Weather Forecast

CS6240- Parallel Data Processing
Project Report

By

Gauri Damle
Sanchana Mohankumar

A report submitted to a faculty of

Northeastern University

April 2023

Table of Contents:

1. Introduction to the problem:

Weather forecast refers to the prediction of future weather conditions for a particular location or region. These predictions are based on the analysis of historical weather data, current weather conditions, and the use of various forecasting techniques and models. In this project we are going to forecast weather by using various data parameters.

Weather forecasting is important because it helps people make informed decisions about how to stay safe and how to plan their activities and operations in a variety of industries and sectors. Few of the details are mentioned below

Safety: Weather forecasting can help people prepare for hazardous weather conditions, such as thunderstorms, hurricanes, and tornadoes, which can cause property damage, injuries, and loss of life.

Planning: Weather forecasts are used by individuals, businesses, and organizations to plan their day-to-day activities, such as deciding what to wear, when to travel, and whether to hold outdoor events.

Agriculture: Farmers rely on weather forecasts to plan their planting and harvesting schedules and make decisions about irrigation, fertilization, and pest management.

Transportation: Weather forecasts help transportation companies, airlines, and airports plan for potential weather-related disruptions, such as flight cancellations or delays due to fog, snow, or thunderstorms.

Energy: Weather forecasts help energy companies plan for fluctuations in demand for heating or cooling due to changes in temperature, wind, and precipitation.

Our objective was to forecast future weather temperatures for all states by choosing the best model among ARIMA and LSTM. We used the Spark platform in AWS (Amazon Web Services) GLUE to run the jobs for modeling. We used PIG Latin scripts for data preprocessing, Hive to create tables using Hadoop EMR, and generated plots through HUE and Quick Sight to view the data and gain insights.

2. Data:

For retrieving our data, we specifically downloaded data from all states and concatenated the data we also performed various data cleaning and accessed Latitude Longitude data file and joined the columns using PIG which will be further explained in Major Task 1. Our data was retrieved as hourly data since we were forecasting temperature for next 30 days (about 4 and a half weeks), we converted to daily data.

Data Description

Element	Description	US	metric	UK
tempmax	Maximum Temperature	F	C	C
tempmin	Minimum Temperature	F	C	C
temp	Temperature (or mean temperature)	F	C	C
dew	Dew Point	F	C	C
feelslike	Feels like	F	C	C
precip	Precipitation	inches	mm	mm
precipprob	Precipitation chance	%	%	%
precipcover	Precipitation Cover	%	%	%
preciptype	Precipitation type	-	-	-
snow	Snow	inches	cm	cm
snowdepth	Snow Depth	inches	cm	cm
windspeed	Wind Speed	mph	kph	kph
windgust	Wind Gust	mph	kph	kph
winddir	Wind Direction	degrees	degrees	degrees
visibility	Visibility	miles	km	km
cloudcover	Cloud Cover	%	%	%
humidity	Relative Humidity	%	%	%
pressure	Sea Level Pressure	mb	mb	mb
solarradiation	Solar Radiation	W/m^2	W/m2	W/m2
solarenergy	Solar Energy	MJ/m^2	MJ/m^2	MJ/m^2
uvindex	UV Index	-	-	-
severerisk	Severe Risk	-	-	-
sunrise	Sunrise time	-	-	-
sunset	Sunset time	-	-	-
moonphase	Moonphase	-	-	-
icon	A weather icon	-	-	-
conditions	Short text about the weather	-	-	-
description	Description of the weather for the day	-	-	-
stations	List of weather stations sources	-	-	-

Link 1- https://www.visualcrossing.com/resources/documentation/weather-data/weather-data-documentation/

Link 2- https://www.visualcrossing.com/weather/weather-data-services/Washington/metric/2023-03-01/2023-03-27

The datasets contain the above-mentioned columns of historical data. The link above discusses the column's specifics and relevance in depth. To obtain data, we can choose data from a certain range and the place we specified.

3. Proposed Goals

- **Major Task 1:** We are transforming data in PIG by merging, filtering, then exporting as a csv file to Hive. We are building HQL scripts in Hive to display in HUE, Quick sight and analyze temperature data using transformed data obtained from PIG.
- **Helper Task**: Write DDL commands to create table schema.
- **Analysis Task**: We want to analyze the weather data and analyze how these variables can help in predicting weather patterns, understanding climate trends and forecast weather patterns for modelling.
- **Major Task 2**: Prediction of temperature using data mining libraries.

  Overview of Task: We are going to forecast the temperature by training the model with 80% of the data and test the forecast using the remaining 20% of the data.

  We will use the Arima, LSTM model and analyze model performance by changing the hyperparameters to get the optimum model. We will report on future temperature forecasts using this model. Lastly, compare model scalability using AWS Glue pyspark

