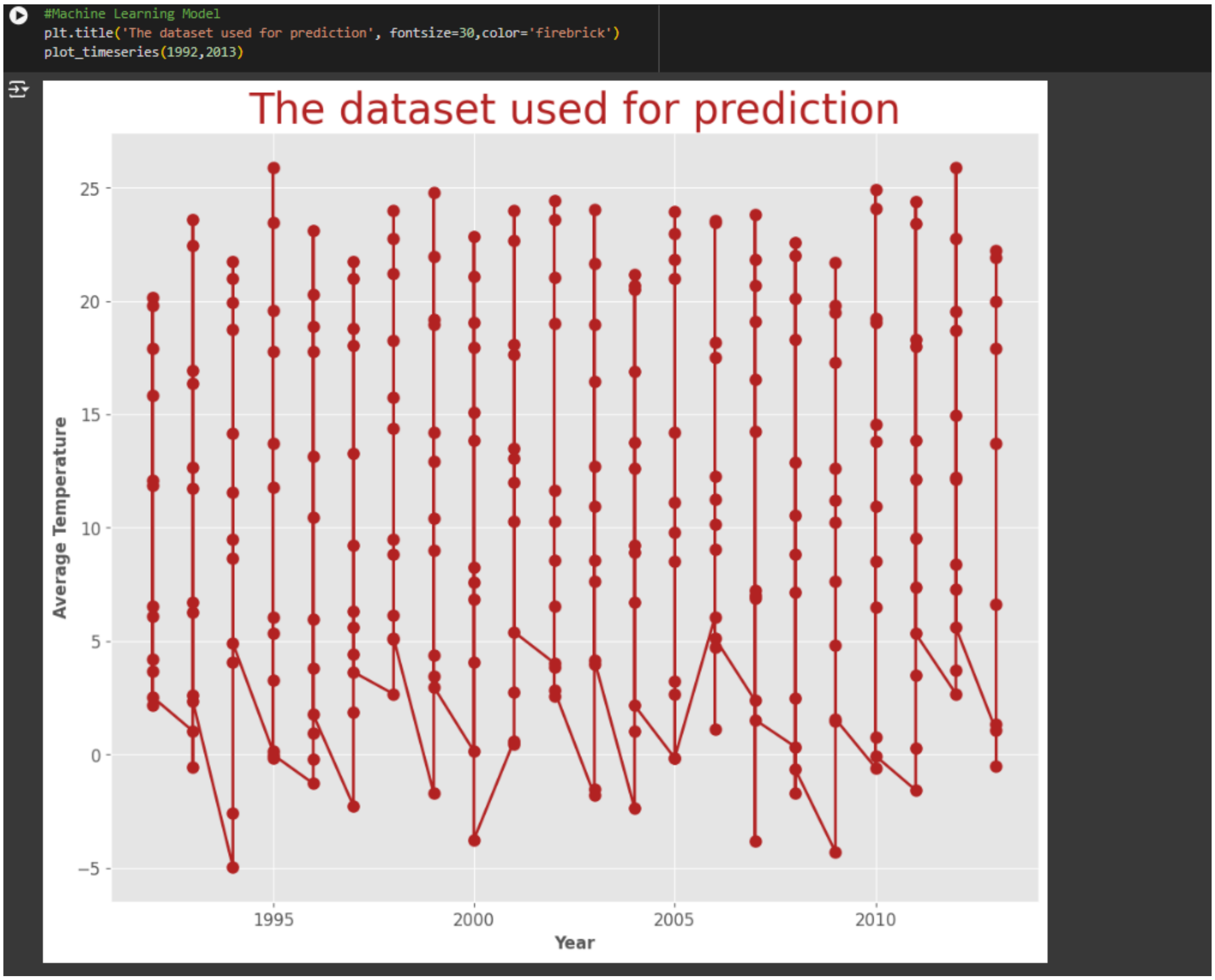
```
Training set features:
Testing set features:
```

	City
165402	Nanjing
18258	Bangalore
209961	Singapore
42224	Cape Town
216990	Sydney
...	...
62837	Dar Es Salaam
35781	Cairo
121597	Lagos
135540	Luanda
53039	Chicago

47836 rows × 1 columns

7. Model Development (2-3 models)

- **Model Selection:**
  Auto Regressive Integrated Moving Average models (ARIMA) is used to forecasting the average temperature.

