CSCI-E88C Final Project

Halvorsen, Hannah

[**[50 points] Project Proposal and Summary**](#_4uofvmp3vdtn)2

[Project Goal and Problem Statement](#_m5qaejocb0g7) 2

[Data Sources](#_95rej4lhgw5n) 2

[Expected Results](#_yu7f29rhf1de) 2

[Application Overview and Technologies used](#_guk5lyhz1hth) 2

[Bonus Options:](#_ex6hr5lg6mak) 3

[**[50 points] Milestone 1: System Design and Architecture**](#_ylsvsg71nax9)3

[System Diagram](#_3ckbpnlmomiv) 3

[Description of each stage of the processing pipeline](#_czam0suf3ewl) 3

[**[100 points] Milestone 2: Implementation and Results**](#_ed6l2j8d3wl8)4

[Code and configuration snippets](#_eai23ihah8bd) 4

[Screenshots of run commands and processing results](#_t9ff1nvln4o0) 5

[**[50 points] Results, analysis, conclusions, and challenges**](#_buiiravsl63l)9

[Summary of results](#_la4ujpgvcxob) 9

[Potential options for scaling to a distributed system](#_1syp5j2s635d) 10

[Application of Functional Programming Principles](#_qgad1cijp2jn) 11

[Challenges with project and what to do differently next time](#_5twt56bzp1q2) 11

[Youtube video URL](#_iu3rngrzf1ld) 12

[**[50 points] Bonus**](#_xzsymsmtfi6s)12

[Visualization](#_dsj0kb7wieim) 12

**1. Project Proposal and Summary**  
a. [Project Goal and Problem Statement](#_m5qaejocb0g7)  
**Goal:**  
Analyze Bluebikes trip data to uncover patterns in rider behavior, popular stations, and system growth. Develop a pipeline using Scala and Spark to process historical bikeshare trip data and station information. Implement predictive analytics (linear regression) for forecasting demand. Visualize key trends and distributions in a manner that supports urban mobility decision-making.

**Problem Statement:**  
With a large volume of trip data produced annually, there is a need for an efficient system to analyze and visualize historical bikeshare usage. Understanding when, where, and how people use bikeshare can help in station planning, inventory management, and improving user experience.

b. Data Sources

* **Bluebikes Trip Histories:** CSV files containing trip start/end times, durations, and station info (downloaded from S3).

(an example has been saved in the Input\_and\_Result\_data.zip named 202411-bluebikes-tripdata.csv)

* **Bluebikes Station Data (Static):** Station names, IDs, coordinates, and capacities.

(this data file has been saved in the Input\_and\_Result\_data.zip named current\_bluebikes\_stations.csv)

c. Expected Results

* Monthly and quarterly aggregated metrics (total trips, average trip duration).
* Month-over-month growth rates.
* Station-level insights (capacity, geographic distribution, usage patterns).
* A basic predictive model (linear regression) to forecast future demand.
* CSV outputs suitable for visualization in Python or other tools.

d. Application Overview and Technologies used

* **Technologies:**
  + Scala and Spark: For data ingestion, cleaning, transformation, aggregation, and modeling.
  + Local environment: Running Spark locally
  + Python/Plotly: For creating visualizations outside the Scala/Spark pipeline.
* **Pipeline Overview:**
  + **Collection:** Download and unzip trip data from S3 and station data from GBFS.
  + **Ingestion:** Load CSV data into Spark.
  + **Preparation:** Clean and join trip data with station data.
  + **Computation:** Compute monthly/quarterly trends, growth rates, station-level usage, and run a linear regression model for demand forecasting.
  + **Presentation:** Produce CSV outputs that can be visualized using Python and Plotly.

e. Bonus Options:

* Visualization: Created a set of recommended visualization plans (heatmaps, line charts, bar charts) using Python/Plotly/Seaborn/Matplotlib.

**2. [50 points] Milestone 1: System Design and Architecture**  
a. System Diagram

A diagram of data processing

Description automatically generated

**Description:**

* The system consists of a local Spark environment reading CSV files, performing ETL operations, and producing cleaned, aggregated datasets and predictions.

b. Description of each stage of the processing pipeline

**Collection:**

* Download trip data from the specified S3 bucket.

