BIGDATA ANALYTICS FINAL PROJECT REPORT

Rainfall Prediction in the USA using Apache Spark

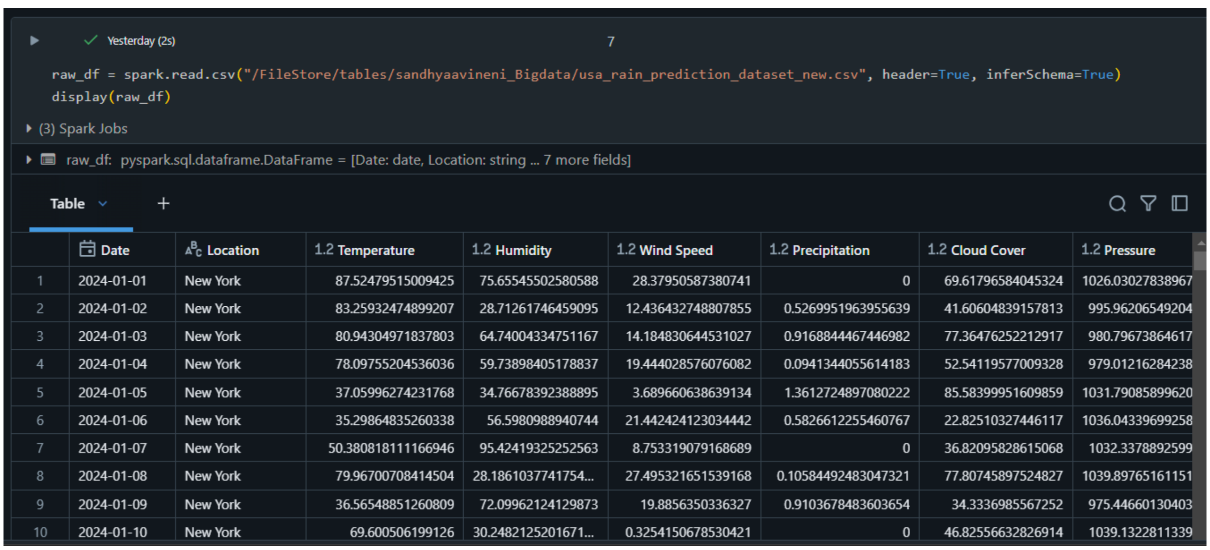
1. Introduction

Rainfall prediction is a crucial component in climate science, agriculture, and disaster preparedness. Accurately forecasting rainfall patterns can help mitigate risks associated with floods, droughts, and water resource management. This project aims to build a predictive model for rainfall in the USA using Apache Spark, leveraging distributed computing for efficient data processing and model training.

2. Dataset

The dataset used for this project is sourced from Kaggle: Rainfall Prediction USA. It contains historical meteorological data, including attributes such as temperature, humidity, wind speed, pressure, and precipitation levels.

The dataset was preprocessed to handle missing values, normalize features, and encode categorical variables to make it suitable for machine learning algorithms.



Yesterday (2s) 7

```
raw_df = spark.read.csv("/FileStore/tables/sandhyaavineni_Bigdata/usa_rain_prediction_dataset_new.csv", header=True, inferSchema=True)
display(raw_df)
```

(3) Spark Jobs

raw_df: pyspark.sql.dataframe.DataFrame = [Date: date, Location: string ... 7 more fields]

Table

	Date	Location	1.2 Temperature	1.2 Humidity	1.2 Wind Speed	1.2 Precipitation	1.2 Cloud Cover	1.2 Pressure
1	2024-01-01	New York	87.52479515009425	75.65545502580588	28.37950587380741	0	69.61796584045324	1026.03027838967
2	2024-01-02	New York	83.25932474899207	28.71261746459095	12.436432748807855	0.5269951963955639	41.60604839157813	995.96206549204
3	2024-01-03	New York	80.94304971837803	64.74004334751167	14.184830644531027	0.9168844467446982	77.36476252212917	980.79673864617
4	2024-01-04	New York	78.09755204536036	59.73898405178837	19.444028576076082	0.0941344055614183	52.54119577009328	979.01216284238
5	2024-01-05	New York	37.05996274231768	34.76678392388895	3.689660638639134	1.3612724897080222	85.58399951609859	1031.79085899620
6	2024-01-06	New York	35.29864835260338	56.5980988940744	21.442424123034442	0.5826612255460767	22.82510327446117	1036.04339699258
7	2024-01-07	New York	50.380818111166946	95.42419325252563	8.753319079168689	0	36.82095828615068	1032.3378892599
8	2024-01-08	New York	79.96700708414504	28.1861037741754...	27.495321651539168	0.10584492483047321	77.80745897524827	1039.89765161151
9	2024-01-09	New York	36.56548851260809	72.09962124129873	19.8856350336327	0.9103678483603654	34.3336985567252	975.44660130403
10	2024-01-10	New York	69.600506199126	30.2482125201671...	0.3254150678530421	0	46.82556632826914	1039.1322811339