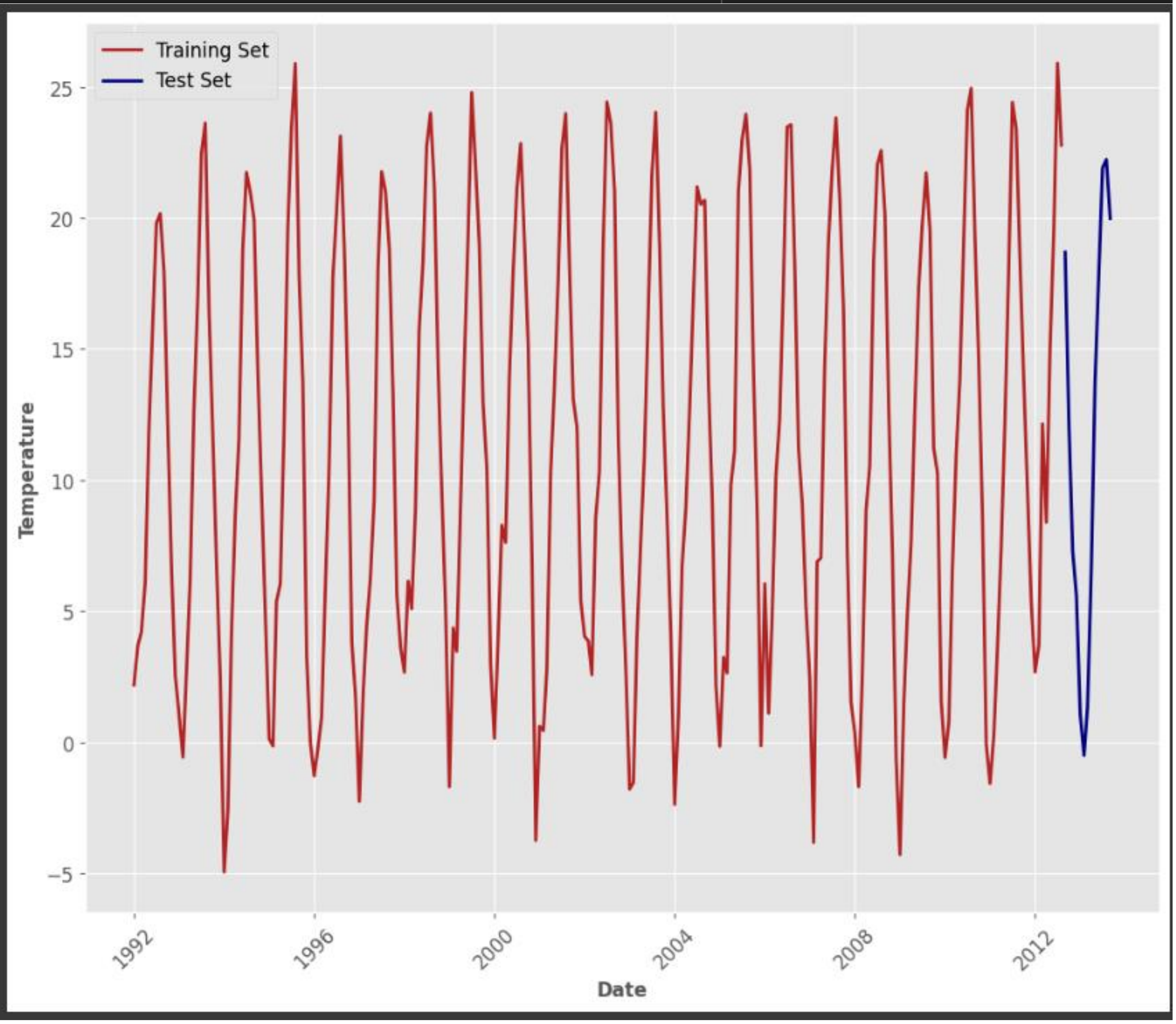


- **Training the Model:**

```
# Convert 'dt' column to datetime format
df['dt'] = pd.to_datetime(df['dt'])

# Extract the year from 'dt'
df['Year'] = df['dt'].dt.year

# Define the function to get time series data for a range of years
def get_timeseries(start_year, end_year, city='Chicago'):
    # Filter the data for the specified city and years
    city_data = df[(df['City'] == city) & (df['Year'] >= start_year) & (df['Year'] <= end_year)]

    # Drop rows with missing temperature data
    city_data = city_data.dropna(subset=['AverageTemperature'])

    return city_data

# Now, get the time series data for 1992 to 2013
temp = get_timeseries(1992, 2013)

# Perform train/test split
N = len(temp['AverageTemperature'])
split = 0.95  # 95% for training, 5% for testing
training_size = round(split * N)

# Splitting the data
train_data = temp.iloc[:training_size]
test_data = temp.iloc[training_size:]

# Print the sizes of train and test sets
print(f"Training data size: {len(train_data)}")
print(f"Test data size: {len(test_data)}")
```

```
Training data size: 248
Test data size: 13
```

```python
# Define the function to plot the data
def plot_from_data(series, date, color='firebrick', with_ticks=True, label=None):
    plt.plot(date, series, color=color, label=label)
    if with_ticks:
        plt.xticks(rotation=45)
    plt.xlabel('Date')
    plt.ylabel('Temperature')
    plt.grid(True)

series = train_data['AverageTemperature']
date = train_data['dt']
test_series = test_data['AverageTemperature']
test_date = test_data['dt']

# Plot the training data
plot_from_data(series, date, label='Training Set')

# Plot the test data
plot_from_data(test_series, test_date, color='navy', with_ticks=False, label='Test Set')

# Add legend and show the plot
plt.legend()
plt.show()
```



This machine learning algorithm are the ARIMA models based on a optimization procedure that adopts the maximum likelihood function.

```python
def optimize_ARIMA(order_list, exog):
    """
        Return dataframe with parameters and corresponding AIC

        order_list - list with (p, d, q) tuples
        exog - the exogenous variable
    """

    results = []

    for order in tqdm_notebook(order_list):
        #try:
        model = SARIMAX(exog, order=order).fit(disp=-1)
    #except:
    #        continue

        aic = model.aic
        results.append([order, model.aic])
    #print(results)
    result_df = pd.DataFrame(results)
    result_df.columns = ['(p, d, q)', 'AIC']
    #Sort in ascending order, lower AIC is better
    result_df = result_df.sort_values(by='AIC', ascending=True).reset_index(drop=True)

    return result_df
```

```python
ps = range(0, 10, 1)
d = 0
qs = range(0, 10, 1)

# Create a list with all possible combination of parameters
parameters = product(ps, qs)
parameters_list = list(parameters)

order_list = []

for each in parameters_list:
    each = list(each)
    each.insert(1, d)
    each = tuple(each)
    order_list.append(each)

result_d_0 = optimize_ARIMA(order_list, exog = series)
```

```
<ipython-input-21-4b27fe9084a9>:11: TqdmDeprecationWarning: This function will be removed in tqdm==5.0.0
Please use `tqdm.notebook.tqdm` instead of `tqdm.tqdm_notebook`
  for order in tqdm_notebook(order_list):
100%  100/100 [03:05<00:00,  3.23s/it]
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473: ValueWarning: An unsupported index was
  self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473: ValueWarning: An unsupported index was
  self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473: ValueWarning: An unsupported index was
  self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473: ValueWarning: An unsupported index was
  self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/statespace/sarimax.py:978: UserWarning: Non-invertible startin
  warn('Non-invertible starting MA parameters found.'
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473: ValueWarning: An unsupported index was
  self._init_dates(dates, freq)
```

```python
result_d_0.head()
```

	(p, d, q)	AIC
0	(4, 0, 6)	1097.333480
1	(4, 0, 5)	1098.088886
2	(5, 0, 6)	1098.094928
3	(3, 0, 5)	1098.118585
4	(7, 0, 8)	1098.172593

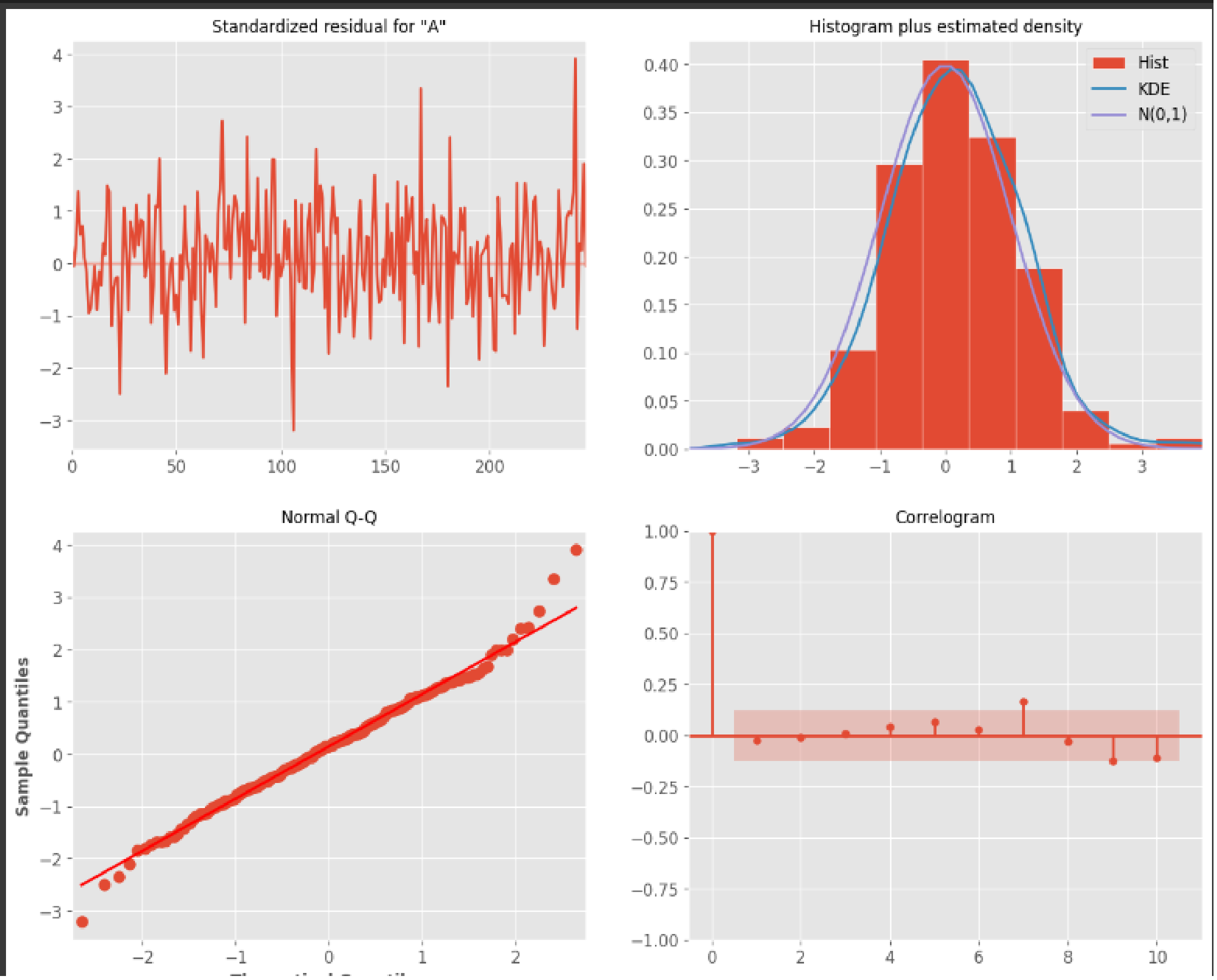
```python
#first-differentiated models are considered by using these lines:
ps = range(0, 10, 1)
d = 1
qs = range(0, 10, 1)

# Create a list with all possible combination of parameters
parameters = product(ps, qs)
parameters_list = list(parameters)

order_list = []

for each in parameters_list:
    each = list(each)
    each.insert(1, d)
    each = tuple(each)
    order_list.append(each)

result_d_1 = optimize_ARIMA(order_list, exog = series)

result_d_1
```

```
<ipython-input-21-4b27fe9084a9>:11: TqdmDeprecationWarning: This function will be removed in tqdm==5.0.0
Please use `tqdm.notebook.tqdm` instead of `tqdm.tqdm_notebook`
  for order in tqdm_notebook(order_list):
100% 100/100 [02:42<00:00,  2.81s/it]
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473: ValueWarning: An unsupported index was provided. As a
  self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473: ValueWarning: An unsupported index was provided. As a
  self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473: ValueWarning: An unsupported index was provided. As a
  self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473: ValueWarning: An unsupported index was provided. As a
  self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473: ValueWarning: An unsupported index was provided. As a
  self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473: ValueWarning: An unsupported index was provided. As a
  self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473: ValueWarning: An unsupported index was provided. As a
  self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473: ValueWarning: An unsupported index was provided. As a
  self._init_dates(dates, freq)
```