4. Technical Discussion:

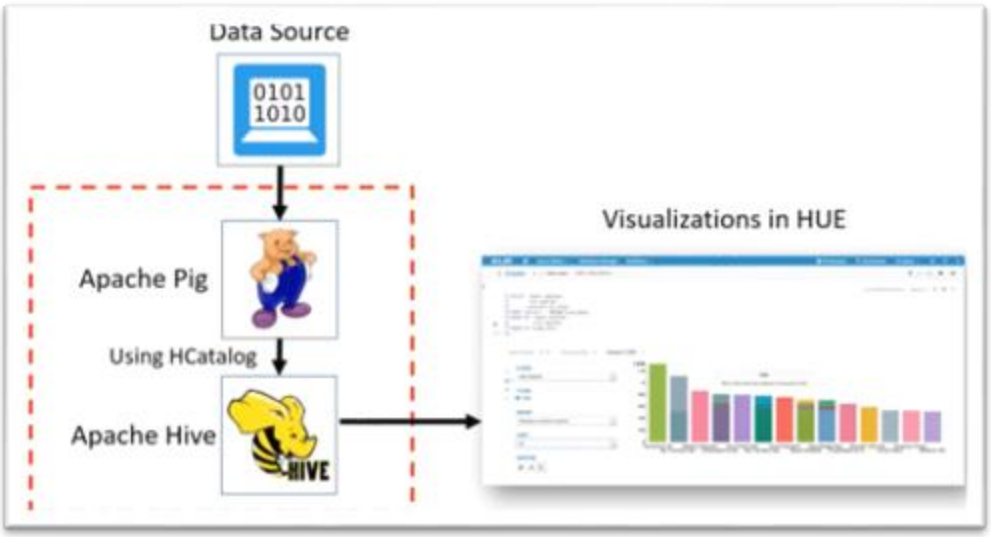


Fig 1: Architecture of Major Task1

- As shown in the workflow, we are using PIG and Hive to create data analysis pipeline.
- The data source is AWS S3 bucket and PIG and Hive scripts are running on AWS EMR Hadoop cluster.
- The data processing is done in Apache PIG script and the basetable is stored in HCatalog instead of again writing to AWS S3 bucket.
- This architecture is made efficient as Hive HCatalog table can share metadata between PIG and Hive increasing performance and reducing read/ write costs.

- Finally, the reporting views and visualizations are generated in HUE and AWS Quick Sight.
- Overall objective of performing the task1 is to solve real world data analytics task on AWS EMR data pipeline and provide insights in form of Dashboard.

4.1 Helper Task

- Creating Hive table with its metadata in HCatalog
- Prior to executing pig scripts to load data into Hive, we used this DDL script to create table schema.

```
create database db;

use db;

CREATE TABLE IF NOT EXISTS `basetable`(
  `name` string,
  `tempmax` float,
  `tempmin` float,
  `temp` float,
  `dew` float,
  `humidity` float,
  `precip` float,
  `snow` float,
  `windspeed` float,
  `sealevelpressure` float,
  `cloudcover` float,
  `visibility` float,
  `solarradiation` float,
  `uvindex` int,
  `datetimevalue` string,
  `state` string,
  `latitude` float,
  `longitude` float,
  `city` string
  )
ROW FORMAT DELIMITED
  FIELDS TERMINATED BY ','
TBLPROPERTIES (
  'skip.header.line.count'='1');
```

Fig 2: DDL Command

4.2 Major Task 1

4.2.1 Source Code

Section 1: Running PIG for data preprocessing

- We selected all 50 USA states data from 1973 to 2023 for this analytical work, with the goal of calculating the state wise average temperature, dew, humidity, precipitation and visualizing it on USA map.
- As seen in the code below, we have preprocessed and filtered the data for the pig job.
- First, we combined all 50 states CSV into single CSV using pig LOAD command and regular expression to match all 50 states CSVs.
- Secondly, we did feature selection by projecting only required columns and filtering the unnecessary ones.
- Lastly, we have used inner join in PIG Latin to merge each state code with its latitude and longitude coordinate so we can visualize it on USA map.

```
DEFINE CSVLoader org.apache.pig.piggybank.storage.CSVLoader;


-- Setting number of reducer tasks to 10 for computation step
SET default_parallel 10;


-- Loading the input from parameter for running in Amazon S3 and EMR
weather_all_states = LOAD 's3://aws-6240-class-homework-landing-zone/all_statesData/weather_data_*.csv' using CSVLoader();
weather_lat_long = LOAD 's3://aws-6240-class-homework-landing-zone/latitude_longitude_states/statelatlong.csv' using CSVLoader();


-- parsing only required columns as weather_all_states_data
weather_all_states_data = FOREACH weather_all_states GENERATE
$0 as name:chararray,
$2 as tempmax:float,
$3 as tempmin:float,
$4 as temp:float,
$6 as dew:float,
$9 as humidity:float,
$10 as precip:float,
$14 as snow:float,
$17 as windspeed:float,
$19 as sealevelpressure:float,
$20 as cloudcover:float,
$21 as visibility:float,
$22 as solarradiation:float,
$24 as uvindex:int,
$1 as datetimevalue:chararray;


lat_long_data = FOREACH weather_lat_long GENERATE
$0 as state: chararray,
$1 as latitude:float,
$2 as longitude:float,
$3 as city:chararray;
```

```
-- joining all states weather data with latitude longitude data
joined_data = JOIN weather_all_states_data BY LOWER(name), lat_long_data BY LOWER(city);



joined_data = foreach joined_data generate
                weather_all_states_data::name as name,
                        weather_all_states_data::tempmax as tempmax,
                        weather_all_states_data::tempmin as tempmin,
                        weather_all_states_data::temp as temp,
                        weather_all_states_data::dew as dew,
                weather_all_states_data::humidity as humidity,
                        weather_all_states_data::precip as precip,
                        weather_all_states_data::snow as snow,
                        weather_all_states_data::windspeed as windspeed,
                        weather_all_states_data::sealevelpressure as sealevelpressure,
                        weather_all_states_data::cloudcover as cloudcover,
                        weather_all_states_data::visibility as visibility,
                        weather_all_states_data::solarradiation as solarradiation,
                        weather_all_states_data::uvindex as uvindex,
                        weather_all_states_data::datetimevalue as datetimevalue,
                   lat_long_data::state as state,
                        lat_long_data::latitude as latitude,
                        lat_long_data::longitude as longitude,
                        lat_long_data::city as city;



DUMP joined_data;

-- Storing the output to hcatalog
STORE joined_data into 'db.basetable' using org.apache.hive.hcatalog.pig.HCatStorer();
```

