Riding Through Seasons: Analyzing Capital Bikeshare and Weather Dynamics

Report by Sakshi Pandit, Saksham Verma, Yahe Yang, Feichen Yu

Table of Contents

Introduction

Bikeshare systems like Capital Bikeshare have become integral to sustainable urban transportation in Washington, DC, offering an eco-friendly travel option for thousands of residents and visitors. Beyond convenience, riding a bike provides significant personal health benefits. According to the CDC, regular physical activity, such as cycling, can reduce the risk of chronic diseases, improve mental health, and enhance overall quality of life (CDC). However, external factors, particularly weather conditions, shape usage patterns. This project examines the interplay between bike-share usage and weather variables such as temperature, precipitation, and seasonal changes, aiming to provide data-driven insights into user behavior and operational challenges. By understanding how external factors influence bike-share ridership, we can promote healthier and more sustainable urban living.

Project Objectives

The primary objective of this project is to analyze how weather conditions influence Capital Bikeshare usage. By leveraging comprehensive datasets from September 2023 to August 2024, the study explores trends across different weather scenarios and user types. Understanding these patterns will enable bike-share operators to optimize their services and urban planners to develop resilient and adaptive infrastructure.

Project Relevance

Capital Bikeshare is a convenient mode of transport and a key component of Washington, DC's efforts to promote sustainable urban living. However, weather conditions often impact ridership, leading to fluctuating demand and operational inefficiencies. By integrating bike-share trip data with weather metrics, this project offers a deeper understanding of how environmental factors influence user behavior, ultimately helping stakeholders make informed decisions to enhance service reliability and user satisfaction.

Project Goals

This project aims to uncover actionable insights by:

1. Identifying patterns in bike-share usage based on weather conditions.
2. Distinguishing behavioral differences between casual riders and annual members.
3. Providing recommendations for bike-share operators to better adapt to weather-driven demand shifts.
4. Supporting urban planners in designing infrastructure that accommodates and promotes resilient bike-share usage.

Through this analysis, the project seeks to empower stakeholders to make informed, proactive decisions that enhance the efficiency, accessibility, and sustainability of the Capital Bikeshare system.

Data Management

Dataset

The analyzed data framework incorporates four primary datasets to investigate Capital Bikeshare usage patterns in DC area from 2020 to 2024. The core dataset from Capital Bikeshare's system provides granular trip-level records detailing individual rides through unique identifiers, temporal data, station information, and user classification, enabling comprehensive analysis of usage patterns and service distribution.

The Visual Crossing Weather API contributes detailed meteorological data for the DC area from September 2023 to August 2024, encompassing temperature metrics, precipitation parameters, atmospheric conditions, and solar radiation measurements. This weather dataset offers a high-resolution environmental context for analyzing the impact of weather conditions on bike-share utilization.

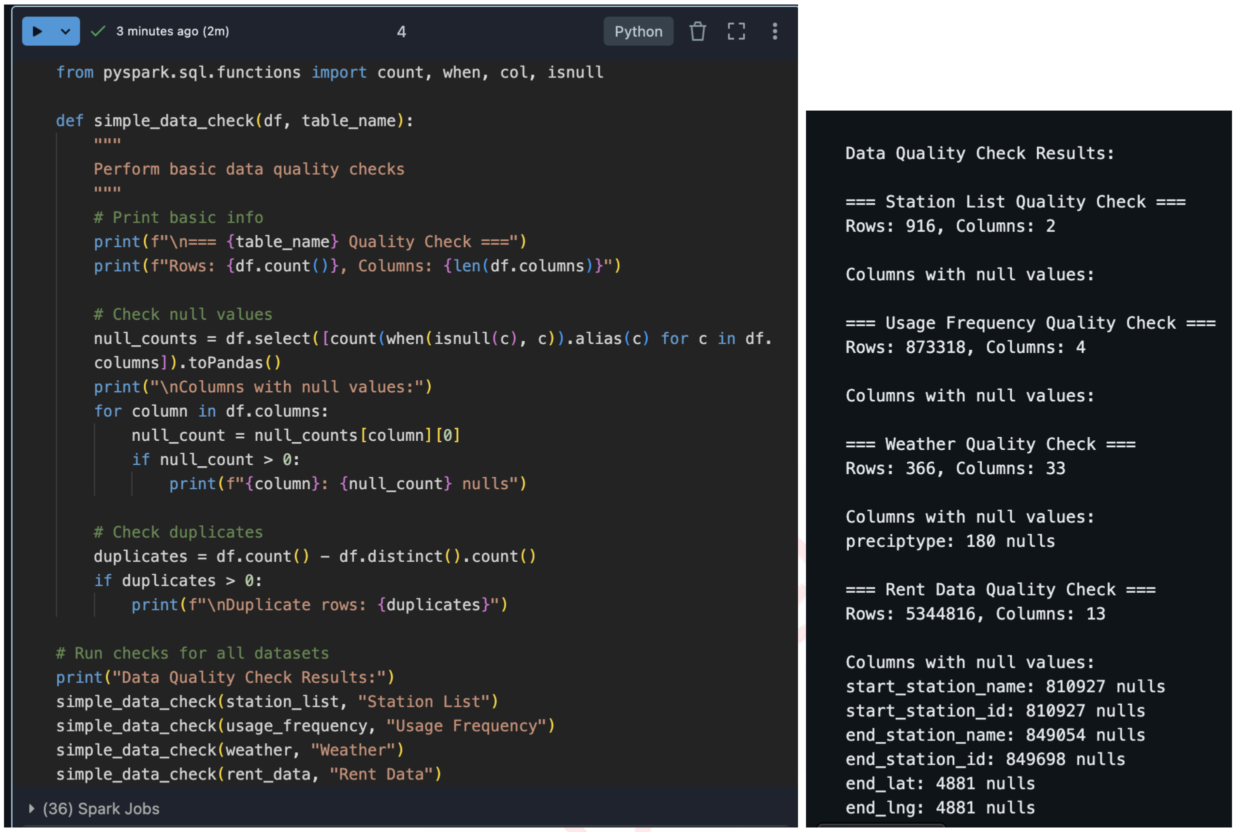
A consolidated dataset available through Kaggle integrates historical bike-share usage with corresponding weather conditions from May 2020 through August 2024. This integration facilitates longitudinal analysis of ridership patterns, station utilization rates, and weather-dependent usage variations, providing a robust foundation for investigating temporal and environmental influences on bike-share service dynamics.

ETL - Extract, Transform, Load

The data extraction phase leverages three primary data sources in CSV format: weather data from Visual Crossing's API, ridership information from Amazon S3, and consolidated historical data from Kaggle. This comprehensive data collection approach ensures coverage of operational and environmental factors affecting Capital Bikeshare usage.

Using Apache Spark in Databricks, the extracted data undergoes transformation and loading processes where raw CSV files are converted into optimized database schemas with appropriate data types. This distributed processing framework enables efficient handling of large-scale data, with weather metrics stored as double-precision values and categorical variables as strings, preparing the datasets for subsequent preprocessing and statistical analysis in the Databricks environment.

Data Quality Checking


3 minutes ago (2m) 4 Python

```python
from pyspark.sql.functions import count, when, col, isnull

def simple_data_check(df, table_name):
    """
    Perform basic data quality checks
    """
    # Print basic info
    print(f"\n=== {table_name} Quality Check ===")
    print(f"Rows: {df.count()}, Columns: {len(df.columns)}")

    # Check null values
    null_counts = df.select([count(when(isnull(c), c)).alias(c) for c in df.
    columns]).toPandas()
    print("\nColumns with null values:")
    for column in df.columns:
        null_count = null_counts[column][0]
        if null_count > 0:
            print(f"{column}: {null_count} nulls")

    # Check duplicates
    duplicates = df.count() - df.distinct().count()
    if duplicates > 0:
        print(f"\nDuplicate rows: {duplicates}")

# Run checks for all datasets
print("Data Quality Check Results:")
simple_data_check(station_list, "Station List")
simple_data_check(usage_frequency, "Usage Frequency")
simple_data_check(weather, "Weather")
simple_data_check(rent_data, "Rent Data")
```

(36) Spark Jobs

```
Data Quality Check Results:

=== Station List Quality Check ===
Rows: 916, Columns: 2

Columns with null values:

=== Usage Frequency Quality Check ===
Rows: 873318, Columns: 4

Columns with null values:

=== Weather Quality Check ===
Rows: 366, Columns: 33

Columns with null values:
preciptype: 180 nulls

=== Rent Data Quality Check ===
Rows: 5344816, Columns: 13

Columns with null values:
start_station_name: 810927 nulls
start_station_id: 810927 nulls
end_station_name: 849054 nulls
end_station_id: 849698 nulls
end_lat: 4881 nulls
end_lng: 4881 nulls
```

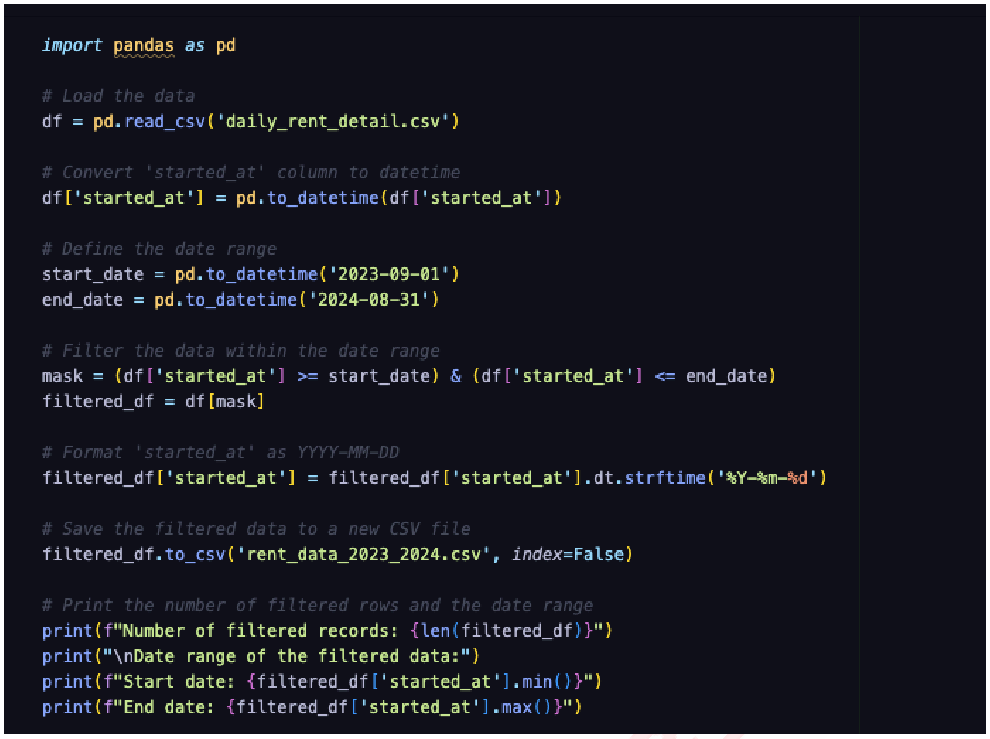
Figure 1. Data Quality Checking for Null Values

Data quality assessment was performed on four key datasets through checks for null values and duplicates, revealing varying completeness levels across the station list (916 locations), usage frequency (873K records), weather data (366 daily records), and rental data (5.3M entries). The analysis identified specific data gaps in station information, coordinates, and precipitation types, providing crucial insights for developing targeted data cleaning and preprocessing strategies in subsequent analysis phases.