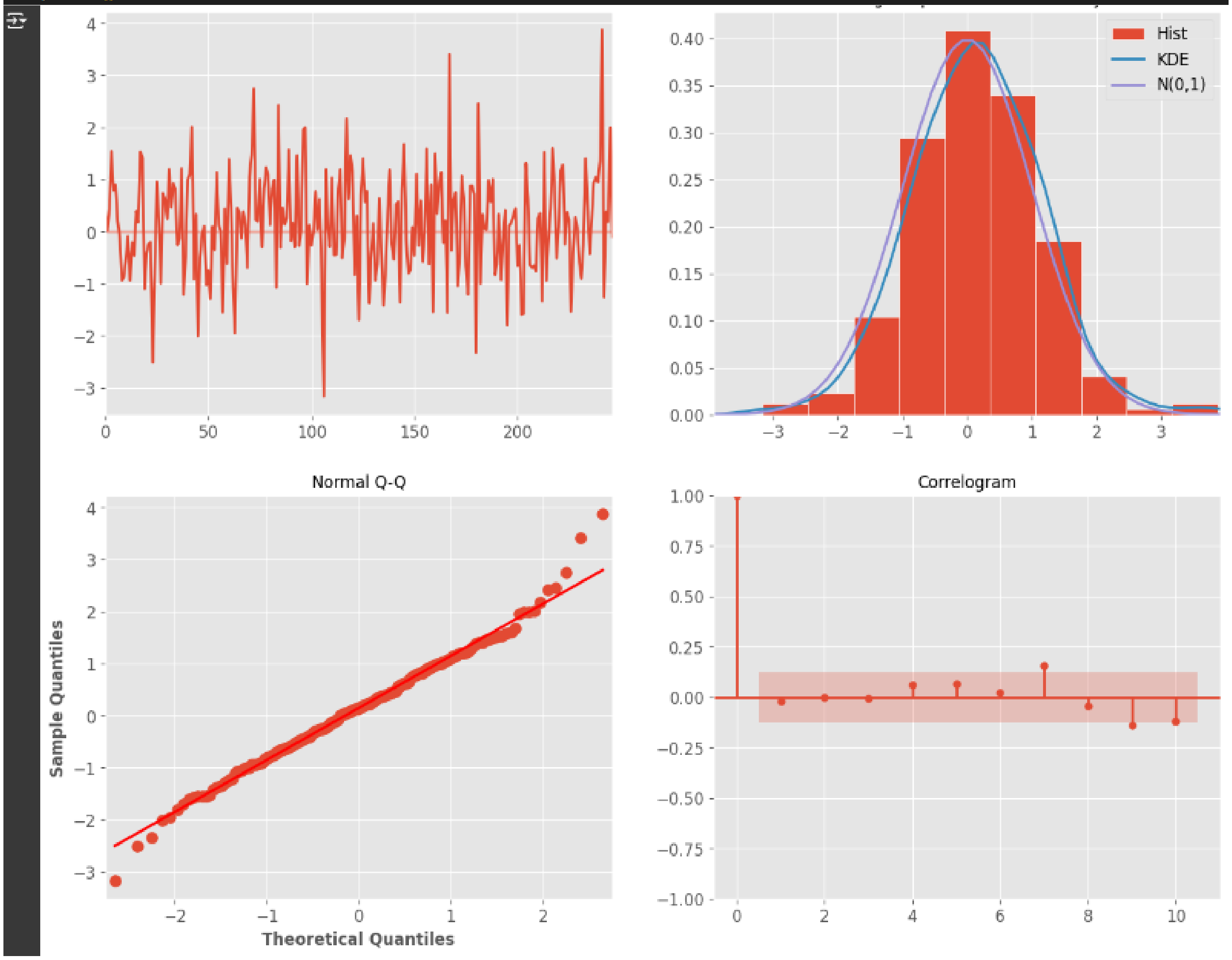
```
result_d_1.head()
```

	(p, d, q)	AIC
0	(2, 1, 5)	1085.631699
1	(2, 1, 6)	1089.793160
2	(3, 1, 5)	1090.048234
3	(8, 1, 9)	1090.558399
4	(9, 1, 9)	1091.198077

```
#Model (2, 1, 5)
best_model_0.plot_diagnostics(figsize=(15,12))
plt.show()
```



```
#Model (2, 1, 6)
best_model_1.plot_diagnostics(figsize=(15,12))
plt.show()
```