Fig 4.2.1.1 Pig Latin script for preprocessing data

Section 2: Hive Analysis

- The pig script output could be saved to S3. However, we choose to communicate metadata between pig and Hive via hive HCatalog. This decreased the number of reads and writes to S3 for storing intermediate output significantly.
- The pig script mentioned above is loading data into the Hive HCatalog **basetable** which helper task created.
- Once **basetable** is populated, Hive script is used for further preprocessing **basetable** data.
- We have created different layered architecture in Hive warehouse.

basetable > **rawtable** (corrected datatypes) > **analysistable** (aggregated data) > **reportingview** (BI-layer for business users)

- In rawtable, datatypes for date, year, month, day are converted into correct formats.
- In analysistable, group by commands were executed for analysis, as shown in the code below.
- Finally, for visualization Hue plots were displayed in reportingview.

Fig 4.2.1.2: Hive Script

4.2.2 Performance Comparison

- We have used the above Pig and Hive scripts to run it in the **Hadoop EMR** cluster using 2 different configurations namely **3 Node cluster** and **6 Node cluster**.

Fig 4.2.2.1: Running Pig and Hive script on EMR using Hadoop with 1 Master and 6 slave Nodes configuration

Table 4.2.2.1 - Config 1 - 1 Primary and 6 core Nodes cluster:

Program	Time Taken	Program type
Step1	26 seconds	Hive Script - DDL
Step2	4 minutes 14 seconds	PIG Script - Data Processing
Step3	36 seconds	Hive Script - Analysis

Table 4.2.2.2 - Config 2- 1 Primary and 3 core Node cluster:

Program	Time Taken	Program type
Step1	27 seconds	Hive Script - DDL
Step2	5 minutes 24 seconds	PIG Script - Data Processing
Step3	37 seconds	Hive Script - Analysis

From the above Table 4.2.2.1 and Table 4.2.2.2 we can observe that Configuration 1 Time Taken is less compared to Config2 as we have more core nodes assigned to it which helps in running parallelly and faster.

4.2.3 Quick sight Plots

- For reporting views and visualizations, we used HUE as well as AWS Quick Sight.
- We first enabled HUE connection on EMR using SSH Port Tunneling, then exported HUE table data into Quick Sight to create plots.

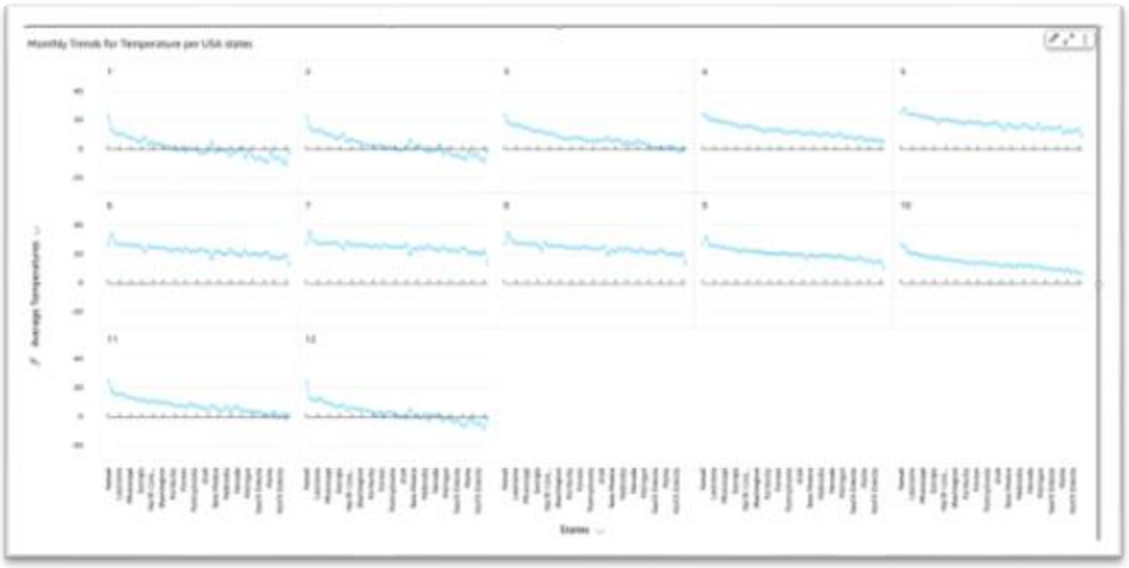


Fig 4.2.3.1 Monthly Trends in weather per state

- In this line chart, monthly trends from Jan to Dec for all years from 1973 to 2023 are provided.