Data Preprocessing

Data Selection

The data selection process focused on isolating Capital Bikeshare usage data for one year from September 2023 through August 2024, establishing a consistent temporal boundary for the analysis. The dataset was loaded from a daily rental detail CSV file, with datetime columns appropriately formatted to facilitate accurate temporal filtering.

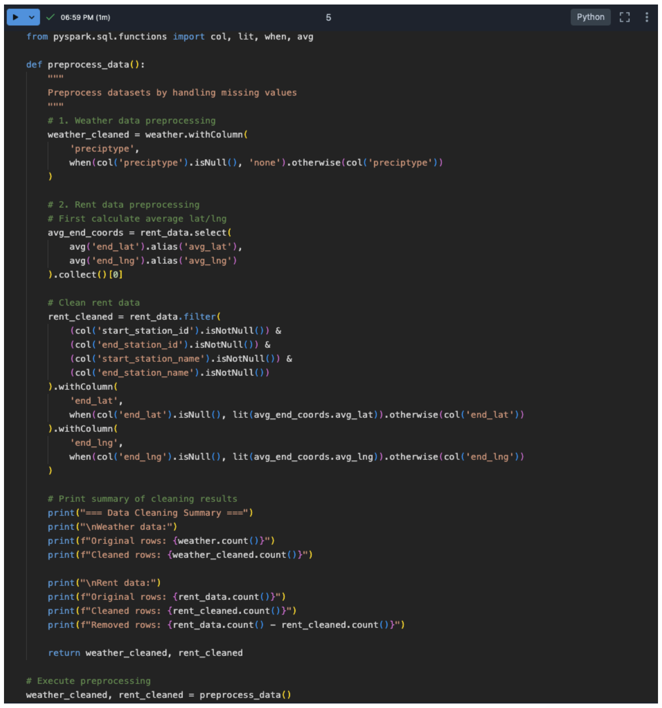


```python
import pandas as pd

# Load the data
df = pd.read_csv('daily_rent_detail.csv')

# Convert 'started_at' column to datetime
df['started_at'] = pd.to_datetime(df['started_at'])

# Define the date range
start_date = pd.to_datetime('2023-09-01')
end_date = pd.to_datetime('2024-08-31')

# Filter the data within the date range
mask = (df['started_at'] >= start_date) & (df['started_at'] <= end_date)
filtered_df = df[mask]

# Format 'started_at' as YYYY-MM-DD
filtered_df['started_at'] = filtered_df['started_at'].dt.strftime('%Y-%m-%d')

# Save the filtered data to a new CSV file
filtered_df.to_csv('rent_data_2023_2024.csv', index=False)

# Print the number of filtered rows and the date range
print(f"Number of filtered records: {len(filtered_df)}")
print("\nDate range of the filtered data:")
print(f"Start date: {filtered_df['started_at'].min()}")
print(f"End date: {filtered_df['started_at'].max()}")
```

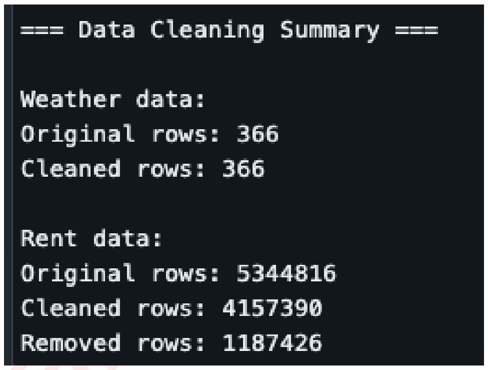
Figure 2. Data Selection with Date Range Filtering

The raw data was processed using pandas through a series of transformations, including datetime conversion, range-based filtering, and standardized date formatting in YYYY-MM-DD format. The filtered dataset was then exported to a new CSV file, preserving the processed temporal boundaries for subsequent analysis stages while maintaining data integrity.

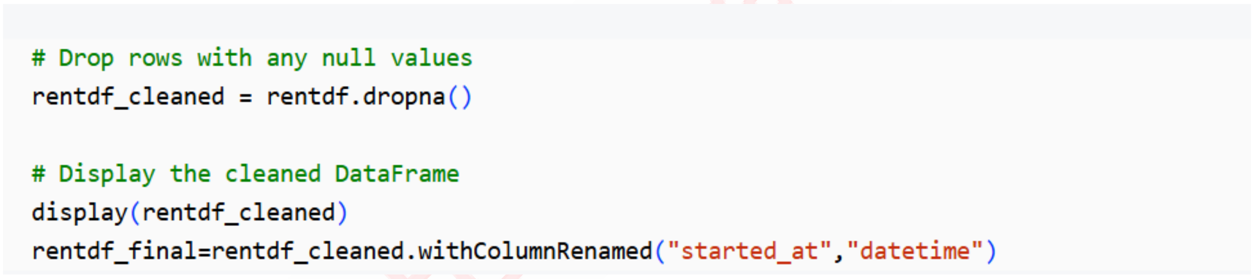
Data Cleaning



```
06:59 PM (1m)    5    Python
from pyspark.sql.functions import col, lit, when, avg

def preprocess_data():
    """
    Preprocess datasets by handling missing values
    """
    # 1. Weather data preprocessing
    weather_cleaned = weather.withColumn(
        'preciptype',
        when(col('preciptype').isNull(), 'none').otherwise(col('preciptype'))
    )

    # 2. Rent data preprocessing
    # First calculate average lat/lng
    avg_end_coords = rent_data.select(
        avg('end_lat').alias('avg_lat'),
        avg('end_lng').alias('avg_lng')
    ).collect()[0]

    # Clean rent data
    rent_cleaned = rent_data.filter(
        (col('start_station_id').isNotNull()) &
        (col('end_station_id').isNotNull()) &
        (col('start_station_name').isNotNull()) &
        (col('end_station_name').isNotNull())
    ).withColumn(
        'end_lat',
        when(col('end_lat').isNull(), lit(avg_end_coords.avg_lat)).otherwise(col('end_lat'))
    ).withColumn(
        'end_lng',
        when(col('end_lng').isNull(), lit(avg_end_coords.avg_lng)).otherwise(col('end_lng'))
    )

    # Print summary of cleaning results
    print("=== Data Cleaning Summary ===")
    print("\nWeather data:")
    print(f"Original rows: {weather.count()}")
    print(f"Cleaned rows: {weather_cleaned.count()}")

    print("\nRent data:")
    print(f"Original rows: {rent_data.count()}")
    print(f"Cleaned rows: {rent_cleaned.count()}")
    print(f"Removed rows: {rent_data.count() - rent_cleaned.count()}")

    return weather_cleaned, rent_cleaned

# Execute preprocessing
weather_cleaned, rent_cleaned = preprocess_data()
```



```
=== Data Cleaning Summary ===

Weather data:
Original rows: 366
Cleaned rows: 366

Rent data:
Original rows: 5344816
Cleaned rows: 4157390
Removed rows: 1187426
```



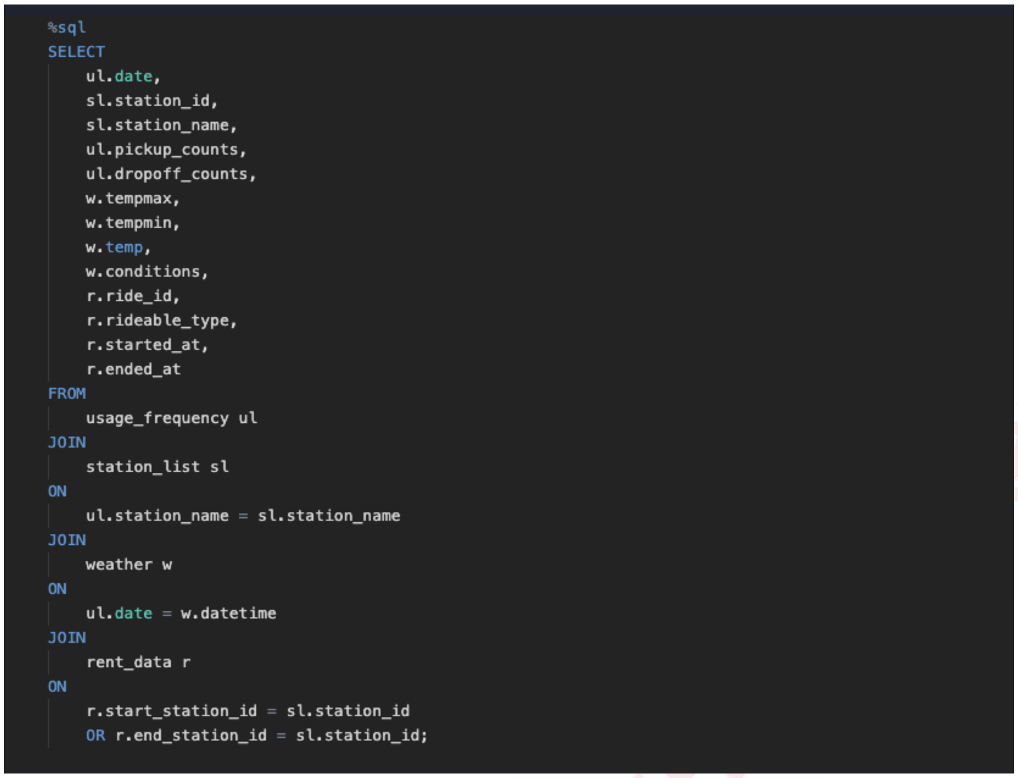
```
# Drop rows with any null values
rentdf_cleaned = rentdf.dropna()

# Display the cleaned DataFrame
display(rentdf_cleaned)
rentdf_final=rentdf_cleaned.withColumnRenamed("started_at","datetime")
```

Figure 3. Data Cleaning and Record Validation Results

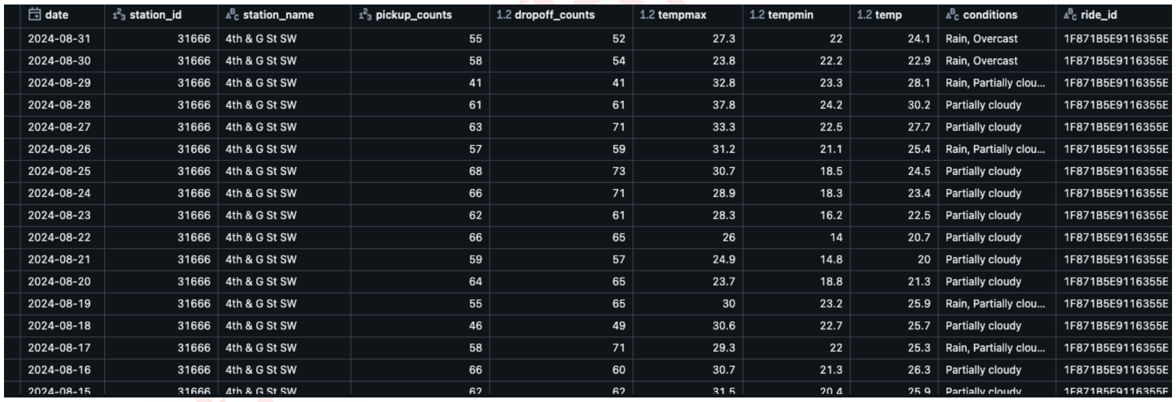
The data cleaning process addressed quality issues across two primary datasets: weather data and rental records. While the weather dataset maintained its original 366 daily records through the cleaning process, the rental data underwent substantial cleaning that reduced the dataset from 5.34 million to 4.16 million records, with approximately 1.19 million incomplete or invalid entries removed.