Convert spark Data frame into Pandas Dataframe

Yesterday (4s)

```python
import plotly.express as px

# Convert Spark DataFrame to Pandas DataFrame
raw_df_pandas = raw_df.toPandas()

# Create scatter plot
fig = px.scatter(
    raw_df_pandas,
    x='Humidity',
    y='Temperature',
    color='Rain Tomorrow',
    opacity=0.5
)
fig.show()
```

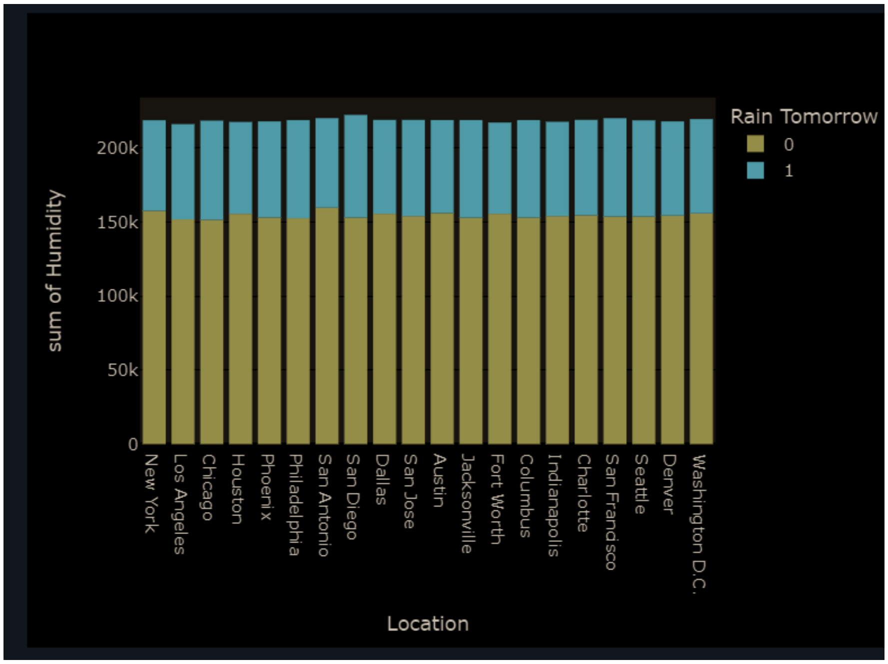
(1) Spark Jobs

Create the histogram

```python
# Ensure raw_df is a valid DataFrame
raw_df = raw_df.toPandas()


# Create the histogram
fig = px.histogram(
  raw_df,
  x='Location',
  y='Humidity',
  color='Rain Tomorrow'
)
fig.show()
```