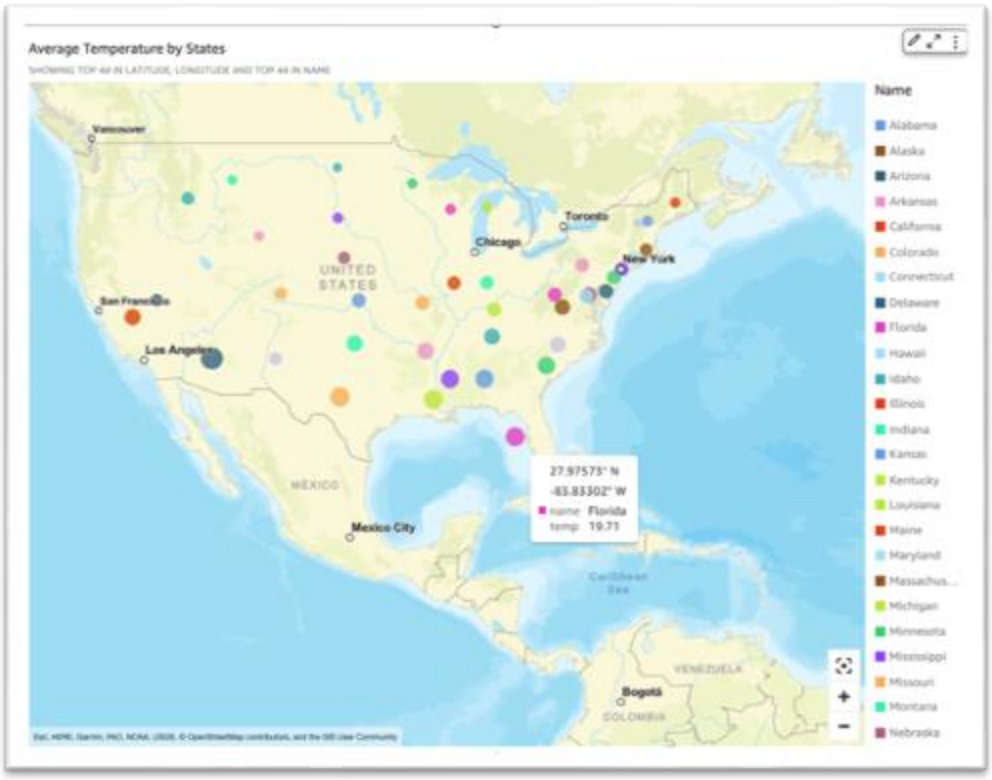


Fig 4.2.3.2: Average Temperature per state

- In this geographical map, the Average Temperature is plotted for all USA states.

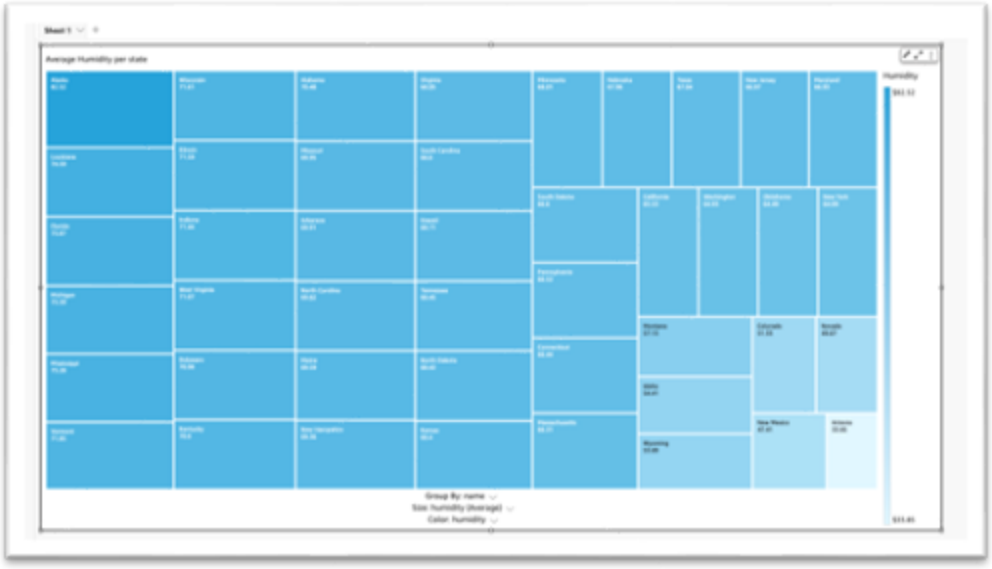


Fig 4.2.3.3: Average Humidity per state

- In this Heatmap, Average Humidity for different USA states is mentioned.

Fig 4.2.3.4: Average Precipitation per state

- In this plot, Average precipitation for different USA states is reported.

4.2.4 Delivarables

Step1

- createTableDDLS.q > source code
- stdoutput.pdf

Step2

- dataProcessing.pig > source code
- stdoutput.pdf
- syslog.pdf

Step3

- hiveDataAnalysis.q > source code
- Stdoutput.pdf

4.3 Analysis Task:

For our project we are working on Weather Forecast data to perform modelling of our data there are few preprocessing steps to consider. In our case we have chosen the ARIMA model considering it's a well suitable model for time series data. We have taken time series data from 1973-2023 and forecasting for next 30 days (about 4 and a half weeks). Before we do modelling, we perform the following steps