- **Tools and Libraries Used:**

```python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import folium
import imageio
from tqdm import tqdm_notebook
from folium.plugins import MarkerCluster
import geoplot as gplt
import geopandas as gpd
import geoplot.crs as gcrs
import imageio
import mapclassify as mc
import statsmodels.api as sm
from statsmodels.tsa.stattools import adfuller
from statsmodels.tsa.arima_model import ARIMA
import scipy
from itertools import product
import seaborn as sns
from statsmodels.graphics.tsaplots import plot_pacf
from statsmodels.graphics.tsaplots import plot_acf
from statsmodels.tsa.arima_process import ArmaProcess
from statsmodels.stats.diagnostic import acorr_ljungbox
from statsmodels.tsa.statespace.sarimax import SARIMAX
from statsmodels.tsa.stattools import adfuller
from statsmodels.tsa.stattools import pacf
from statsmodels.tsa.stattools import acf

plt.style.use('ggplot')
plt.rcParams['font.family'] = 'sans-serif'
plt.rcParams['font.serif'] = 'Ubuntu'
plt.rcParams['font.monospace'] = 'Ubuntu Mono'
plt.rcParams['font.size'] = 14
plt.rcParams['axes.labelsize'] = 12
plt.rcParams['axes.labelweight'] = 'bold'
plt.rcParams['axes.titlesize'] = 12
plt.rcParams['xtick.labelsize'] = 12
plt.rcParams['ytick.labelsize'] = 12
plt.rcParams['legend.fontsize'] = 12
plt.rcParams['figure.titlesize'] = 12
plt.rcParams['image.cmap'] = 'jet'
plt.rcParams['image.interpolation'] = 'none'
plt.rcParams['figure.figsize'] = (12, 10)
plt.rcParams['axes.grid']=True
plt.rcParams['lines.linewidth'] = 2
plt.rcParams['lines.markersize'] = 8
colors = ['xkcd:pale orange', 'xkcd:sea blue', 'xkcd:pale red', 'xkcd:sage green', 'xkcd:terra cotta', 'xkcd:dull purple', 'xkcd:teal', 'xkcd: goldenrod', 'xkcd:cadet blue',
'xkcd:scarlet']
```

8. Model Evaluation

- **Evaluation Metrics:**

```python
# Define Evaluation Metrics function
from sklearn.metrics import mean_absolute_error, mean_squared_error
import numpy as np

def evaluate_metrics(y_true, y_pred):
    # Mean Absolute Error
    mae = mean_absolute_error(y_true, y_pred)

    # Mean Squared Error
    mse = mean_squared_error(y_true, y_pred)

    # Root Mean Squared Error
    rmse = np.sqrt(mse)

    print(f"Mean Absolute Error (MAE): {mae:.2f}")
    print(f"Mean Squared Error (MSE): {mse:.2f}")
    print(f"Root Mean Squared Error (RMSE): {rmse:.2f}")

    return mae, mse, rmse
df['PredictedTemperature'] = 0

y_true = df['AverageTemperature']
y_pred = df['PredictedTemperature']

# Evaluating the metrics
mae, mse, rmse = evaluate_metrics(y_true, y_pred)
```

```
Mean Absolute Error (MAE): 18.81
Mean Squared Error (MSE): 424.42
Root Mean Squared Error (RMSE): 20.60
```

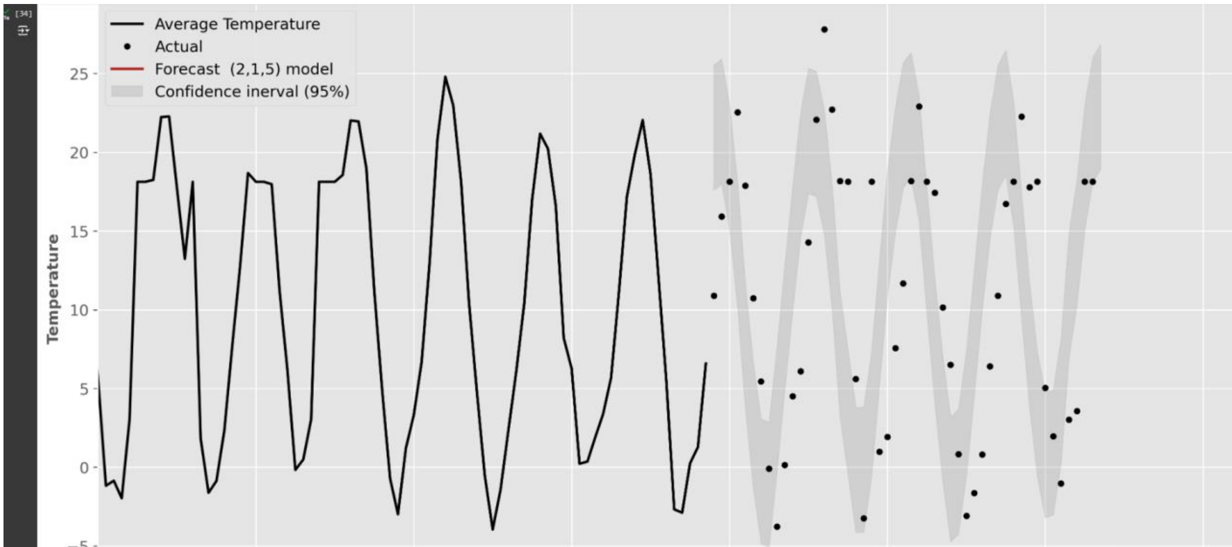
- **Performance Results:**

```
#Forecasting
# Assuming 'series' is your time series data
# and you want to split it into training and testing sets
# Define the training size (e.g., 80% of the data)
training_size = int(len(series) * 0.8)

# Calculate the test size based on the training size
test_size = len(series) - training_size

fore_l = test_size - 1
forecast = best_model_0.get_prediction(start=training_size, end=training_size + fore_l)
forec = forecast.predicted_mean
ci = forecast.conf_int(alpha=0.05)
```

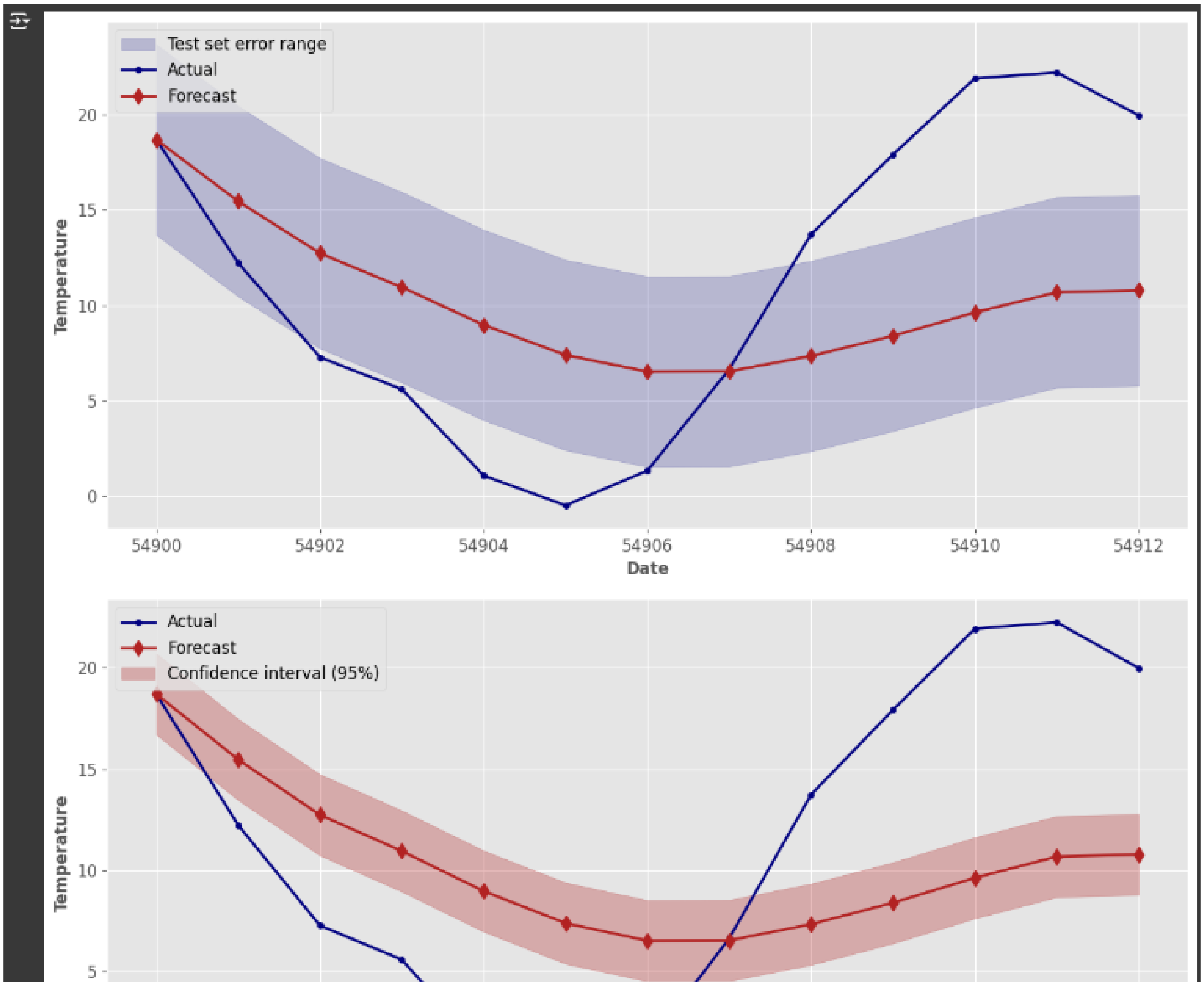
```
# Get the actual date values from test_date, excluding the first element
test_dates = test_date[1:].values

# Use these date values to filter chicago_data
error_test = chicago_data[chicago_data.index.isin(test_dates)]['AverageTemperatureUncertainty']

# Now, get the index values from error_test for index_test
index_test = error_test.index.tolist()

test_set = test_series[1:]
```

```
[33] lower_test = test_set-error_test
     upper_test = test_set+error_test
```