The cleaning procedure focused on handling missing values in critical fields such as station information (IDs and names), coordinate data (latitude and longitude), and other essential attributes. The implementation utilized SQL functions to efficiently process large-scale data, ensuring data consistency while preserving the integrity of valid records for subsequent analysis stages.

Data Merging



```
%sql
SELECT
    ul.date,
    sl.station_id,
    sl.station_name,
    ul.pickup_counts,
    ul.dropoff_counts,
    w.tempmax,
    w.tempmin,
    w.temp,
    w.conditions,
    r.ride_id,
    r.rideable_type,
    r.started_at,
    r.ended_at
FROM
    usage_frequency ul
JOIN
    station_list sl
ON
    ul.station_name = sl.station_name
JOIN
    weather w
ON
    ul.date = w.datetime
JOIN
    rent_data r
ON
    r.start_station_id = sl.station_id
    OR r.end_station_id = sl.station_id;
```



date	station_id	station_name	pickup_counts	dropoff_counts	tempmax	tempmin	temp	conditions	ride_id
2024-08-31	31666	4th & G St SW	55	52	27.3	22	24.1	Rain, Overcast	1F871B5E9116355E
2024-08-30	31666	4th & G St SW	58	54	23.8	22.2	22.9	Rain, Overcast	1F871B5E9116355E
2024-08-29	31666	4th & G St SW	41	41	32.8	23.3	28.1	Rain, Partially clou...	1F871B5E9116355E
2024-08-28	31666	4th & G St SW	61	61	37.8	24.2	30.2	Partially cloudy	1F871B5E9116355E
2024-08-27	31666	4th & G St SW	63	71	33.3	22.5	27.7	Partially cloudy	1F871B5E9116355E
2024-08-26	31666	4th & G St SW	57	59	31.2	21.1	25.4	Rain, Partially clou...	1F871B5E9116355E
2024-08-25	31666	4th & G St SW	68	73	30.7	18.5	24.5	Partially cloudy	1F871B5E9116355E
2024-08-24	31666	4th & G St SW	66	71	28.9	18.3	23.4	Partially cloudy	1F871B5E9116355E
2024-08-23	31666	4th & G St SW	62	61	28.3	16.2	22.5	Partially cloudy	1F871B5E9116355E
2024-08-22	31666	4th & G St SW	66	65	26	14	20.7	Partially cloudy	1F871B5E9116355E
2024-08-21	31666	4th & G St SW	59	57	24.9	14.8	20	Partially cloudy	1F871B5E9116355E
2024-08-20	31666	4th & G St SW	64	65	23.7	18.8	21.3	Partially cloudy	1F871B5E9116355E
2024-08-19	31666	4th & G St SW	55	65	30	23.2	25.9	Rain, Partially clou...	1F871B5E9116355E
2024-08-18	31666	4th & G St SW	46	49	30.6	22.7	25.7	Partially cloudy	1F871B5E9116355E
2024-08-17	31666	4th & G St SW	58	71	29.3	22	25.3	Rain, Partially clou...	1F871B5E9116355E
2024-08-16	31666	4th & G St SW	66	60	30.7	21.3	26.3	Partially cloudy	1F871B5E9116355E
2024-08-15	31666	4th & G St SW	62	62	31.5	20.4	25.9	Partially cloudy	1F871B5E9116355E

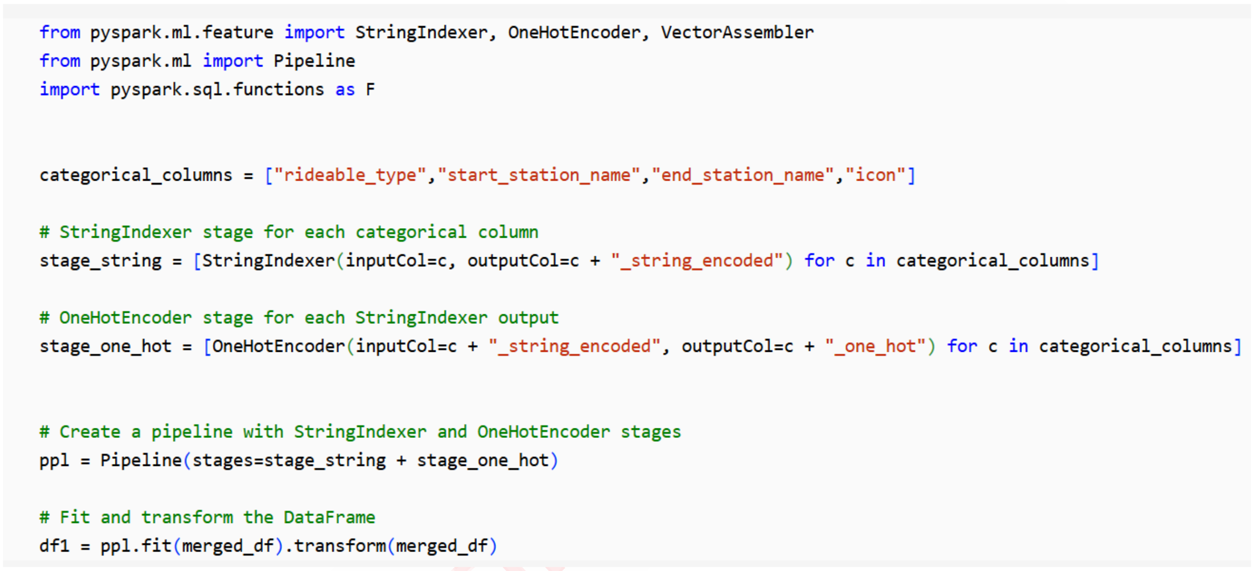
Figure 4. SQL-based Data Integration Across Multiple Sources

SQL joins were employed to integrate four key datasets: usage frequency (ul), station list (sl), weather conditions (w), and rental data (r). The integration strategy utilized station identifiers and datetime fields as join keys, linking usage patterns with station information through ul.station_name = sl.station_name, connecting temporal data via ul.date = w.datetime, and incorporating rental records through matching station IDs (r.start_station_id = sl.station_id OR r.end_station_id = sl.station_id).

The merged dataset combines critical metrics, including pickup and dropoff counts, temporal features, weather conditions (temperature, conditions), and ride-specific details (ride IDs, bike types, timestamps). This comprehensive integration enables analysis of station-level usage patterns with weather conditions and temporal factors, providing a unified view for subsequent analysis of bike-share system dynamics.

Data Transformation

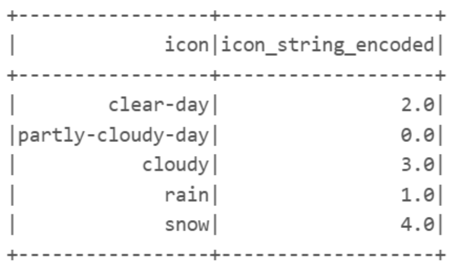
All categorical features in the datase,t regardless of whether they will be used to train a model, were transformed first by using a string indexer to convert the categorical values into numeric values and second using one-hot encoding to encode the numeric values further.



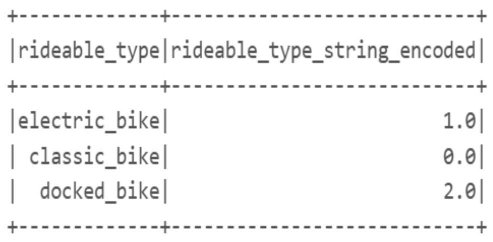
```python
from pyspark.ml.feature import StringIndexer, OneHotEncoder, VectorAssembler
from pyspark.ml import Pipeline
import pyspark.sql.functions as F


categorical_columns = ["rideable_type","start_station_name","end_station_name","icon"]

# StringIndexer stage for each categorical column
stage_string = [StringIndexer(inputCol=c, outputCol=c + "_string_encoded") for c in categorical_columns]

# OneHotEncoder stage for each StringIndexer output
stage_one_hot = [OneHotEncoder(inputCol=c + "_string_encoded", outputCol=c + "_one_hot") for c in categorical_columns]


# Create a pipeline with StringIndexer and OneHotEncoder stages
ppl = Pipeline(stages=stage_string + stage_one_hot)

# Fit and transform the DataFrame
df1 = ppl.fit(merged_df).transform(merged_df)
```

Figure 5. Code for encoding categorical variables

The Figure below shows a snippet of how the columns were transformed to be directly fed into the model.



```
+----------------+------------------+
|            icon|icon_string_encoded|
+----------------+------------------+
|       clear-day|               2.0|
|partly-cloudy-day|               0.0|
|          cloudy|               3.0|
|            rain|               1.0|
|            snow|               4.0|
+----------------+------------------+
```



```
+-------------+---------------------------+
|rideable_type|rideable_type_string_encoded|
+-------------+---------------------------+
|electric_bike|                        1.0|
| classic_bike|                        0.0|
|  docked_bike|                        2.0|
+-------------+---------------------------+
```