1. Data Preprocessing and EDA

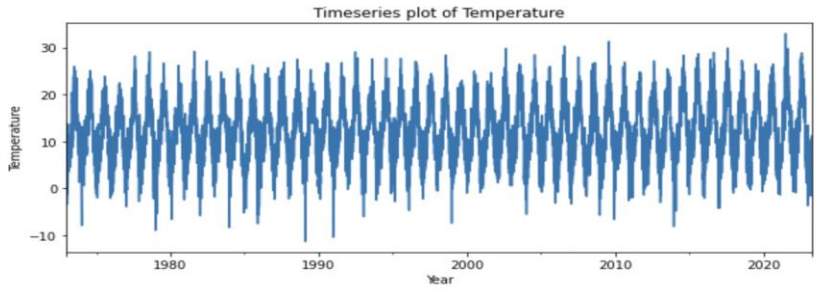


Figure 4.3.1 Time series plot of Data

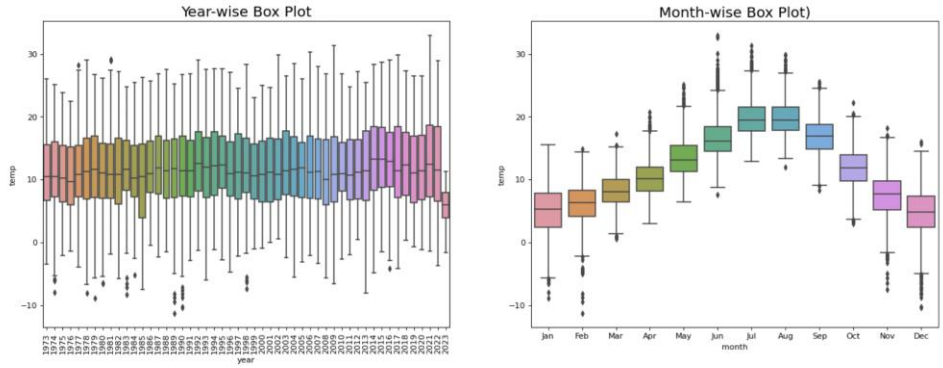
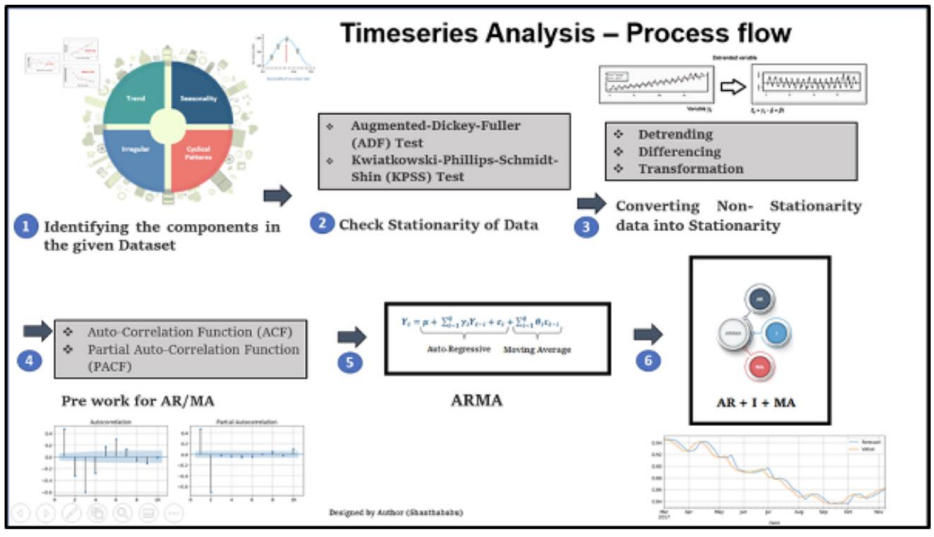


Figure 4.3.2 Year Wise and Month wise box plot

As we see in the above Fig 4.3.1 shows the timeseries data distribution and Fig 4.3.2 shows the distribution of monthly and yearly temperature

2. Statistical Test



As we can see in the above image Timeseries data undergoes a series of preprocessing steps so first we will have to check the trend, seasonality and stationery of data. If the data could not be

figured by time series plots visually, we will test using ADF Test to confirm the nature of our data.

Using the Augmented Dickey-Fuller test we determined the data is stationary, as p value is greater than 0.05, and the auto Arima library function to select the optimum model parameters as shown in Fig 4.3.4.

```
from statsmodels.tsa.stattools import adfuller
def ad_test(dataset):
    dftest = adfuller(dataset, autolag = 'AIC')
    print("1. ADF : ",dftest[0])
    print("2. P-Value : ", dftest[1])
    print("3. Num Of Lags : ", dftest[2])
    print("4. Num Of Observations Used For ADF Regression:", dftest[3])
    print("5. Critical Values :")
    for key, val in dftest[4].items():
        print("\t",key, ": ", val)
ad_test(df['temp'])
```

```
1. ADF :  -10.602891072057334
2. P-Value :  6.106400625860563e-19
3. Num Of Lags :  45
4. Num Of Observations Used For ADF Regression: 18320
5. Critical Values :
        1% :  -3.4307069987174152
        5% :  -2.861697780089129
        10% :  -2.5668539821686513
```

Fig 4.3.3: Augmented Dickey–Fuller test

```
from pmdarima import auto_arima
stepwise_fit = auto_arima(df['temp'], trace=True, suppress_warnings=True)
```

```
Best model:  ARIMA(1,1,3)(0,0,0)[0]
Total fit time: 66.512 seconds
```

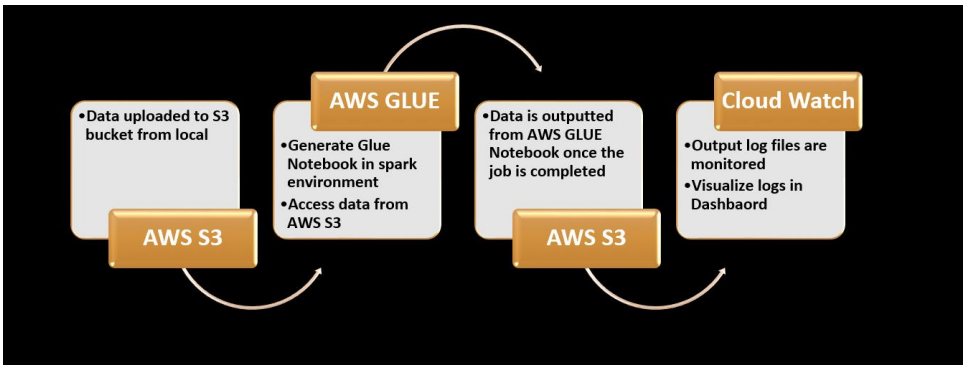
Fig 4.3.4: Choosing the best parameters for our model

Finally, we have finalized our model hyperparameters and finished the preprocessing step we will be moving to modelling part which is Major Tasl 2

4.4 Major Task 2

In this section we will be using the ARIMA and LSTM model to forecast temperature values from April 15, 2023, to May 15, 2023. For this task we have performed various preprocessing steps mentioned in Analysis Task followed by Modelling. This is the flow in which we performed the analysis, as shown in the Model Design below. For our modelling part we first tested our models in a local environment and selected the best performing model and executed them in AWS GLUE Notebook using Spark Environment with different workers.