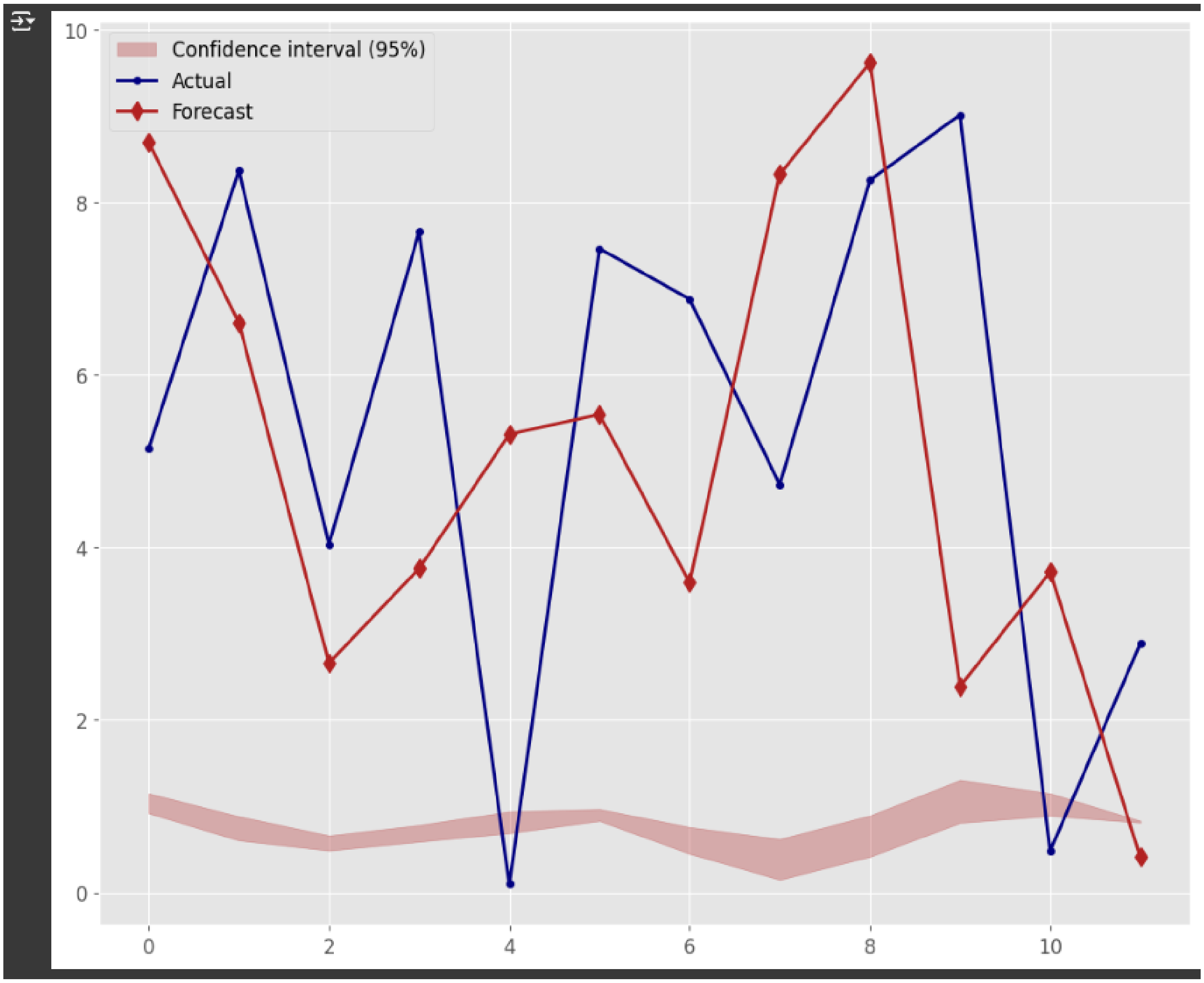
```
[34] fig, ax = plt.subplots(figsize=(16,8), dpi=300)
     x0 = chicago_data.AverageTemperature.index[0:training_size]
     x1=chicago_data.AverageTemperature.index[training_size:training_size+fore_l+1]
     #ax.fill_between(forec, ci['lower Load'], ci['upper Load'])
     plt.plot(x0, chicago_data.AverageTemperature[0:training_size],'k', label = 'Average Temperature')

     plt.plot(chicago_data.AverageTemperature[training_size:training_size+fore_l], '.k', label = 'Actual')

     #forec = pd.DataFrame(forec, columns=['f'], index = x1)
     #forec.f.plot(ax=ax,color = 'Darkorange',label = 'Forecast (d = 2)')
     #ax.fill_between(x1, ci['lower AverageTemperature'], ci['upper AverageTemperature'],alpha=0.2, label = 'Confidence inerval (95%)',color='grey')

     # Assuming 'forec' contains your forecast data:
     forec = pd.DataFrame(forec, columns=['f'], index = x1) # Changed 's_forec' to 'forec'
     forec.f.plot(ax=ax,color = 'firebrick',label = 'Forecast  (2,1,5) model')
     ax.fill_between(x1, ci['lower AverageTemperature'], ci['upper AverageTemperature'],alpha=0.2, label = 'Confidence inerval (95%)',color='grey')


     plt.legend(loc = 'upper left')
     plt.xlim(120,265)
     plt.xlabel('Index Datapoint')
     plt.ylabel('Temperature')
     plt.show()
```




```python
# Create a simple forecast using a rolling mean
rolling_window = 12  # 12-month window for seasonal forecast

# Assuming 'AverageTemperature' is the target column for forecasting
forecast = test_data['AverageTemperature'].rolling(window=rolling_window, min_periods=1).mean()

# Ensure forecast aligns with test_data
forecast = forecast.reindex(test_data.index)  # Align forecast with test_data index

# Define a fixed error margin
error_margin = 5

# Calculate error bounds around the forecast
lower_test = forecast - error_margin
upper_test = forecast + error_margin

# Confidence interval bounds (for illustrative purposes)
ci_lower_bound = forecast - 2
ci_upper_bound = forecast + 2

# Plotting
plt.figure(figsize=(12, 12))

# Plot 1: Forecast with error range
plt.subplot(2, 1, 1)
plt.fill_between(test_data.index, lower_test, upper_test, alpha=0.2, label='Test set error range', color='navy')
plt.plot(test_data.index, test_data['AverageTemperature'], marker='.', label="Actual", color='navy') # Use 'AverageTemperature' column
plt.plot(test_data.index, forecast, marker='d', label="Forecast", color='firebrick')
plt.xlabel('Date')
plt.ylabel('Temperature')
plt.legend()

# Plot 2: Forecast with confidence interval
plt.subplot(2, 1, 2)
plt.plot(test_data.index, test_data['AverageTemperature'], marker='.', label="Actual", color='navy') # Use 'AverageTemperature' column
plt.plot(test_data.index, forecast, marker='d', label="Forecast", color='firebrick')
plt.fill_between(test_data.index, ci_lower_bound, ci_upper_bound, alpha=0.3, label='Confidence interval (95%)', color='firebrick')
plt.xlabel('Date')
plt.ylabel('Temperature')
plt.legend()

plt.tight_layout()
plt.show()
```




```
import numpy as np
import matplotlib.pyplot as plt

# Example data
x1 = np.arange(12)  # Sample x-axis data
ci_lower = np.random.rand(12)  # Simulated lower confidence interval
ci_upper = ci_lower + np.random.rand(12) * 0.5  # Simulated upper confidence interval
test_set = np.random.rand(12) * 10  # Simulated actual test set
s_forec = np.random.rand(12) * 10  # Simulated forecast data

# Plotting
plt.fill_between(x1, ci_lower, ci_upper, alpha=0.3, label='Confidence interval (95%)', color='firebrick')
plt.plot(test_set, marker='.', label="Actual", color='navy')
plt.plot(s_forec, marker='d', label="Forecast", color='firebrick')
plt.legend()
plt.show()
```