**Ingestion:**

* Load local CSV files into Spark DataFrames.

**Preparation:**

* Standardize schemas and timestamps.
* Join trip data with station data on station name or ID.

**Computation:**

* Compute monthly and quarterly aggregates.
* Calculate month-over-month growth rates.
* Run a linear regression model to predict future demand.
* Perform station-level analyses for capacity and usage patterns.

**Presentation:**

* Write out CSV results for monthly trends, station usage, growth rates, and predictions.
* These CSVs are then used to create visualizations in Python/Plotly.

**3. [100 points] Milestone 2: Implementation and Results**  
a. Code and configuration snippets

* Implemented Scala code for:

data download (BluebikesDownloader.scala)

station data transformation (StationDataTransformer.scala),

station data analysis (StationDataAnalysis.scala)

station data downloader (StationDataDownloader.scala)

station geographic analysis (StationGeographicAnalysis.scala)

temporal analysis (TemporalAnalysis.scala)

data validation (DataValidation.scala),

monthly/quarterly analysis (MonthlyAndQuarterlyAnalysis.scala),

growth rate analysis (GrowthRateAnalysis.scala),

station-level analysis (StationLevelAnalysis.scala),

demand prediction (DemandPrediction.scala).

(these scala files have been saved in the Full\_Source\_Code.zip under pathname /src/main/scala/bluebikes)

* Code comments and structured logic provided.
* Schemas and transformations enforced in Spark code.

b. Screenshots of run commands and processing results

* Due to verbosity, output logs were not fully captured, but CSV outputs confirm successful completion.
* Running commands via sbt run and selecting appropriate main classes generated final CSVs (e.g., monthly\_trend.csv, predictions.csv)

(these csv files have been saved in the Input\_and\_Result\_data.zip)

Overview

The code is divided into several modules, each performing a specific part of the data pipeline:

1. Data Acquisition and Extraction:
   * BluebikesDownloader.scala: Downloads monthly trip data files (originally Hubway data, now Bluebikes) from an AWS S3 bucket and unzips them into a local directory.
   * StationDataDownloader.scala: Fetches station information (in JSON format) from Bluebikes’ GBFS feed.
   * StationDataTransformer.scala: Converts the downloaded station JSON data to CSV format for easier Spark-based analysis.
2. Data Validation and Cleaning:
   * DataValidation.scala: Validates the downloaded trip and station data against predefined schemas to ensure data integrity and cleanliness.
3. Data Aggregation and Analysis:
   * MonthlyAndQuarterlyAnalysis.scala: Aggregates trip data on a monthly and quarterly basis, calculating total trips and average durations.
   * GrowthRateAnalysis.scala: Calculates month-over-month growth rates in total trips.
   * StationDataSparkAnalysis.scala: Examines station-level data, including total capacity and capacity distributions.
   * StationGeographicAnalysis.scala: Performs geographic binning of stations based on latitude/longitude to analyze spatial distributions.
4. Predictive Modeling:
   * DemandPrediction.scala: Uses Spark MLlib to train a regression model for predicting monthly demand (total trips) based on selected features (e.g., month, average duration). It generates predictions and saves them to CSV.
5. Station-Level Insights:
   * StationLevelAnalysis.scala: Joins trip data with station data to analyze usage patterns at the individual station level over time.

File-by-File Details

BluebikesDownloader.scala

* Purpose:  
  Downloads ZIP files containing trip data from the AWS S3 bucket and unzips them locally.
* Key Steps:
  1. Makes an HTTP request to the S3 index page.
  2. Parses the returned XML to extract the names of all .zip files available.
  3. Downloads each .zip file if not already present and extracts them into an unpacked/ directory.
* Result:  
  After running this, the local unpacked/ directory will contain .csv files with trip data for different time periods.