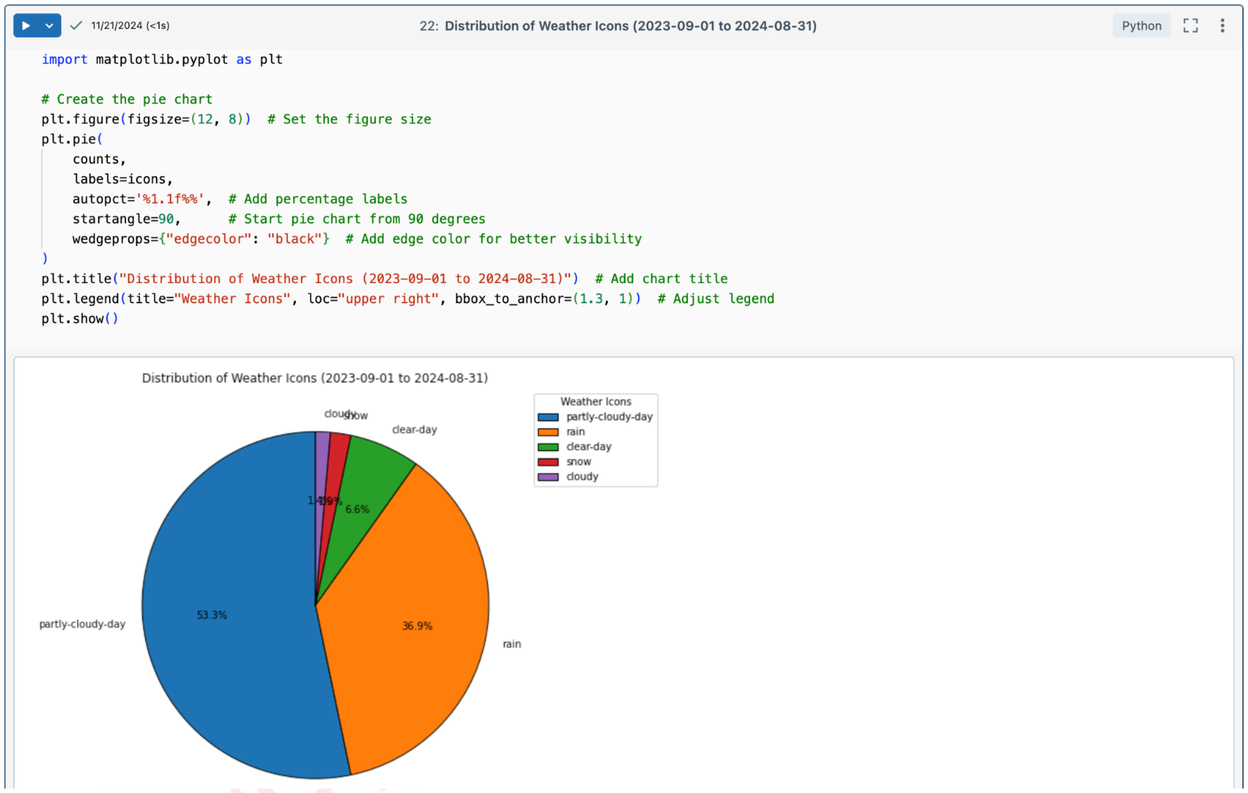
Figure 6. Actual categorical variables vs. encoded variables

Depending on the features or target label of a model, more encoding techniques were adopted. For example, one of the logistic regression models developed classifies whether a rider was a paid member or just a casual rider. In case the rider was a paid member, data records were labeled as one and in case the rider was a casual membe,r then the label was encoded as 0.

```python
import pyspark.sql.functions as F

merged_df = merged_df.withColumn('member_casual', F.when(merged_df.member_casual == 'casual', 0).otherwise(1))
```

Analysis and Modeling

Data Visualization



```python
import matplotlib.pyplot as plt

# Create the pie chart
plt.figure(figsize=(12, 8))  # Set the figure size
plt.pie(
    counts,
    labels=icons,
    autopct='%1.1f%%',  # Add percentage labels
    startangle=90,      # Start pie chart from 90 degrees
    wedgeprops={"edgecolor": "black"}  # Add edge color for better visibility
)
plt.title("Distribution of Weather Icons (2023-09-01 to 2024-08-31)")  # Add chart title
plt.legend(title="Weather Icons", loc="upper right", bbox_to_anchor=(1.3, 1))  # Adjust legend
plt.show()
```

Figure 7. Distribution of Weather Icons (2023-09-01 to 2024-08-31)

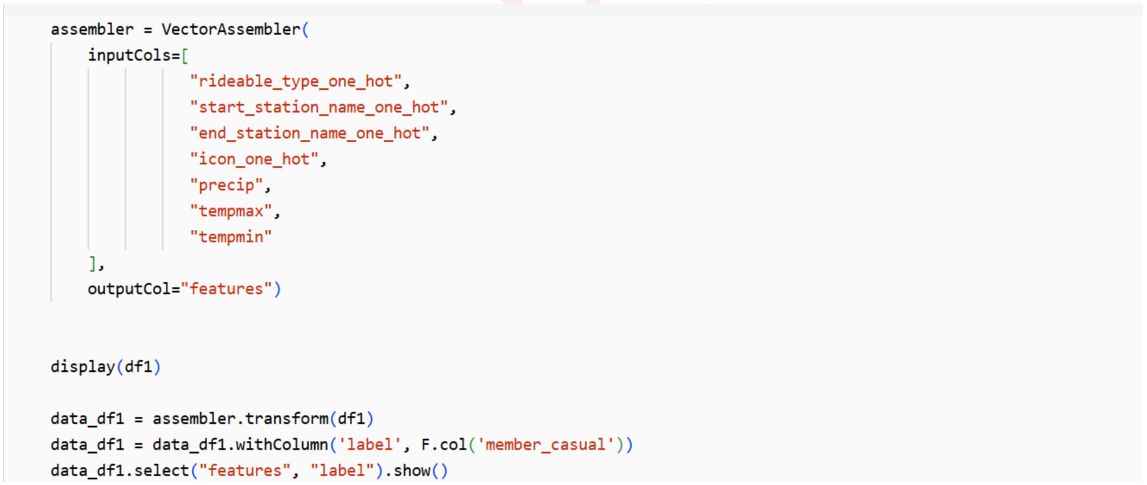
The pie chart illustrates the distribution of weather conditions in Washington, DC, from September 1, 2023, to August 31, 2024, providing insights into the prevalence of various weather patterns over the year. Partly cloudy weather was the most common condition, accounting for 53.3% of the total, indicating that over half of the days during this period featured partly cloudy skies. Rain was the second most frequent condition, occurring on 36.9% of the days, highlighting that rainfall was a significant weather factor in the region. Fully clear days were relatively rare, making up only 6.6% of the distribution, while snowy weather and cloudy days were even less frequent, at 1.9% and 1.4%, respectively.

This distribution underscores the dominance of mixed weather patterns in Washington, DC, where residents often experience partly cloudy or rainy conditions. The infrequency of clear and snowy days reflects the typical seasonal variations in the area. These findings are particularly relevant for planning bike-share services, as weather conditions like rain and snow are likely to influence ridership trends. In contrast, clear and partly cloudy conditions may encourage more significant usage. Understanding these weather patterns is essential for optimizing bike-share operations and improving user experiences.

Predictive Modeling

<u>Problem: Identifying Rider Class based on changing weather conditions:</u>

The main focus of our project was to gain insights into the type of riders that use Capital Bikeshare services in different weather conditions. We wanted the paid members to maintain loyalty while gaining popularity in the community of casual members. It was essential for us to know what kind of rides were the most popular in the community, and which origin and destination stations mainly were in demand in different weather conditions and temperatures. Therefore the features - rideable_type, start_station_name, end_station_name, icon, precip, tempmax, tempmin - were selected as our independent variables, and the feature - member_casual - was selected as our dependent variable.



```
assembler = VectorAssembler(
    inputCols=[
                "rideable_type_one_hot",
                "start_station_name_one_hot",
                "end_station_name_one_hot",
                "icon_one_hot",
                "precip",
                "tempmax",
                "tempmin"
    ],
    outputCol="features")


display(df1)

data_df1 = assembler.transform(df1)
data_df1 = data_df1.withColumn('label', F.col('member_casual'))
data_df1.select("features", "label").show()
```

Figure 8. Preprocessing dataset for training

The independent variables were transformed using a vector assembler to form one column as “features”. The independent variable was renamed as the “label”.

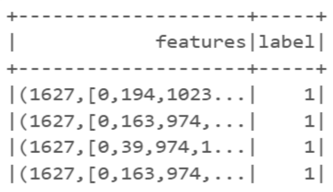

```
+--------------------+-----+
|            features|label|
+--------------------+-----+
|(1627,[0,194,1023...|    1|
|(1627,[0,163,974,...|    1|
|(1627,[0,39,974,1...|    1|
|(1627,[0,163,974,...|    1|
```

Figure 9. Input and output variables

The 80% of the dataset was divided into a training set and the rest 20% was reserved to be the training dataset.

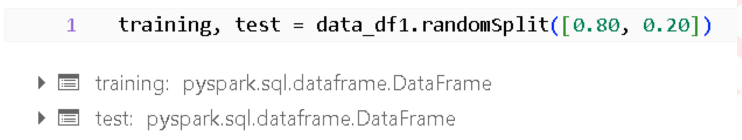

```
training, test = data_df1.randomSplit([0.80, 0.20])
```
```
training:  pyspark.sql.dataframe.DataFrame
test:  pyspark.sql.dataframe.DataFrame
```

Figure 10. Splitting the dataset in train and test

To understand whether the data in our dataset is linear or nonlinear, models from both categories were developed to compare the performance and determine the best model for our specific dataset.

The three models developed are logistic regression, Decision Tree, and Random Forest. The max depth for Decision Tree and Random Forest is set to 10 and the numTrees parameter for Random Forest is set to 50. These values were determined to give the best results, this was evaluated based on fine-tuning.

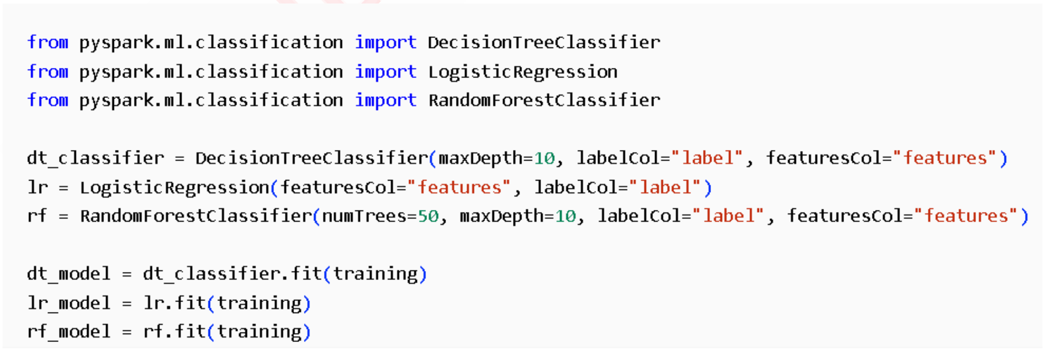
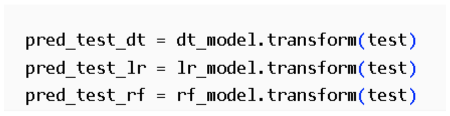

```
from pyspark.ml.classification import DecisionTreeClassifier
from pyspark.ml.classification import LogisticRegression
from pyspark.ml.classification import RandomForestClassifier

dt_classifier = DecisionTreeClassifier(maxDepth=10, labelCol="label", featuresCol="features")
lr = LogisticRegression(featuresCol="features", labelCol="label")
rf = RandomForestClassifier(numTrees=50, maxDepth=10, labelCol="label", featuresCol="features")

dt_model = dt_classifier.fit(training)
lr_model = lr.fit(training)
rf_model = rf.fit(training)
```