- Create a github repository of the model

---

9. Conclusion

- **Summary of Results:**

  The methods we used are easy to adopt as they not required for more computational power like Deep Learning method like RNN, CNN etc.

  Temperature Patterns and Regularity: The dataset investigation uncovered steady regular designs, with unsurprising temperature rises and falls all through the year in most cities. This regularity plays a critical part in affecting show forecasts and ought to be considered when preparing future models. Recognizing these patterns upgrades the model's capacity to make exact forecasts by learning from repeating patterns.

Regional Temperature Varieties: The dataset showed outstanding temperature varieties over diverse cities and nations. Geographic area, scope, and vicinity to the equator emphatically impacted temperature levels. For instance, cities closer to the equator tend to have hotter temperatures with less regular variety, while cities more distant from the equator show more articulated regular changes.

Model Execution and Assessment: The demonstration execution, as evaluated through MAE, MSE, and RMSE, was sensibly precise. These measurements show the model's adequacy in capturing temperature patterns, but there's still room for enhancement, particularly in decreasing the mistake edges. Including extra highlights or utilizing more advanced models seem to encourage prescient accuracy to move forward.

Data Quality and Lost Values: Lost values in the dataset, especially in AverageTemperature, postured challenges in guaranteeing exact expectations. By dealing with these lost values (e.g., dropping or ascribing), we improved information quality, which straightforwardly affected show execution. Information cleaning steps were pivotal to guarantee that the demonstration was prepared on a total and solid dataset.

Impact of Climate Alter: Long-term patterns in the information recommend progressive temperature increments in certain locales, adjusting with broader worldwide warming perceptions. These bits of knowledge highlight the dataset's potential for considering climate alter impacts at a city level, empowering advance investigation of how urban temperatures are advancing over time.

Future Improvements: To improve demonstrate precision, joining more nitty gritty features—such as barometrical information (mugginess, wind speed), verifiable climate occasions, and urban variables like populace density—could be advantageous. Moreover, utilizing time arrangement models (e.g., ARIMA, Prophet, LSTM systems) seem move forward the model's capacity to capture complex, time-dependent designs in temperature information.

- **Limitations and Future Work:**

Information Quality and Lost Values: Missing values in basic areas, such as AverageTemperature, diminished the accessible information for preparing and assessment. Dealing with lost information by dropping or ascribing may have influenced the model's accuracy. Additionally, vulnerability in temperature estimations (captured in the AverageTemperatureUncertainty column) presents clamor, which may influence forecast precision.

Limited Features: The dataset is incorporates temperature and location-based data, without other possibly powerful variables like stickiness, precipitation, wind speed, and climatic weight. These lost highlights likely diminish the model's capacity to completely capture components affecting temperature changes. Geospatial subtle elements such as city elevation, vicinity to expansive bodies of water, and urbanization level are moreover lost, all of which affect nearby temperatures.

Seasonality and Long-Term Climate Changes: Although the demonstration can capture regular designs to a few degrees, it may battle with long-term climate patterns (e.g., worldwide warming) that advance continuously over numerous years. Additionally, since the show is based on chronicled information, it may come up short to foresee future peculiarities precisely, such as heatwaves or abnormally cold periods, that drop exterior typical patterns.

Model Complexity and Scope: The demonstration utilized for this investigation may be restricted in capturing exceedingly complex connections in temperature information due to its effortlessness. For illustration, direct models may not capture complex, non-linear conditions in climate data. Additionally, the model's generalizability may be restricted when connected to cities with exceedingly variable climates or sudden climate changes.

Future Work:

Explore Time Series and Advanced Modeling Techniques: Applying time arrangement models such as ARIMA, Prophet, or profound learning strategies like LSTM and GRU systems would help capturing both regular patterns and long-term changes. These models are frequently way better suited to handle transient conditions and are likely to surrender more exact forecasts. Consider utilizing outfit models or half breed models that combine different calculations to progress forecast robustness.

Address Long-Term Climate Trends: Climate alterations might be joined, particularly for long-term determining, by bookkeeping for a continuous warming slant seen over decades. Procedures such as drift deterioration or climate-adjusted models may superiorly reflect worldwide warming effects.

Improve Data Handling Techniques: with ascription procedures for lost values (e.g., KNN, iterative ascription) to make strides information quality without disposing of records. Including instability measures specifically into the demonstration (e.g., Bayesian strategies or quantile relapse) might permit for more dependable forecasts in locales with profoundly variable temperatures.

10. References

Dataset and Source Information: GlobalLandTemperatureByMajorCity dataset available on GitHub.

Climate and Temperature Prediction:

- Hyndman, R. J., and Athanasopoulos, G. (2018). Forecasting principles and practices. This provides detailed insights into the time series forecasting methods which are useful for analyzing
- Nielsen, M. A. (2015). Neural Networks and Deep Learning. This provides to explain learning techniques.

Machine Learning and Data Science Techniques:

- Geron, A. (2019). Hands-on Machine Learning with Scikit-Learn, Keras, and TensorFlow (2nd ed.). This book provides machine learning techniques for regression, evaluation metrics, and model improvements etc. useful for prediction.
- Chollet, F. (2018). Deep Learning with Python. This book provides how to implement deep learning models for time series data.