Output-



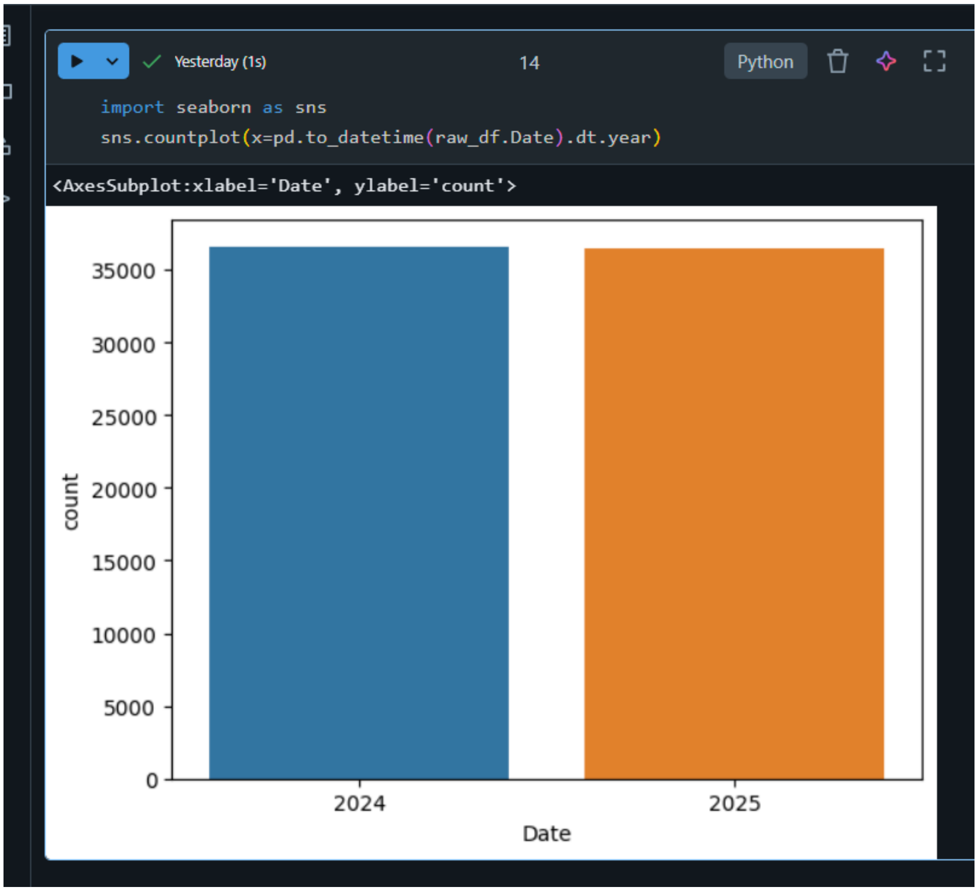
Df.info

Yesterday (<1s) 13

```
raw_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 73100 entries, 0 to 73099
Data columns (total 9 columns):
 #   Column          Non-Null Count  Dtype
---  ------          --------------  -----
 0   Date            73100 non-null  object
 1   Location        73100 non-null  object
 2   Temperature     73100 non-null  float64
 3   Humidity        73100 non-null  float64
 4   Wind Speed      73100 non-null  float64
 5   Precipitation   73100 non-null  float64
 6   Cloud Cover     73100 non-null  float64
 7   Pressure        73100 non-null  float64
 8   Rain Tomorrow   73100 non-null  int32
dtypes: float64(6), int32(1), object(2)
memory usage: 4.7+ MB
```

This code snippet uses the Seaborn library in Python to create a count plot visualizing the distribution of data points across different years.



Result:The plot shows two bars, one for the year 2024 and another for 2025. The height of each bar corresponds to the number of data points (rows in the DataFrame) that fall into that year. This gives a quick visual representation of the distribution of data across the two years. The y-axis label ("count") clarifies that the height represents the frequency of each year.

Yesterday (<1s) 31

```python
from sklearn.preprocessing import OneHotEncoder
```

Yesterday (<1s) 32

```python
train_inputs[categorical_cols]= train_inputs[categorical_cols].fillna('unknown')
val_inputs[categorical_cols]= val_inputs[categorical_cols].fillna('unknown')
test_inputs[categorical_cols]= test_inputs[categorical_cols].fillna('unknown')
```

Yesterday (<1s) 33

```python
encoder = OneHotEncoder(sparse=False, handle_unknown='ignore').fit(train_inputs
[categorical_cols])
```

Yesterday (<1s) 34

```python
train_inputs
```

This line imports the OneHotEncoder class from the sklearn.preprocessing module. sklearn (scikit-learn) is a popular Python library for machine learning. OneHotEncoder is used to convert categorical variables into a numerical representation suitable for machine learning algorithms.

These lines handle missing values in the categorical features. Assuming train_inputs, val_inputs, and test_inputs are DataFrames or similar structures containing training, validation, and testing data respectively, and categorical_cols is a list of column names representing categorical features:

train_inputs

This line likely represents a point where the training data is being processed further. The train_inputs DataFrame (or equivalent) will *not* yet contain the one-hot encoded features at this point. Further code (not shown) would be necessary to apply the trained encoder (encoder) to transform the training, validation, and testing data. The transformation would typically use the transform() method of the encoder object on each dataset separately.