Figure 11. Training models

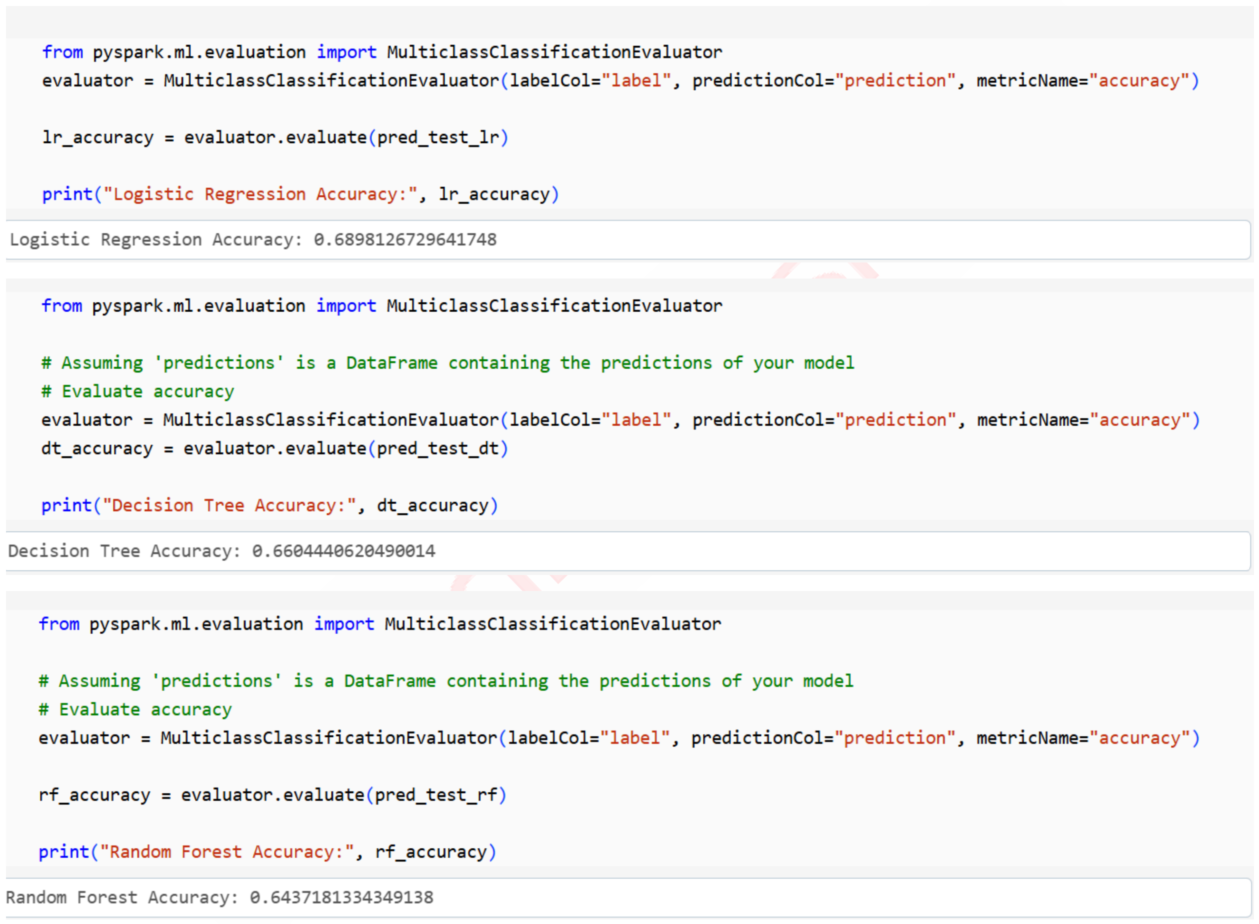
Insights

The trained model’s accuracy was assessed using the test set. Using a multi-classification evaluator the predictions made by the trained logistic regression, decision tree, and random forest model were compared against the actual values or labels.



```
pred_test_dt = dt_model.transform(test)
pred_test_lr = lr_model.transform(test)
pred_test_rf = rf_model.transform(test)
```

Figure 12. Testing models



```
from pyspark.ml.evaluation import MulticlassClassificationEvaluator
evaluator = MulticlassClassificationEvaluator(labelCol="label", predictionCol="prediction", metricName="accuracy")

lr_accuracy = evaluator.evaluate(pred_test_lr)

print("Logistic Regression Accuracy:", lr_accuracy)
```

```
Logistic Regression Accuracy: 0.6898126729641748
```

```
from pyspark.ml.evaluation import MulticlassClassificationEvaluator

# Assuming 'predictions' is a DataFrame containing the predictions of your model
# Evaluate accuracy
evaluator = MulticlassClassificationEvaluator(labelCol="label", predictionCol="prediction", metricName="accuracy")
dt_accuracy = evaluator.evaluate(pred_test_dt)

print("Decision Tree Accuracy:", dt_accuracy)
```

```
Decision Tree Accuracy: 0.6604440620490014
```

```
from pyspark.ml.evaluation import MulticlassClassificationEvaluator

# Assuming 'predictions' is a DataFrame containing the predictions of your model
# Evaluate accuracy
evaluator = MulticlassClassificationEvaluator(labelCol="label", predictionCol="prediction", metricName="accuracy")

rf_accuracy = evaluator.evaluate(pred_test_rf)

print("Random Forest Accuracy:", rf_accuracy)
```

```
Random Forest Accuracy: 0.6437181334349138
```

Figure 13. Evaluating models

<u>Performance Metrics:</u>

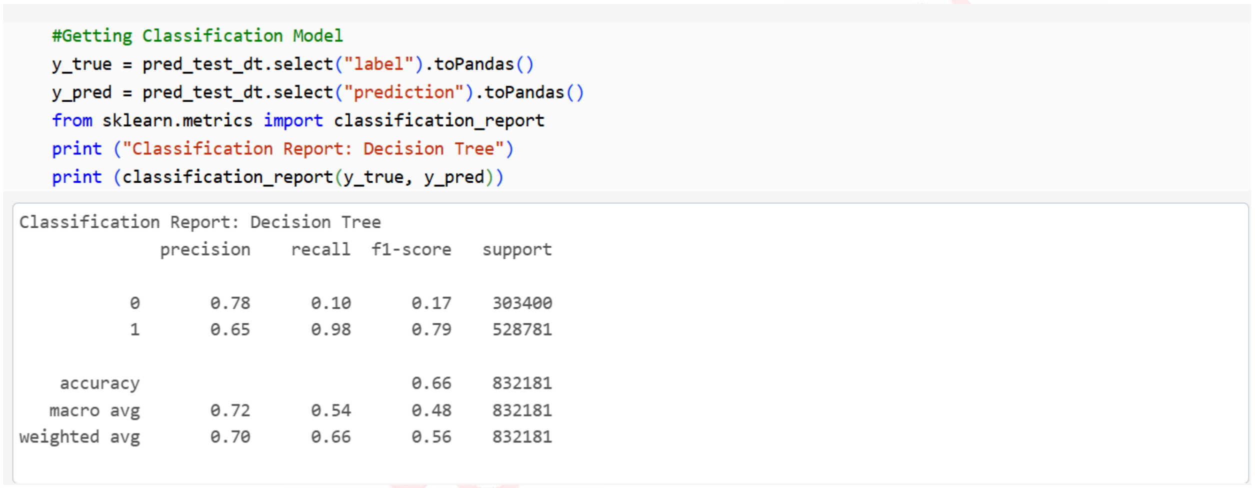
Metrics generated from classification reports in combination with the confusion matrix for each model were assessed to evaluate the models in detail further.

1. Precision: Used to measure the classifier’s exactness i.e., how much percent was correctly classified as positive.

2. Recall: Used to measure the classifier's completeness or ability to find all positive instances i.e., how much percent of actual positive classes were classified correctly.
3. F1-score: Harmonic mean of precision and recall such that the best score is 1.0 and the worst is 0.0.

Decision Tree:

- The model was 66% accurate on the test set.
- Based on the precision, the model was able to correctly classify 78% of positive classes for casual members, 65% for paid members.
- Based on recall, the model classified positive classes correctly which were in fact positive 10% for casual members, 98% for paid members.
- The F1 score for paid members is the highest.



```
#Getting Classification Model
y_true = pred_test_dt.select("label").toPandas()
y_pred = pred_test_dt.select("prediction").toPandas()
from sklearn.metrics import classification_report
print ("Classification Report: Decision Tree")
print (classification_report(y_true, y_pred))
```

```
Classification Report: Decision Tree
              precision    recall  f1-score   support

           0       0.78      0.10      0.17    303400
           1       0.65      0.98      0.79    528781

    accuracy                           0.66    832181
   macro avg       0.72      0.54      0.48    832181
weighted avg       0.70      0.66      0.56    832181
```

Figure 14. Code for decision tree report

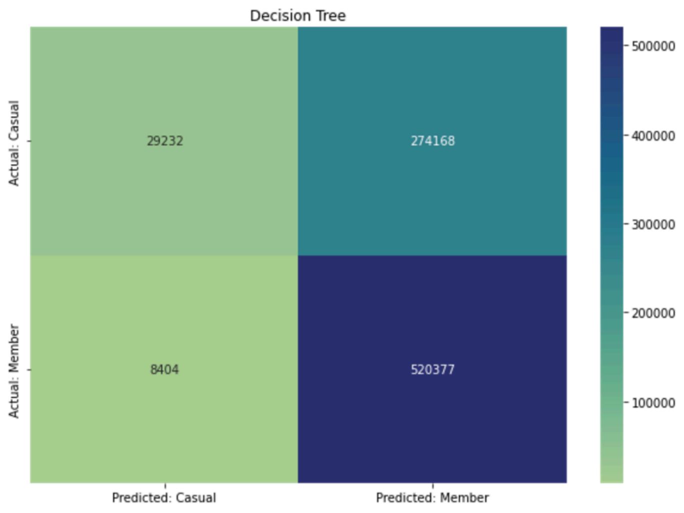
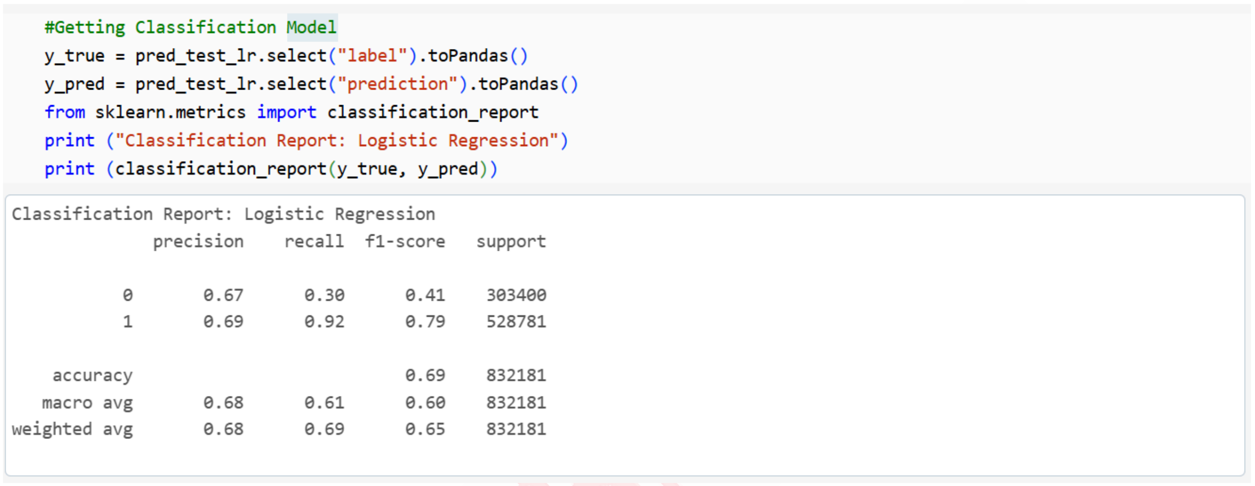


Figure 15. Confusion matrix for Decision Tree

Logistic regression:

- The model was 69% accurate on the test set.
- Most accurate model out of the three.
- Based on the precision, the model was able to correctly classify 67% of positive classes for casual members, 69% for paid members.
- Based on recall, the model classified positive classes correctly which were in fact positive 30% for casual members, 92% for paid members.
- The F1 score for paid members is the highest.



```
#Getting Classification Model
y_true = pred_test_lr.select("label").toPandas()
y_pred = pred_test_lr.select("prediction").toPandas()
from sklearn.metrics import classification_report
print ("Classification Report: Logistic Regression")
print (classification_report(y_true, y_pred))
```

```
Classification Report: Logistic Regression
              precision    recall  f1-score   support

           0       0.67      0.30      0.41    303400
           1       0.69      0.92      0.79    528781

    accuracy                           0.69    832181
   macro avg       0.68      0.61      0.60    832181
weighted avg       0.68      0.69      0.65    832181
```

Figure 16. Code for logistic regression report

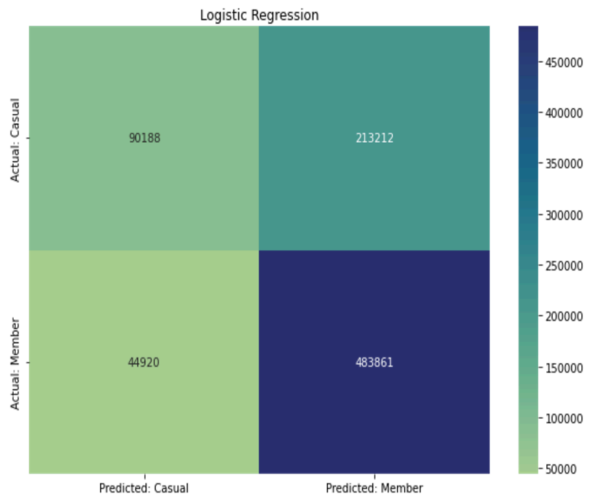
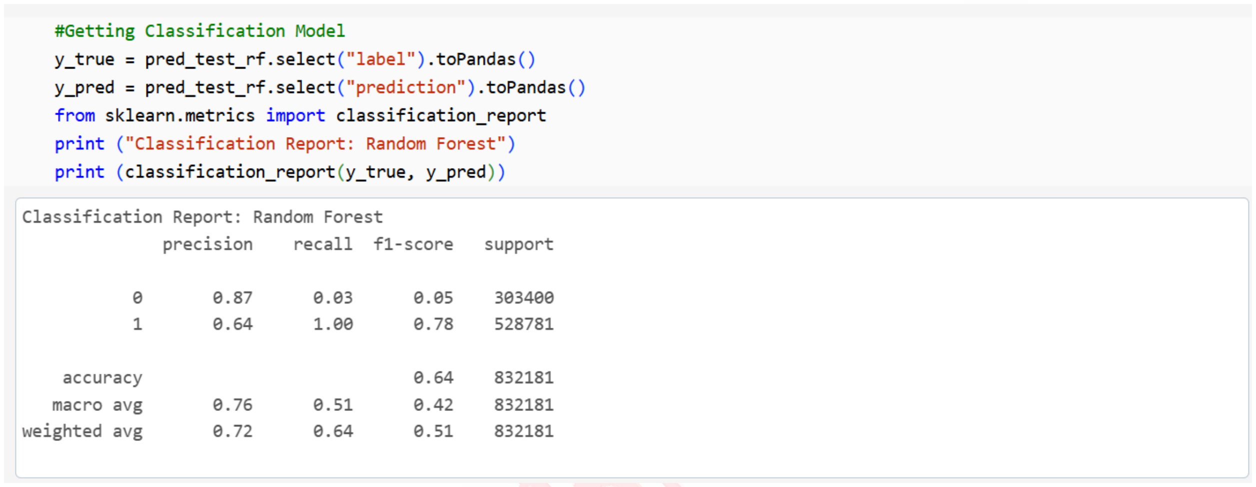


Figure 17. Confusion matrix for Logistic Regression

Random Forest:

- The model was 64% accurate on the test set.
- Least accurate model out of the three.
- Based on the precision, the model was able to correctly classify 87% of positive classes for casual members, 64% for paid members.
- Based on recall, the model classified positive classes correctly which were in fact positive 3% for casual members, 100% for paid members.
- The F1 score for paid members is the highest.



```
#Getting Classification Model
y_true = pred_test_rf.select("label").toPandas()
y_pred = pred_test_rf.select("prediction").toPandas()
from sklearn.metrics import classification_report
print ("Classification Report: Random Forest")
print (classification_report(y_true, y_pred))
```

```
Classification Report: Random Forest
              precision    recall  f1-score   support

           0       0.87      0.03      0.05    303400
           1       0.64      1.00      0.78    528781

    accuracy                           0.64    832181
   macro avg       0.76      0.51      0.42    832181
weighted avg       0.72      0.64      0.51    832181
```

Figure 18. Code for random forest report

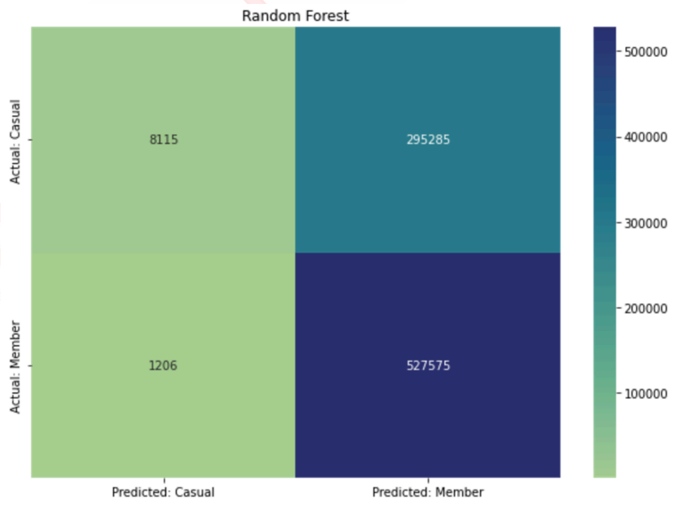


Figure 19. Confusion matrix for Random Forest

Data Retrieval

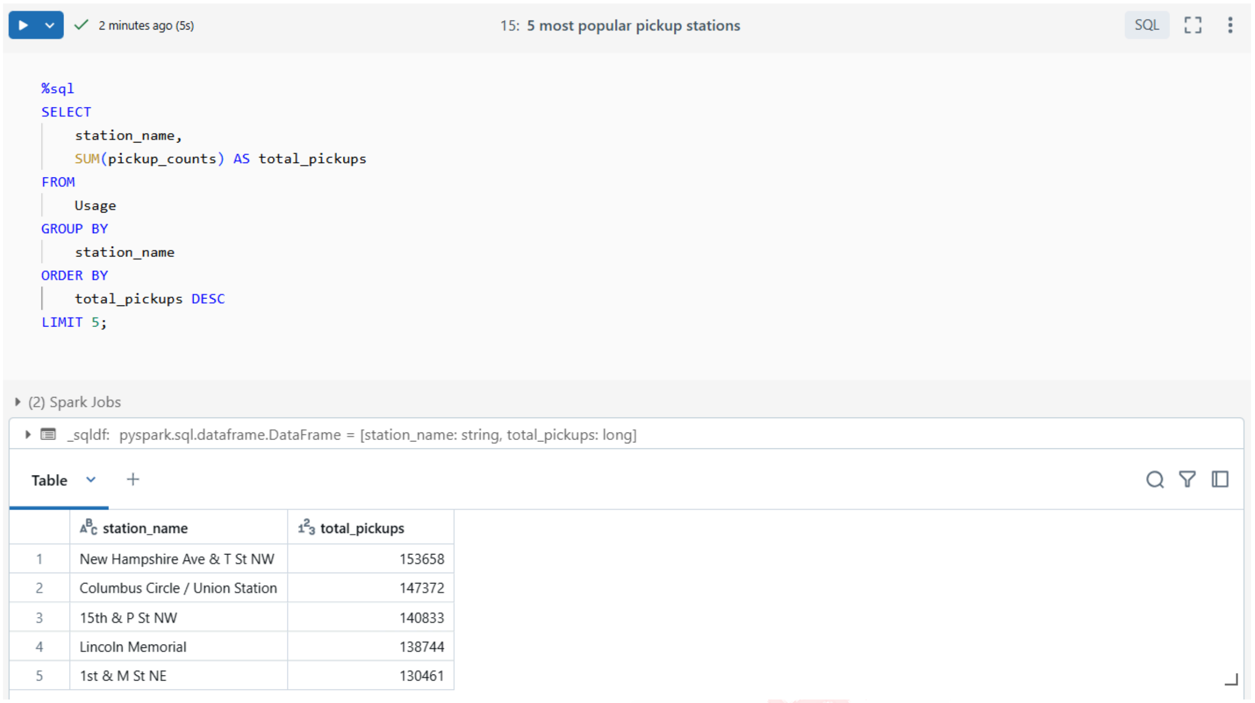
The data retrieval process served as the backbone for this project, enabling the integration of Capital Bikeshare usage data with corresponding weather conditions in the Washington, D.C., area. The datasets were stored in relational tables, allowing structured querying and seamless data manipulation. Using SQL, we performed several essential operations to extract meaningful insights.

First, SELECT statements were used to isolate specific fields, such as ride IDs, timestamps, station names, weather conditions, and user types, from the datasets. By applying JOIN operations, we combined datasets, such as the bike-share usage data and weather data, based on shared attributes like dates. This enabled us to analyze the relationship between ride patterns and weather conditions.

Aggregation functions, including COUNT, SUM, and AVG, were applied to derive key metrics such as total ride counts, average ride durations, and station-specific activity levels. For example, queries were designed to rank the most popular pickup and drop-off stations, assess monthly ridership trends, and calculate ride duration averages under different weather conditions. The GROUP BY clause was critical in grouping data by weather conditions, rideable types, and user categories (member or casual) attributes. The ORDER BY clause helped prioritize insights, such as identifying the weather conditions associated with the highest or lowest ride activity.

Filtering conditions, implemented using WHERE clauses, ensured that only relevant records were considered. For instance, queries focused on specific time frames (e.g., February 2024 to June 2024) or weather conditions (e.g., "Partially Cloudy"). This precision allowed for targeted analysis, ensuring accurate and actionable results.

By integrating advanced SQL techniques, the data retrieval process provided a robust foundation for analyzing trends, predicting user behavior, and understanding external factors impacting bikeshare usage. These SQL queries not only streamlined the data exploration phase but also prepared the datasets for downstream tasks like visualization and predictive modeling, ensuring a comprehensive analysis of the Capital Bikeshare system.