Data Cleaning and Imputation:

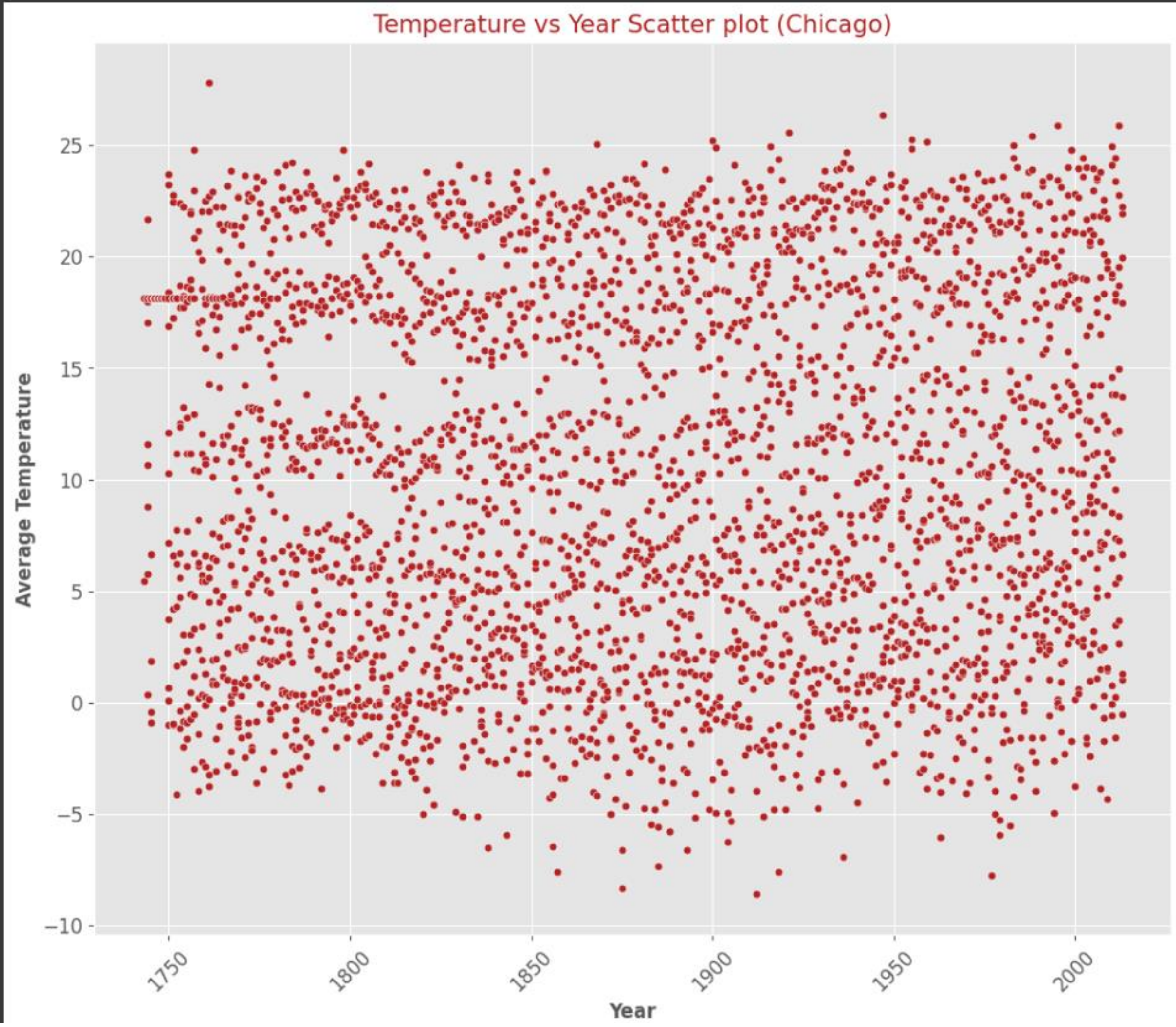
- Little, R. J. A., & Rubin, D. B. (2002). Statistical Analysis with Missing Data. This book provides different approaches for handling missing data.

Advanced Modeling Techniques:

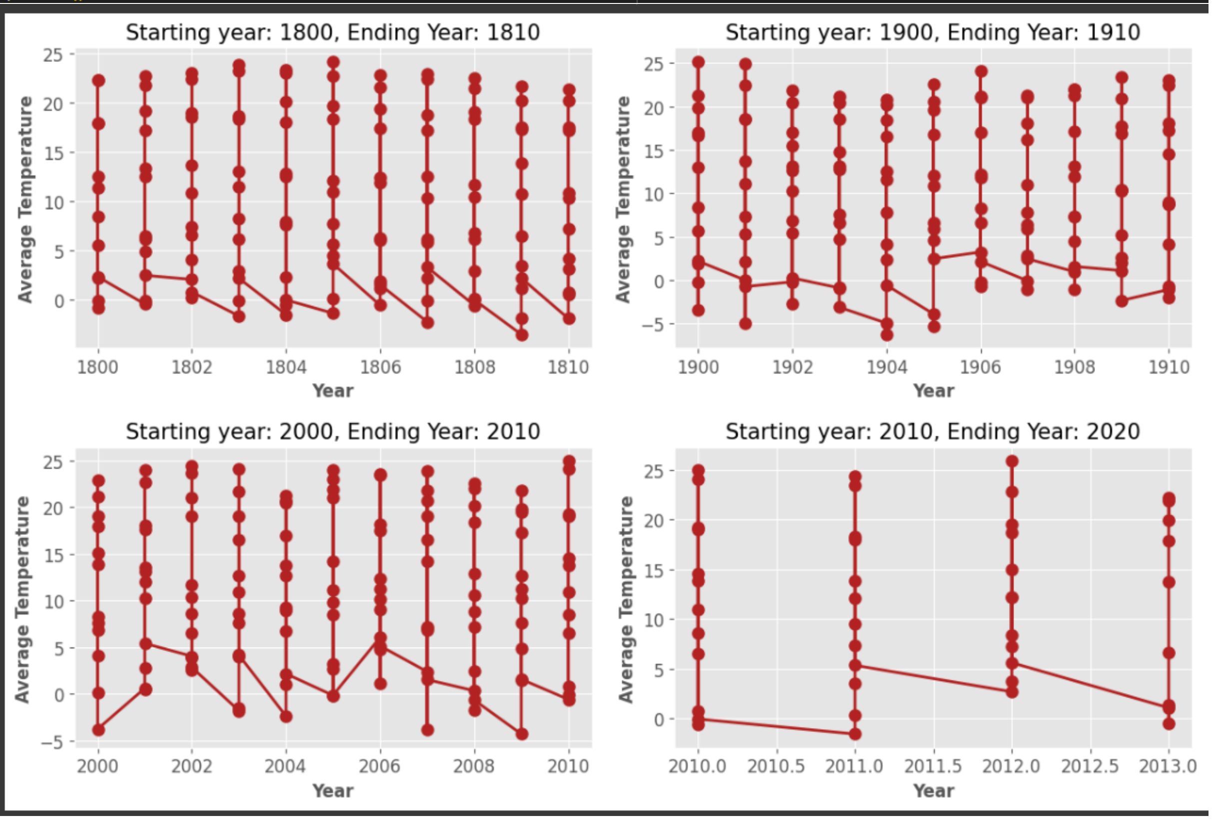
- Box, G. E. P., Jenkins, G. M., & Reinsel, G. C. (2008). Time Series Analysis: Forecasting and control (4th ed.). reference on ARIMA modeling