Module Design



4.4.1 Source Code

Model 1: Arima Model

```
# create an empty DataFrame for storing predictions and evaluation results for all states
all_states_results = pd.DataFrame(columns=['name', 'Date', 'ARIMA Predictions', 'RMSE', 'MSE'])
# iterate over each state in pandasDF
for state in pandasDF['name'].unique():
    # filter data for the current state
    state_data = pandasDF[pandasDF['name'] == state]
    # convert temperature data to float
    state_data['temp'] = state_data['temp'].astype(float)
    # Data Train Test Split
    train = state_data.iloc[:-30]
    test = state_data.iloc[-30:]
    # ARIMA Model
    model = ARIMA(train['temp'], order=(1, 1, 3))
    model = model.fit()
    # Future Predictions
    index_future_dates = pd.date_range(start='2023-04-15', end='2023-05-15')
    pred = model.predict(start=len(state_data), end=len(state_data)+30, typ='levels').rename('ARIMA Predictions')
    pred.index = index_future_dates
    future_predictions = pd.DataFrame(pred)
    # reset_index
    future_predictions = future_predictions.reset_index()
    # Rename_column
    future_predictions = future_predictions.rename(columns={'index': 'Date'})
    # add State column
    future_predictions['name'] = state
    # evaluate RMSE and MSE
    rmse = np.sqrt(mean_squared_error(test['temp'], future_predictions['ARIMA Predictions'].iloc[-30:]))
    mse = mean_squared_error(test['temp'], future_predictions['ARIMA Predictions'].iloc[-30:])
    # add predictions and evaluation results to all_states_results DataFrame
    for index, row in future_predictions.iterrows():
        all_states_results = all_states_results.append({'name': state, 'Date': row['Date'],
                                                        'ARIMA Predictions': row['ARIMA Predictions'],
                                                        'RMSE': rmse, 'MSE': mse}, ignore_index=True)
```

Source code 4.4.1.1 Arima model for all states

Model 2: LSTM Model

```
states = ['Georgia', 'New Hampshire', 'Wisconsin', 'Minnesota',
       'Maryland', 'Pennsylvania', 'Texas', 'Florida', 'Michigan',
       'Hawaii', 'Tennessee', 'Arkansas', 'Ohio', 'Nebraska', 'Montana',
       'Indiana', 'New York', 'South Carolina', 'Mississippi',
       'North Carolina', 'Connecticut', 'Oklahoma', 'West Virginia',
       'Colorado', 'Maine', 'Louisiana', 'Utah', 'North Dakota',
       'Virginia', 'Delaware', 'Kentucky', 'Massachusetts', 'New Jersey',
       'Alabama', 'Arizona', 'California', 'Nevada', 'Alaska',
       'Rhode Island', 'Illinois', 'Idaho', 'Oregon', 'Iowa', 'Vermont',
       'Kansas', 'Washington', 'Missouri', 'New Mexico', 'South Dakota']

state_data = {}
train_data = {}
test_data = {}
train_X, train_y = {}, {}
test_X, test_y = {}, {}
scalers = {}
models = {}

for state in states:
    state_data[state] = data.loc[data['name'] == state]['temp'].values.reshape(-1, 1)

    # Pad all state arrays to have the same length as the longest array
    max_length = max([len(state_data[state]) for state in state_data])
    state_temp = state_data[state]
    num_rows_to_add = max_length - len(state_temp)
    state_temp = np.pad(state_temp, ((0,num_rows_to_add), (0,0)), 'constant')
    state_data[state] = state_temp

    if len(state_data[state]) == 0:
        print(f"No data for {state}, skipping")
        continue
```

Source code 4.4.1.2: Padding of data to maintain same length of array across all states

```
    # Scale the data for each state
    scaler = MinMaxScaler()
    state_data[state] = scaler.fit_transform(state_data[state])
    scalers[state] = scaler

    # Define the training and testing data for each state separately
    state_data_padded = state_data[state]
    train_size = int(len(state_data_padded) * 0.8)
    train_data[state] = state_data_padded[:train_size, :]
    test_data[state] = state_data_padded[train_size:, :]

    # Define the training and testing sets for each state separately
    state_train_X, state_train_y = [], []
    state_test_X, state_test_y = [], []

    for i in range(n_steps, len(train_data[state])):
        state_train_X.append(train_data[state][i-n_steps:i, 0])
        state_train_y.append(train_data[state][i, 0])
    for i in range(n_steps, len(test_data[state])):
        state_test_X.append(test_data[state][i-n_steps:i, 0])
        state_test_y.append(test_data[state][i, 0])

    train_X[state] = np.array(state_train_X)
    train_y[state] = np.array(state_train_y)
    test_X[state] = np.array(state_test_X)
    test_y[state] = np.array(state_test_y)

    # Reshape the training and testing sets
    train_X[state] = np.reshape(train_X[state], (train_X[state].shape[0], train_X[state].shape[1], 1))
    test_X[state] = np.reshape(test_X[state], (test_X[state].shape[0], test_X[state].shape[1], 1))
```