In summary, this code prepares categorical features for machine learning by (1) handling missing values, (2) creating a one-hot encoder, and (3) fitting the encoder to the training data. The actual one-hot encoding of the datasets is expected to happen in subsequent code.

3. Algorithms Used

Several machine learning algorithms were implemented and evaluated to predict rainfall:

- **Logistic Regression**: Used as a baseline to understand relationships between meteorological features and rainfall.
- Logistic regression is a statistical model predicting the probability of a binary outcome. It uses a sigmoid function to map linear combinations of predictor variables to probabilities between 0 and 1. The model estimates coefficients through maximum likelihood estimation. It's widely used for classification tasks, distinguishing between two groups. Interpretation focuses on the odds ratios of predictors. Logistic regression assumes a linear relationship between predictors and the log-odds of the outcome.

```
Yesterday (<1s)    38
from sklearn.linear_model import LogisticRegression
```

```
Yesterday (<1s)    39
model=LogisticRegression(solver='liblinear').fit(train_inputs[numeric_cols+encoded_cols],train_targets)
```

```
Yesterday (<1s)    40
print(numeric_cols, encoded_cols)

['Temperature', 'Humidity', 'Wind Speed', 'Precipitation', 'Cloud Cover', 'Pressure'] ['Location_Austin', 'Location_Charlotte', 'Location_Chicago',
'Location_Columbus', 'Location_Dallas', 'Location_Denver', 'Location_Fort Worth', 'Location_Houston', 'Location_Indianapolis', 'Location_Jacksonvill
e', 'Location_Los Angeles', 'Location_New York', 'Location_Philadelphia', 'Location_Phoenix', 'Location_San Antonio', 'Location_San Diego', 'Location
_San Francisco', 'Location_San Jose', 'Location_Seattle', 'Location_Washington D.C.']
```

- **Decision Trees**: Captured non-linear dependencies between variables.

Yesterday (<1s)

```
model.fit(x_val, val_targets)
```

```
DecisionTreeClassifier
DecisionTreeClassifier(random_state=42)
```

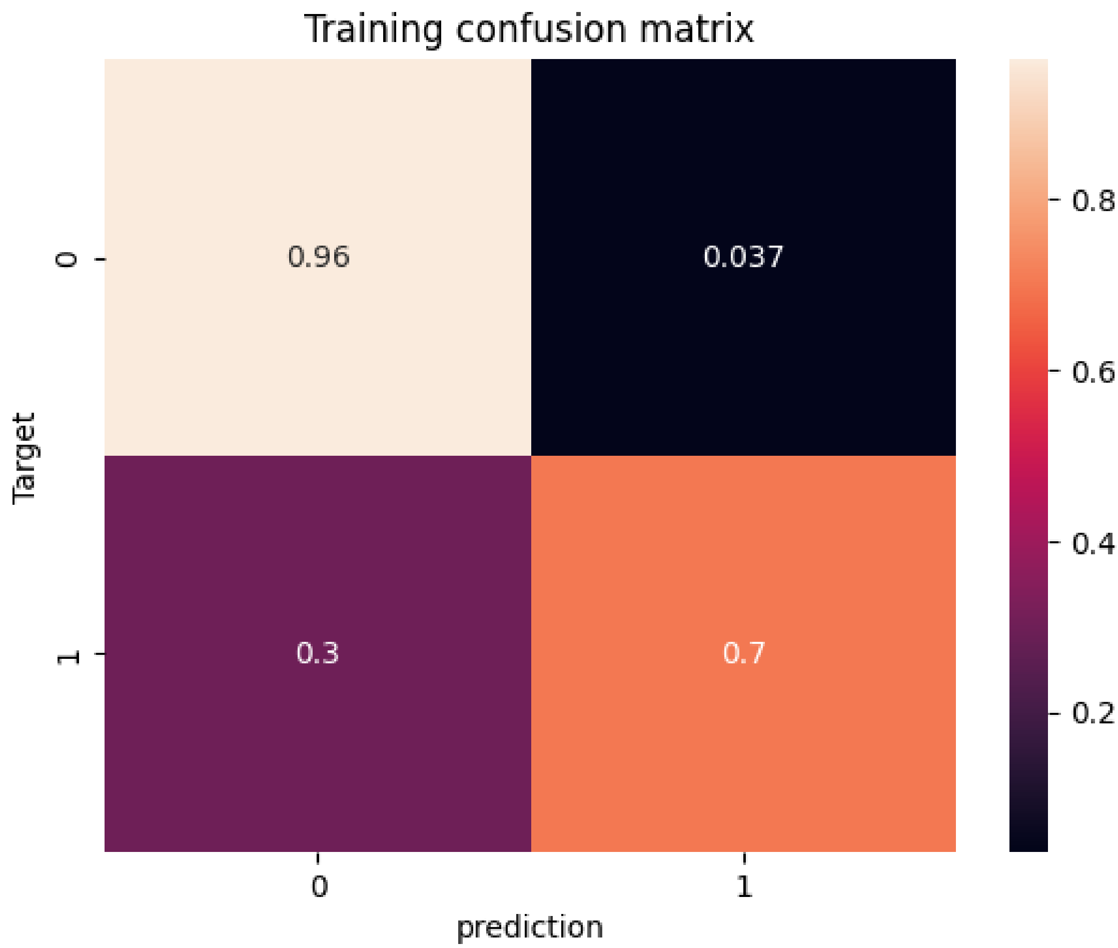
Yesterday (<1s)

```
val_preds=model.predict(x_val)
val_preds
```

```
array([0, 0, 0, ..., 0, 0, 0], dtype=int32)
```