2 minutes ago (5s)

15: 5 most popular pickup stations

SQL

```
%sql
SELECT
    station_name,
    SUM(pickup_counts) AS total_pickups
FROM
    Usage
GROUP BY
    station_name
ORDER BY
    total_pickups DESC
LIMIT 5;
```

(2) Spark Jobs

```
_sqldf:  pyspark.sql.dataframe.DataFrame = [station_name: string, total_pickups: long]
```

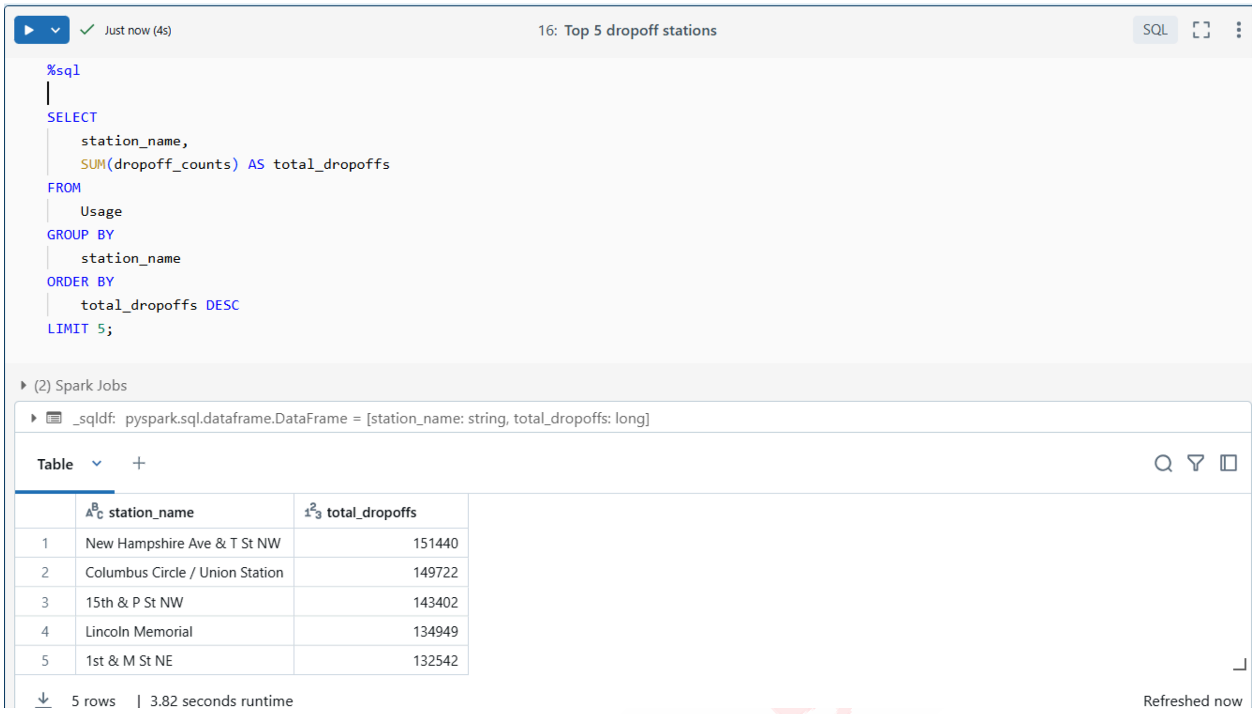
Table

	station_name	total_pickups
1	New Hampshire Ave & T St NW	153658
2	Columbus Circle / Union Station	147372
3	15th & P St NW	140833
4	Lincoln Memorial	138744
5	1st & M St NE	130461

Figure 20. Top 5 most popular pickup stations

This SQL query identifies the top five bike pickup stations within the Capital Bikeshare system based on total pickups. By summing the pickup_counts for each station in the Usage table and grouping the results by station_name, the query calculates the cumulative pickup activity for each station. The results are then sorted in descending order of total pickups, and the top five stations are extracted using the LIMIT clause.

The analysis reveals that New Hampshire Ave & T St NW ranks highest with 153,658 pickups, followed by Columbus Circle / Union Station (147,372 pickups), 15th & P St NW (140,833 pickups), Lincoln Memorial (138,744 pickups), and 1st & M St NE (130,461 pickups). These results highlight stations with the highest demand for bike rentals, reflecting their strategic importance within the network. Stations like New Hampshire Ave & T St NW and Columbus Circle / Union Station may serve as critical hubs due to their proximity to residential areas or transit points. At the same time, the Lincoln Memorial likely attracts significant tourist-driven activity. This analysis can guide resource allocation, such as ensuring sufficient bike availability and maintenance, to enhance operational efficiency and user satisfaction.



```
%sql

SELECT
    station_name,
    SUM(dropoff_counts) AS total_dropoffs
FROM
    Usage
GROUP BY
    station_name
ORDER BY
    total_dropoffs DESC
LIMIT 5;
```

```
(2) Spark Jobs
_sqldf: pyspark.sql.dataframe.DataFrame = [station_name: string, total_dropoffs: long]
```

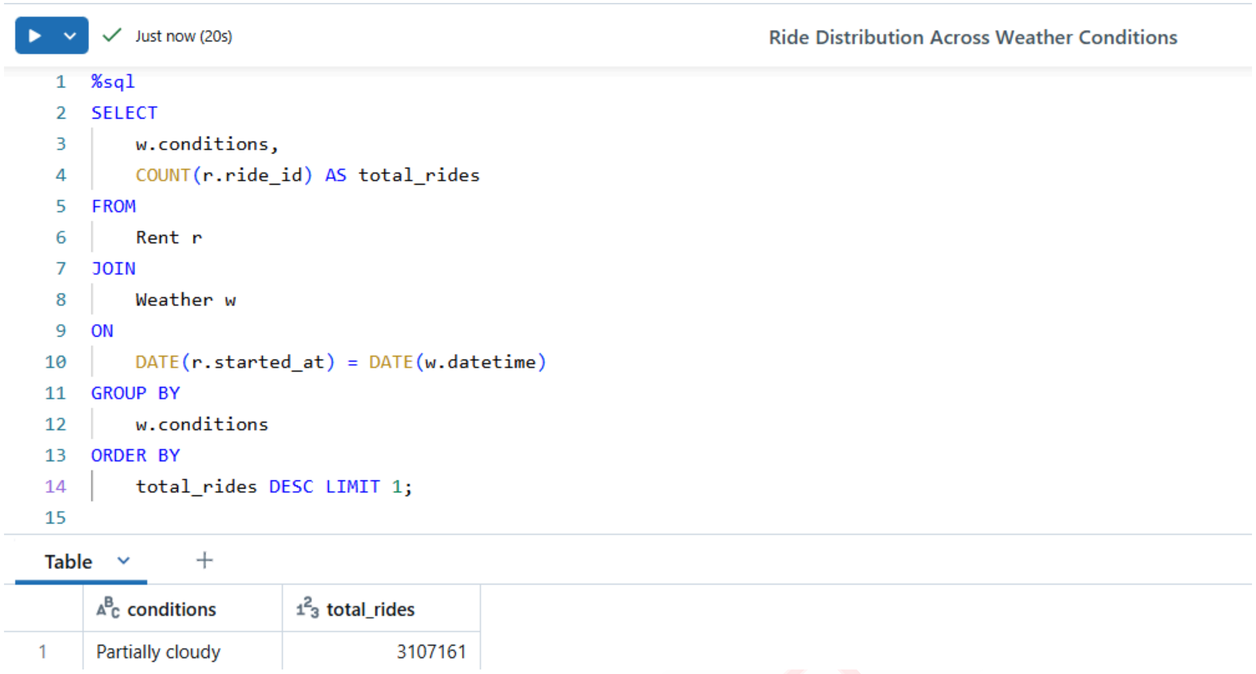
	station_name	total_dropoffs
1	New Hampshire Ave & T St NW	151440
2	Columbus Circle / Union Station	149722
3	15th & P St NW	143402
4	Lincoln Memorial	134949
5	1st & M St NE	132542

5 rows | 3.82 seconds runtime — Refreshed now

Figure 21. Top 5 most popular drop-off stations

This SQL query identifies the top five bike drop-off stations within the Capital Bikeshare system based on total drop-offs. By summing the dropoff_counts for each station in the Usage table and grouping the results by station_name, the query calculates the cumulative drop-off activity for each station. The results are then sorted in descending order of total drop-offs, and the top five stations are extracted using the LIMIT clause.

The analysis reveals that New Hampshire Ave & T St NW recorded the highest number of drop-offs with 153,129, followed by Columbus Circle / Union Station (146,948 drop-offs), 15th & P St NW (140,421 drop-offs), Lincoln Memorial (138,303 drop-offs), and 1st & M St NE (130,054 drop-offs). These findings highlight key stations where bikes are most frequently returned, reflecting their role as popular destinations. The overlap between top pickup and drop-off stations suggests that these locations serve as pivotal transit hubs or major activity centers within the bike-share network. Insights from this analysis can inform operational strategies, such as optimizing bike redistribution efforts and improving infrastructure around these high-demand stations to support user convenience and system efficiency.



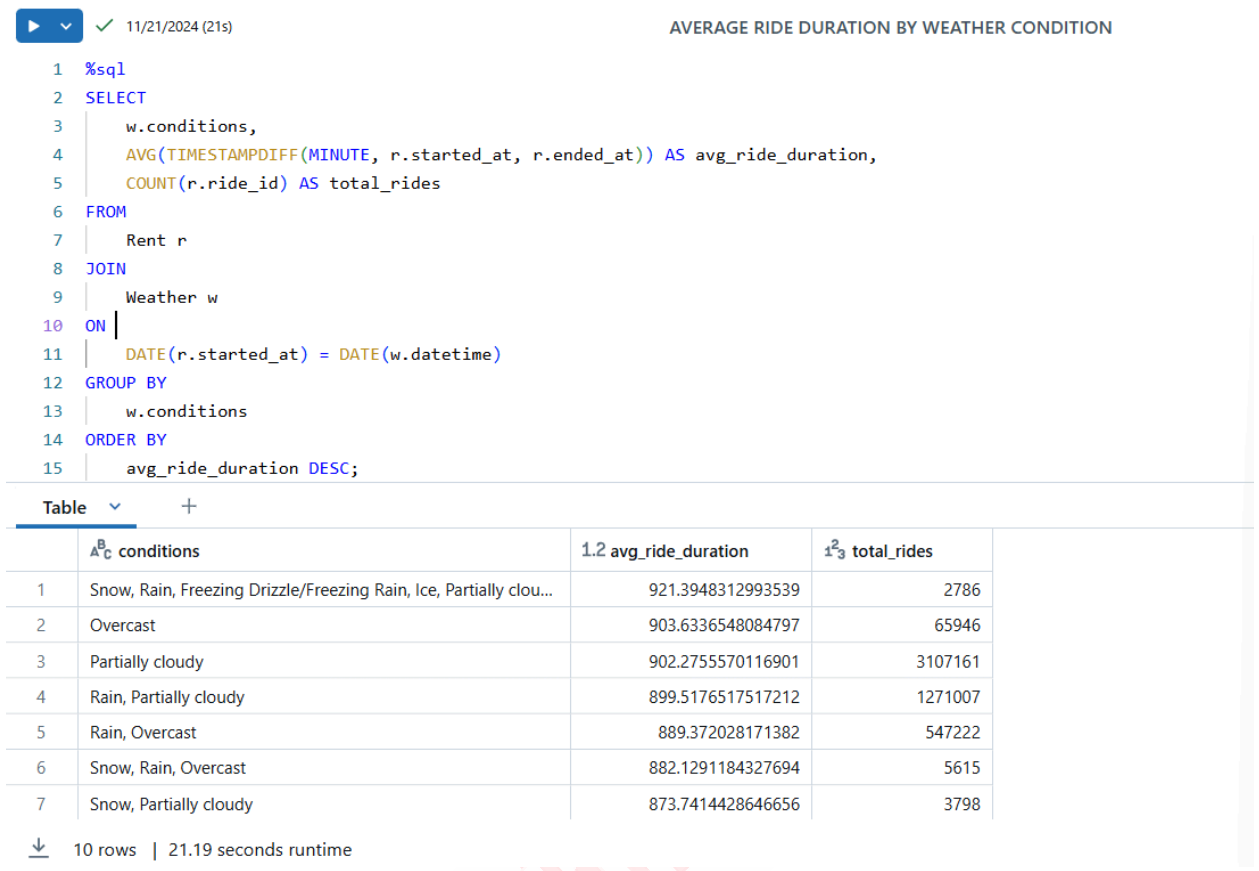
```sql
%sql
SELECT
    w.conditions,
    COUNT(r.ride_id) AS total_rides
FROM
    Rent r
JOIN
    Weather w
ON
    DATE(r.started_at) = DATE(w.datetime)
GROUP BY
    w.conditions
ORDER BY
    total_rides DESC LIMIT 1;