Source code 4.4.1.3: Train Test split

```
    # Define the LSTM model for each state separately
    model = Sequential()
    model.add(LSTM(50, return_sequences=True, input_shape=(n_steps, 1)))
    model.add(LSTM(50, return_sequences=True))
    model.add(LSTM(50))
    model.add(Dense(1))
    model.compile(optimizer='adam', loss='mse')
    models[state] = model

for state in states:
    print(f"Training model for {state}")
    state_data_len = state_data[state].shape[0]
    state_train_X = train_X[state][:n_steps*train_size]
    state_test_X = test_X[state][-n_steps:]
    state_train_y = train_y[state][:n_steps*train_size]
    state_test_y = test_y[state][-n_steps:]

    history = models[state].fit(state_train_X, state_train_y, epochs=3, batch_size=32, validation_data=(state_test_X, state_test_y))
```

Source code 4.4.1.4: LSTM model design with hyperparameter and model fit

```
# Initialize lists to store evaluation metrics
mse_scores = []
mae_scores = []
rmse_scores = []
state_names = []
# Loop over all states and evaluate the model for each
for state in states:
    # Evaluate the model on the test data for this state
    y_pred = models[state].predict(test_X[state])
    y_pred = scalers[state].inverse_transform(y_pred)
    y_true = scalers[state].inverse_transform(test_y[state].reshape(-1, 1))

    # Calculate the MSE, MAE, and RMSE for this state
    mse = mean_squared_error(y_true, y_pred)
    mae = mean_absolute_error(y_true, y_pred)
    rmse = mean_squared_error(y_true, y_pred, squared=False)

    # Print the evaluation metrics for this state
    print(f"Evaluation metrics for {state}:")
    print(f"MSE: {mse:.4f}")
    print(f"MAE: {mae:.4f}")
    print(f"RMSE: {rmse:.4f}")

    # Add the evaluation metrics for this state to the lists
    mse_scores.append(mse)
    mae_scores.append(mae)
    rmse_scores.append(rmse)
    state_names.append(state)

# Create a dataframe of evaluation metrics
df_metrics = pd.DataFrame({
    'State': state_names,
    'MSE': mse_scores,
    'MAE': mae_scores,
    'RMSE': rmse_scores
})
```

Source code 4.4.1.5: Evaluation metrics looping through all states

GLUE SETUP:

The below image is the screenshot of the setup of AWS GLUE Notebook Setup

Part 1:

```
#AWS Glue Studio Notebook
###Run this cell to set up and start your interactive session
%idle_timeout 2880
%glue_version 3.0
%worker_type G.1X
%number_of_workers 5

import sys
from awsglue.transforms import *
from awsglue.dynamicframe import DynamicFrame
from awsglue.utils import getResolvedOptions
from pyspark.context import SparkContext
from awsglue.context import GlueContext
from awsglue.job import Job

sc = SparkContext.getOrCreate()
glueContext = GlueContext(sc)
spark = glueContext.spark_session
job = Job(glueContext)
```

Example: Create a DynamicFrame from a table in the AWS Glue Data Catalog and display its schema

```
###Create a DynamicFrame from a table in the AWS Glue Data Catalog and display its schema
dyf = glueContext.create_dynamic_frame.from_options(
            connection_type="s3",
            connection_options={"paths": ["s3://project.final/weather_data.csv"], "recurse": False},
            format="csv",
            format_options={"withHeader": True},
            transformation_ctx="source_transformation_context",
        )
```

Example: Convert the DynamicFrame to a Spark DataFrame and display a sample of the data

```
###Convert the DynamicFrame to a Spark DataFrame and display a sample of the data¶
df = dyf.toDF()
###Convert Spark DataFrame to Pandas DataFrame
pandasDF = df.toPandas()
```

Source code 3.4.1.6 Data imported from S3 and changed to dynamic frame to Spark Data Frame and to Pandas Data Frame

- The Source code for model is inputted here

Part 2:

```
## Convert pandas Dataframe to Spark Dataframe
all_states_results = spark.createDataFrame(all_states_results)
## Uploading future Prediction output to S3
glueContext.write_dynamic_frame.from_options(
    frame = DynamicFrame.fromDF(all_states_results, glueContext, "output"),
    connection_type = "s3",
    connection_options = {"path": "s3://project.final/output/Worker_3/"},
    format = "csv",
    format_options = {"writeHeader": True}
)
```

Source code 4.4.1.7 The output executed as Data Frame is converted to Spark Data frame and back to Dynamic Frame and the output is send to AWS S3

Above as we can see there are 3 sections which are ARIMA and LSTM Model source code followed by AWS GLUE setup. In the GLUE Setup as we can see Part 1 is part of inputting data followed by you can input your model source code and finally Part 2 where the output data frame is again converted to spark data frame and exported to AWS S3.

The above code stores the RMSE, MSE, MAE values and future prediction values for the next 30 days (about 4 and a half weeks) in AWS S3. The output files are attached as mentioned in Deliverables below.