- the training and prediction process using a **Decision Tree Classifier** for classification tasks.
- **Key Takeaways:**
- The model may be **overfitting** or not learning properly, as it predicts only a single class (all zeros).
- Further investigation is needed, possibly possible solutions include:
    - Adjusting **hyperparameters** (e.g., max depth, min samples split).
    - Checking for **class imbalance** in the dataset.
    - Trying a different **classification algorithm**.

Training confusion matrix



- Top-Left (0.96): This cell indicates that 96% of the instances belonging to class 0 were correctly predicted as class 0 (True Positives for class 0).
- Top-Right (0.037): This shows that only 3.7% of instances belonging to class 0 were incorrectly classified as class 1 (False Positives).
- Bottom-Left (0.3): 30% of instances belonging to class 1 were incorrectly predicted as class 0 (False Negatives).
- Bottom-Right (0.7): 70% of instances belonging to class 1 were correctly classified as class 1 (True Positives for class 1).

4. Data Processing Pipeline

The following steps outline the data processing and modeling pipeline implemented in Apache Spark:

1. **Data Ingestion**: Loaded the dataset into DataFrame.
2. **Data Cleaning**: Handled missing values, removed duplicates, and filtered out irrelevant data.
3. **Feature Engineering**: Encoded categorical features, normalized numerical values, and derived new features.
4. **Data Splitting**: Divided the dataset into training and testing sets (80-20 split).
5. **Model Training**: Trained various machine learning models using Spark MLlib.
6. **Hyperparameter Tuning**: Used cross-validation to optimize model parameters.
7. **Model Evaluation**: Assessed performance using accuracy metrics.

5. Scalability Experiments

To analyze the scalability of the pipeline, we experimented with different numbers of partitions and reducers:

- Increasing partitions improved data parallelism, reducing processing time.
- Adjusting reducer counts impacted on model training efficiency.

Model evaluation & Accuracy score

Yesterday (<1s)

```
val_preds=model.predict(x_val)
accuracy_score(val_targets, val_preds)
```

```
0.9033149171270718
```

Yesterday (<1s)