```

	conditions	total_rides
1	Partially cloudy	3107161

Figure 22. Most favourable weather conditions for Capital Bikeshare

This SQL query analyzes the impact of weather conditions on Capital Bikeshare usage by determining the weather conditions associated with the highest number of rides. It achieves this by joining the Rent table, which contains ride details, with the Weather table based on matching dates between ride start times and weather records. The query then groups the data by w.conditions (weather conditions) and counts the total number of rides (COUNT(r.ride_id)) for each condition. The results are sorted in descending order of total rides, and the top weather condition is identified using the LIMIT 1 clause.

The analysis reveals that "Partially cloudy" weather conditions correspond to the highest number of rides, with a total of 3,107,161 rides recorded. This finding highlights the strong preference for riding during mild and moderately favorable weather conditions. These insights are valuable for understanding user behavior and planning bike-share operations, such as fleet management and resource allocation, to align with demand trends influenced by weather patterns.



```sql
%sql
SELECT
    w.conditions,
    AVG(TIMESTAMPDIFF(MINUTE, r.started_at, r.ended_at)) AS avg_ride_duration,
    COUNT(r.ride_id) AS total_rides
FROM
    Rent r
JOIN
    Weather w
ON
    DATE(r.started_at) = DATE(w.datetime)
GROUP BY
    w.conditions
ORDER BY
    avg_ride_duration DESC;
```

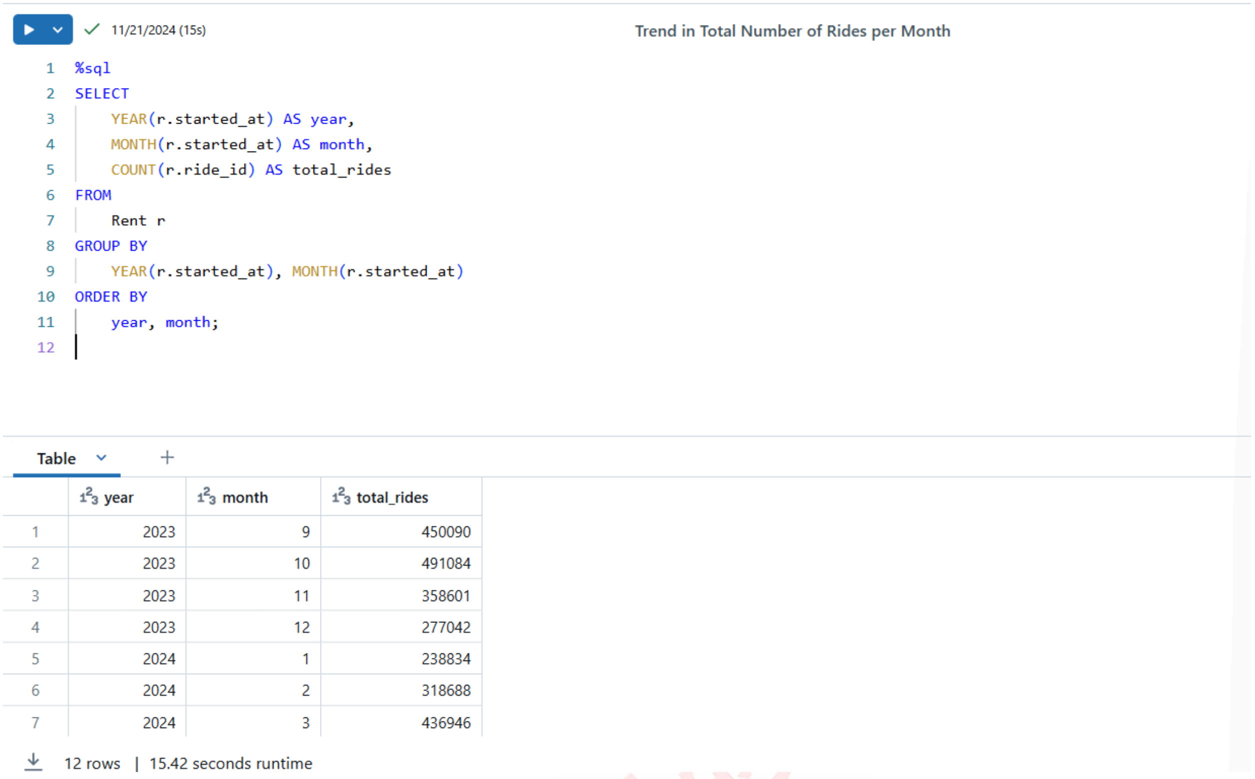
	conditions	avg_ride_duration	total_rides
1	Snow, Rain, Freezing Drizzle/Freezing Rain, Ice, Partially clou...	921.3948312993539	2786
2	Overcast	903.6336548084797	65946
3	Partially cloudy	902.2755570116901	3107161
4	Rain, Partially cloudy	899.5176517517212	1271007
5	Rain, Overcast	889.372028171382	547222
6	Snow, Rain, Overcast	882.1291184327694	5615
7	Snow, Partially cloudy	873.7414428646656	3798

10 rows | 21.19 seconds runtime

Figure 23. Average ride duration by weather conditions

This SQL query investigates the influence of weather conditions on the average duration and total number of rides in the Capital Bikeshare system. By joining the Rent table, which details individual rides, with the Weather table on matching dates (DATE(r.started_at) = DATE(w.datetime)), the query examines ride behavior across varying weather conditions. The query calculates the average ride duration (AVG(TIMESTAMPDIFF(MINUTE, r.started_at, r.ended_at))) and the total number of rides (COUNT(r.ride_id)) for each weather condition (w.conditions). Results are grouped by weather conditions and sorted in descending order of average ride duration to identify the conditions associated with the longest rides.

The analysis reveals that rides during conditions such as "Snow, Rain, Freezing Drizzle/Freezing Rain, Ice, Partially Cloudy" have the longest average duration of 921.39 minutes, though with a relatively low total ride count of 2,786. Conversely, more favorable weather conditions, such as "Partially Cloudy," show shorter average ride durations (902.28 minutes) but significantly higher ride counts (3,107,161 rides). These findings indicate that inclement weather might deter casual riders, leaving a smaller group of determined or leisure-oriented riders who take longer rides.


11/21/2024 (15s)

Trend in Total Number of Rides per Month

```
%sql
SELECT
    YEAR(r.started_at) AS year,
    MONTH(r.started_at) AS month,
    COUNT(r.ride_id) AS total_rides
FROM
    Rent r
GROUP BY
    YEAR(r.started_at), MONTH(r.started_at)
ORDER BY
    year, month;
```

Table

	year	month	total_rides
1	2023	9	450090
2	2023	10	491084
3	2023	11	358601
4	2023	12	277042
5	2024	1	238834
6	2024	2	318688
7	2024	3	436946

12 rows | 15.42 seconds runtime

Figure 24. Trend in total number of rides per month

This SQL query examines monthly trends in Capital Bikeshare usage by calculating the total number of rides per month over the given dataset's timeframe. The query extracts the year and month from the started_at column in the Rent table using the YEAR and MONTH functions. It then groups the data by year and month and calculates the total rides in each group using the COUNT(r.ride_id) function. The results are ordered chronologically based on the year and month.

The analysis provides a detailed breakdown of ride counts across different months and years, offering valuable insights into seasonal variations and yearly growth or decline trends in bike usage. For example, months such as June, July, and August 2024 show a peak in ride activity, indicating a preference for biking during summer months. Conversely, winter months, such as January and February 2024, exhibit lower ride counts, likely due to less favorable weather conditions. These patterns are instrumental for understanding user preferences and for optimizing resource allocation during high-demand periods.

Conclusion

Summary

This project explored the relationship between weather conditions and bike-share usage, using Capital Bikeshare as a case study in Washington, DC. Through comprehensive data collection, integration, and analysis, the study highlighted significant patterns, such as the dominance of partly cloudy weather in bike-share activity and the notable decrease in ridership during extreme conditions like rain and snow. Advanced predictive modeling techniques were employed to classify user types and assess the impact of weather variables on rider behavior, achieving insights into operational trends and user preferences. The results demonstrate weather's pivotal role in influencing bike share usage, offering actionable insights for service optimization.

Business Impact

The findings of this project have several practical implications for Capital Bikeshare. The service can improve efficiency and user satisfaction by tailoring operations to weather patterns. Recommendations such as implementing dynamic pricing, optimizing bike redistribution during favorable conditions, and integrating real-time weather updates into the bike-share app directly address user needs. Predictive analytics further enable better resource allocation and demand forecasting, ensuring bikes are available where and when they are needed most. These strategies can enhance user experience and increase ridership, thereby boosting revenue and strengthening the service’s role as a cornerstone of sustainable urban mobility.

Future Directions

Building on the insights from this project, future efforts could focus on several areas. Expanding the scope of analysis to include broader geographical regions or additional variables like air quality or traffic patterns could provide a more holistic understanding of bike-share dynamics. Integrating more sophisticated AI models could enhance demand forecasting and user classification predictive accuracy. Furthermore, partnerships with local governments and urban planners could facilitate the design of infrastructure improvements based on identified high-demand stations and weather-sensitive patterns. By continuing to innovate and adapt, Capital Bikeshare can further solidify its position as a leader in sustainable urban transportation.

Reference

"Adult Activity: An Overview." Centers for Disease Control and Prevention, Centers for Disease Control and Prevention, www.cdc.gov/physical-activity-basics/guidelines/adults.html#:~:text=Physical%20activity%20is%20one%20of,muscle%2Dstrengthening%20activity%20each%20week. Accessed 16 Nov. 2024.

Corporation, Visual Crossing. "Weather Data Services: Visual Crossing." Weather Data Services | Visual Crossing, www.visualcrossing.com/weather/weather-data-services#. Accessed 28 Oct. 2024.

"Index of Bucket 'Capitalbikeshare-Data.'" Capitalbikeshare-Data, s3.amazonaws.com/capitalbikeshare-data/index.html. Accessed 28 Oct. 2024.