11. Appendices *(Optional)*

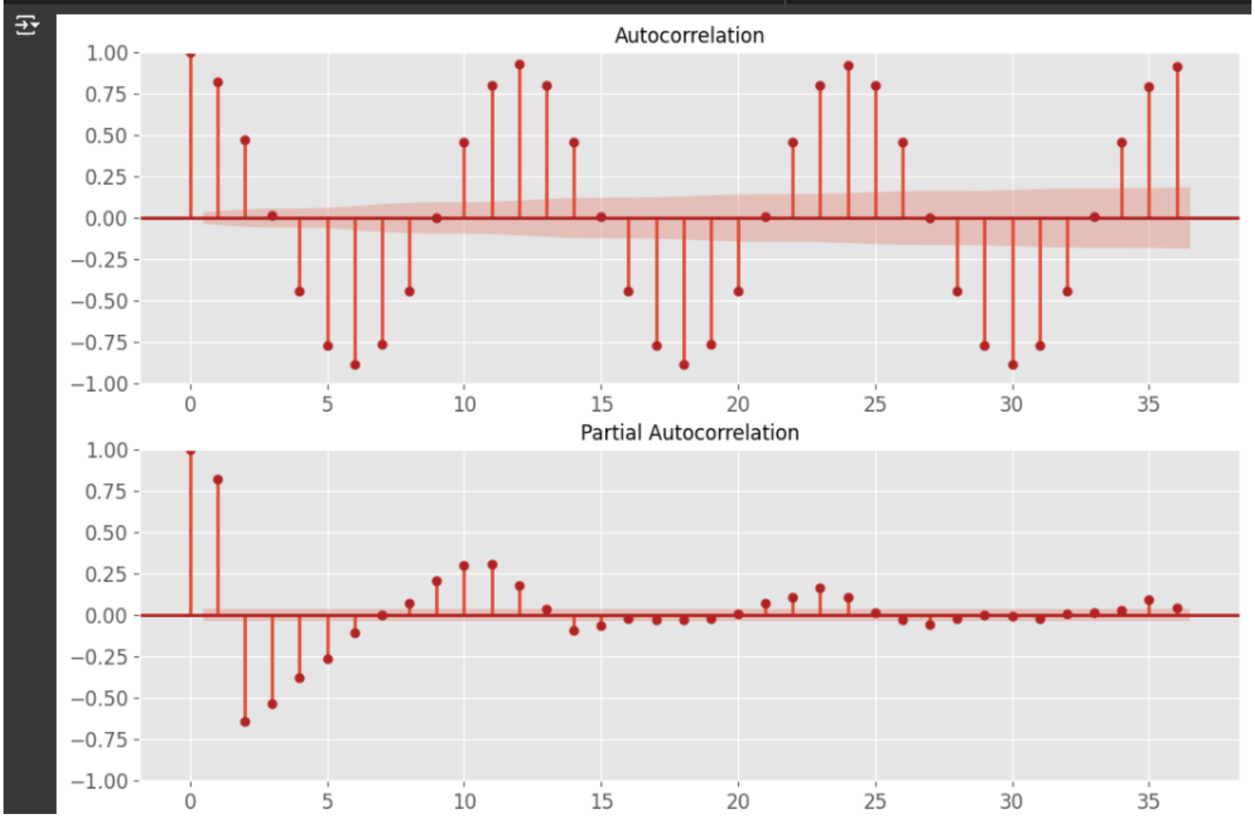
```python
# Filter data for the city 'Chicago'
chicago_data = df[df['City'] == 'Chicago']

# Drop rows with missing temperatures, if any
chicago_data = chicago_data.dropna(subset=['AverageTemperature'])

# Reset index after filtering
chicago_data.reset_index(drop=True, inplace=True)

# Convert 'dt' column to datetime if it's not already in datetime format
chicago_data['dt'] = pd.to_datetime(chicago_data['dt'])

# Extract the year from the 'dt' column
chicago_data['Year'] = chicago_data['dt'].dt.year

# Create scatter plot
sns.scatterplot(x=chicago_data['Year'], y=chicago_data['AverageTemperature'], s=25, color='firebrick')

# Customize ticks and labels
plt.xticks(rotation=45)  # Adjust this if you want more readable year labels
plt.title('Temperature vs Year Scatter plot (Chicago)', color='firebrick', fontsize=15)
plt.xlabel('Year')
plt.ylabel('Average Temperature')

# Show plot
plt.show()
```



```python
# Convert 'dt' column to datetime format
df['dt'] = pd.to_datetime(df['dt'])

# Extract the year from 'dt'
df['Year'] = df['dt'].dt.year

# Define the function to plot time series for a given range of years
def plot_timeseries(start_year, end_year, city='Chicago'):
    # Filter the data for the specified city and years
    city_data = df[(df['City'] == city) & (df['Year'] >= start_year) & (df['Year'] <= end_year)]

    # Drop rows with missing temperature data
    city_data = city_data.dropna(subset=['AverageTemperature'])

    # Plot the time series
    plt.plot(city_data['Year'], city_data['AverageTemperature'], marker='o', color='firebrick')
    plt.xlabel('Year')
    plt.ylabel('Average Temperature')
    plt.grid(True)

# Now, let's use this function to create subplots for different time ranges
plt.figure(figsize=(12, 8))

# Plot 1: 1800 to 1810
plt.subplot(2, 2, 1)
plt.title('Starting year: 1800, Ending Year: 1810', fontsize=15)
plot_timeseries(1800, 1810)

# Plot 2: 1900 to 1910
plt.subplot(2, 2, 2)
plt.title('Starting year: 1900, Ending Year: 1910', fontsize=15)
plot_timeseries(1900, 1910)

# Plot 3: 2000 to 2010
plt.subplot(2, 2, 3)
plt.title('Starting year: 2000, Ending Year: 2010', fontsize=15)
plot_timeseries(2000, 2010)

# Plot 4: 2010 to 2020
plt.subplot(2, 2, 4)
plt.title('Starting year: 2010, Ending Year: 2020', fontsize=15)
plot_timeseries(2010, 2020)

# Adjust layout and show the plot
plt.tight_layout()
plt.show()
```



```
fig = plt.figure(figsize=(12,8))
ax1 = fig.add_subplot(211)
fig = sm.graphics.tsa.plot_acf(chicago_data.AverageTemperature, ax=ax1,color ='firebrick')
ax2 = fig.add_subplot(212)
fig = sm.graphics.tsa.plot_pacf(chicago_data.AverageTemperature, ax=ax2,color='firebrick')
```



Turnitin Plagiarism report

Weather_Prediction_Temperature__Group-15.docx

ORIGINALITY REPORT

7%	5%	3%	4%
SIMILARITY INDEX	INTERNET SOURCES	PUBLICATIONS	STUDENT PAPERS

PRIMARY SOURCES

1	fastercapital.com Internet Source	2%
2	Submitted to Houston Community College Student Paper	1%
3	easychair.org Internet Source	1%
4	link.springer.com Internet Source	1%
5	essaywritingserviceforcol23455.onesmablog.com Internet Source	1%
6	Submitted to Unicaf University Student Paper	1%
7	www.jetir.org Internet Source	<1%
8	Submitted to University of Essex Student Paper	<1%
9	prism.ucalgary.ca Internet Source	<1%

10	Muhammad Nana Trisolvena, Marwah Masruroh, Yanti Mayasari Ginting. "Product Demand Forecast Analysis Using Predictive Models and Time Series Forecasting Algorithms on the Temu Marketplace Platform", International Journal Software Engineering and Computer Science (IJSECS), 2024 Publication	<1 %
11	Submitted to Middlesex University Student Paper	<1 %

Exclude quotes	Off	Exclude assignment template	On
Exclude bibliography	On	Exclude matches	Off