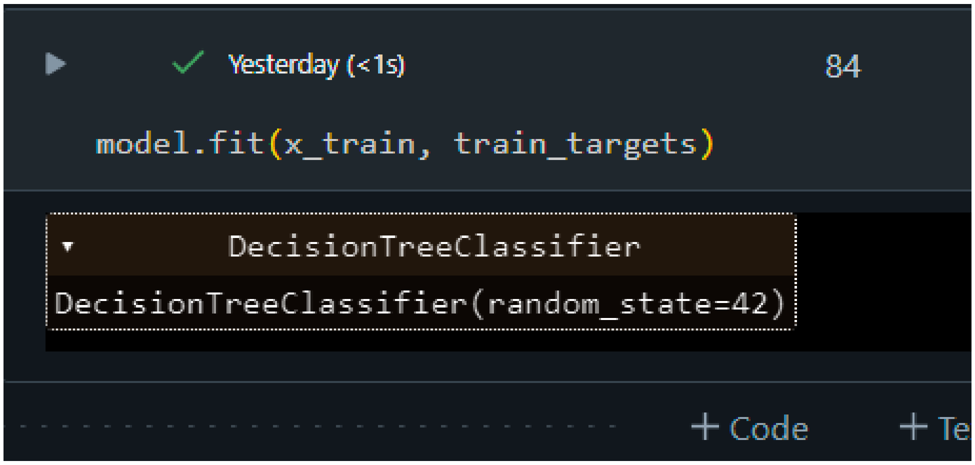
```
test_preds=model.predict(x_test)
accuracy_score(test_targets,test_preds)
```

```
0.9030434782608696
```

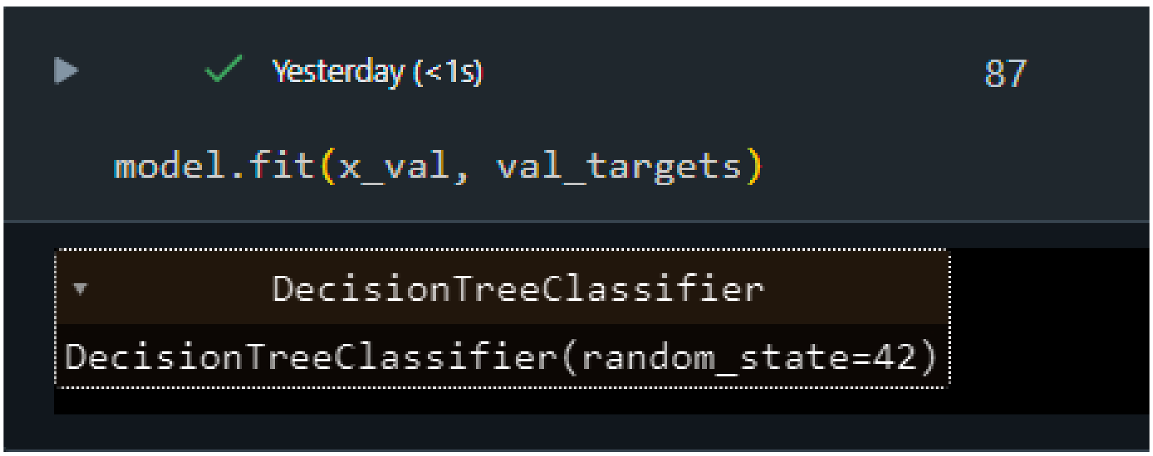
- **Key Takeaways:**
- The model achieves high accuracy on both validation (**90.33%**) and test (**90.30%**) datasets.
- Similar accuracy values suggest that the model is **not overfitting** and generalizes well.
- Logistic regression effectively classifies data based on learned patterns

Training set evaluation

1. model.fit(x_train, train_targets): This line trains a decision tree classification model.
    - model: This is a variable (likely instantiated earlier) representing the decision tree classifier. The random_state=42 argument ensures reproducibility of the results.
    - x_train: This represents the training data features.
    - train_targets: This represents the corresponding target variables (the correct classifications) for the training data.



Validation set evaluation



model.fit(x_val, val_targets): This line trains the *same* decision tree classifier but this time using a validation set (x_val, val_targets). Note that the model is being trained on the validation set instead of on the training set. This seems incorrect. The model should be trained once, on the training set, and then used to evaluate performance on a validation set.

6. Conclusion and Future Work

This project successfully implemented a rainfall prediction model using Apache Spark, demonstrating the efficiency of distributed computing for large-scale meteorological data processing. Future work includes:

- Integrating additional weather parameters from satellite data sources like MODIS and Sentinel.
- Implementing deep learning approaches (e.g., LSTMs) for better temporal pattern recognition.

- Deploying the model as a real-time weather prediction service using Spark Streaming.

This project highlights the potential of big data analytics in climate prediction and disaster management, paving the way for more sophisticated and accurate forecasting models.