4.4.2 Performance Comparison

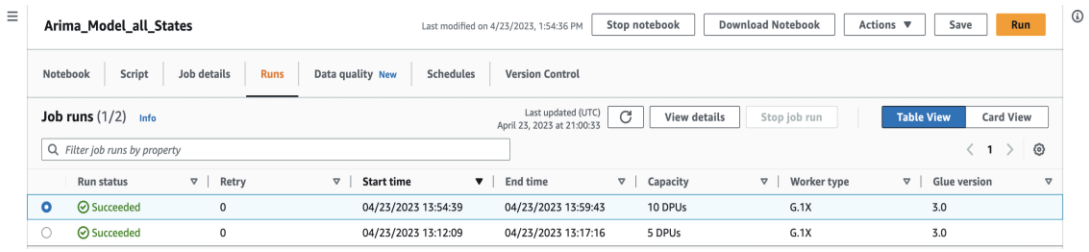


Figure 4.4.2.1 Successful job run of AWS Glue ARIMA Model with Workers 5 and 10

Table 4.4.2.1 Model Performance across Oregon state

Model	State	RMSE	MSE
Arima Model	Oregon	2.769	7.67
LSTM Model	Oregon	2.101	4.41

Table 4.4.2.2 ARIMA Model Performance with Worker 5 and 10

Model	Workers	Time Taken
Arima Model	5	4 minute 15 seconds
Arima Model	10	4 minutes

Table 4.4.2.1 demonstrates that there is not much of a difference in the model's performance, so we chose the ARIMA model as our best model performance because the LSTM model's implementation is more expensive and unsustainable over the long term, and because the RMSE value did not significantly decrease.

Table 4.4.2.2 demonstrates the model performance of AWS GLUE with worker 5 and 10, as we can see there is no significant difference between 2 cases, it's because when you assign a certain number of workers to your AWS Glue notebook, it is just a configuration setting that tells AWS Glue how many workers it should use, but the actual number of workers can vary based on the workload. If the workload increases, AWS Glue can add more workers to handle the additional load, and if the workload decreases, AWS Glue can remove workers to save resources.

This makes AWS GLUE dealing with workloads efficiently and cost-effectively but still we can configure a minimum and maximum number of workers to limit the range of worker adjustment by AWS Glue.

Figure 4.4.2.1 depicts the successful run of AWS GLUE jobs with worker 5 and 10

4.4.3 Delivarables

Arima_Model_AWS_GLUE -> contains all details about AWS GLUE output of worker 5 and 10, sour code, log files

local_run_source_code_Arima_LSTM.ipynb -> sour code of ARIMA and LSTM model

Weather_Forcast_Analysis_Task_source_code .ipynb -> sour code of Analysis Task

5. Challenges

Major Task 1 and Helper Task

- We opted to store output from Pig to Hive using HCatlog instead of using an intermediate process of storing data in AWS S3 as it was not efficient which costed more time and memory. Finally, we used HCatStorer() in pig script to store intermediate output in Hive metastore as Hive script can directly read from it.
- Issue faced while testing PIG-HIVE data pipeline indicating datatype mismatch. Example output in pig char-array should correspond to the string datatype in Hive. As date datatype is not basic datatypes in pig, we utilized char-array and processed the date in year, month, and date format using Hive UDFs

Major Task 2 and Analysis Task

- Seasonality was visible from the plot while observing the data, as shown in Figure 4.3.2, however our ADF test revealed there was no seasonality. To further test the model, we utilized the Sarimax model, which is for seasonal data, but we ended up receiving no seasonal order. This is because studies have shown that occasionally white noise prevents the seasonality trend from being exhibited clearly. To further validate, we utilized ACF, PACF plots and calculated lags to determine the a, d, and q values and manually included the seasonal order, but this did not enhance the performance, confirming that the data is not seasonal.
- Setting up Pyspark in locally consumed a lot of time so we ran it in AWS GLUE which has Pyspark environment. In which setting up Data CatLog pipeline to access data and run the AWS GLUE consumed lot of resources unnecessarily instead we directly accessed the data from S3.
- We further wanted to increase the model's accuracy, so we ran an LSTM model, which needed a lot of preprocessing processes to be taken care of. We also employed data from 50 US states, making it difficult and time-consuming to evaluate each state's data using a distinct model. With the ARIMA model, running the model was simpler and faster.

6. Conclusion

By observing Major task 1 and Major Task 2, In Major task 1 while using Hadoop EMR , we were able to manually configure the number of systems and had a lot of configurations in our control, whereas using AWS GLUE, even when we assign workers, the dynamic scaling model ensures that your AWS Glue notebook can handle varying workloads efficiently and affordably,

without requiring you to manually adjust the number of workers. The range of worker adjustment via AWS Glue may still be limited by configuring a minimum and maximum number of workers.

In this project, we used Serverless GLUE as well as EMR to observe scalability in processing big data. Overall, we succeeded in analyzing performance of tasks running in parallel processing environments and building scalable, end-to-end data processing pipelines from scratch.

7. Future Improvements

- In Major Task1, based on the increase in number of datapoints we can use AWS Load balancer to balance the incoming traffic into the system and automate the process of scaling.
- Further for Major Task 2, We can improve the performance of the model by exploring more custom models built by weather forecasting companies which will help in increasing accuracy of the prediction.
- We can access real time data through GLUE ETL and deploy model and visualize in Quick Sight Dashboard

8. Reference

https://www.visualcrossing.com/weather/weather-data-services/Washington/metric/2023-03-01/2023-03-27

https://github.com/nachi-hebbar/ARIMA-Temperature_Forecasting/blob/master/Temperature_Forecast_ARIMA.ipynb

https://www.freecodecamp.org/news/how-to-combine-multiple-csv-files-with-8-lines-of-code-265183e0854/

https://www.machinelearningplus.com/time-series/arima-model-time-series-forecasting-python/

https://www.kaggle.com/code/prashant111/complete-guide-on-time-series-analysis-in-python/notebook

Contribution:

Major Task 1, Helper task - 90% Gauri 10% Sanchana

Major Task 2, Analysis task- 90% Sanchana 10% Gauri