DataValidation.scala

* Purpose:  
  Ensures that the downloaded data conform to expected schemas. For the trip data, it enforces data types (e.g., IntegerType for tripduration), and for station data, it checks for required fields like station\_id and name.
* Key Steps:
  1. Reads tripdata.csv and station\_data.csv into Spark DataFrames using a predefined schema.
  2. Checks for null values in critical columns and ensures numeric columns do not have invalid values (e.g., negative capacities).
* Result:  
  If validations pass, the data is clean and ready for further analysis. If not, it throws exceptions.

DemandPrediction.scala

* Purpose:  
  Uses a simple regression model (Linear Regression) to predict total trips in a given month based on features such as month and average\_duration.
* Key Steps:
  1. Reads the monthly\_trend.csv (generated by MonthlyAndQuarterlyAnalysis).
  2. Cleans the data and prepares features using a VectorAssembler.
  3. Splits the data into training and testing sets and trains a Linear Regression model.
  4. Evaluates the model using RMSE (Root Mean Squared Error).
  5. Saves predictions to ./unpacked/predictions.csv.
* Result:  
  Generates a predictions CSV file with predicted total trips, allowing assessment of future demand.

GrowthRateAnalysis.scala

* Purpose:  
  Computes month-over-month growth rates in total trips.
* Key Steps:
  1. Reads monthly\_trend.csv.
  2. Uses Spark window functions (lag) to compare the current month’s total trips with the previous month’s total trips.
  3. Writes out growth\_rates.csv.
* Result:  
  A CSV file showing how ridership changes from one month to the next, providing insights into trend growth patterns.

MonthlyAndQuarterlyAnalysis.scala

* Purpose:  
  Aggregates the raw trip data into monthly and quarterly summaries.
* Key Steps:
  1. Reads all \*-tripdata.csv from the unpacked/ directory.
  2. Extracts year, month, and quarter from trip start times.
  3. Groups by (year, month) and (year, quarter) to compute total\_trips and average\_duration.
  4. Saves results as monthly\_trend.csv and quarterly\_trend.csv.
* Result:  
  Two CSV files with monthly and quarterly aggregated metrics, used as input for other analyses like GrowthRateAnalysis and DemandPrediction.

StationDataDownloader.scala and StationDataTransformer.scala

* Purpose:
  + StationDataDownloader: Fetches station information (JSON) from the Bluebikes GBFS feed and saves it locally.
  + StationDataTransformer: Converts the downloaded station\_information.json into a clean CSV file (station\_data.csv) suitable for Spark analysis.
* Result:  
  A station\_data.csv file containing station IDs, names, lat/lon coordinates, and capacity.

StationDataSparkAnalysis.scala

* Purpose:  
  Performs various analyses on station data, such as total capacity and capacity distributions.
* Key Steps:
  1. Reads station\_data.csv.
  2. Summarizes station capacity (total, average, distribution by bins).
  3. Writes out results such as capacity\_bins.csv and max\_capacity\_station.csv.
* Result:  
  CSV files showing distributions and station with the highest capacity.

StationGeographicAnalysis.scala

* Purpose:  
  Categorizes stations into geographic bins (e.g., South, Central, North; East, West) and aggregates capacity based on geography.
* Result:  
  A CSV file geographic\_distribution.csv summarizing station capacities by geographic region.

StationLevelAnalysis.scala

* Purpose:  
  Combines trip data with station information to determine usage patterns at the station level over time.
* Key Steps:
  1. Normalizes station names to join trip data and station data on a common key.
  2. Aggregates by (station\_id, year, month) to find the number of trips starting from each station and their average durations.
  3. Saves results as station\_usage.csv.
* Result:  
  Insights into how each station's usage evolves month to month.

Running the Code

After compiling sbt run and then selecting the desired main class, each object with a main method can be executed independently to perform its respective task.

Because large amounts of log output may be produced, I redirect output to a file or rely on the printed CSV outputs to verify the final results because terminal logs are extensive. The key artifacts from each analysis step are the CSV files written out to unpacked/ or station\_data/ directories.

Results and Artifacts:

* Trip Data CSVs in unpacked/.
* Monthly and Quarterly Trends in ./unpacked/monthly\_trend.csv and ./unpacked/quarterly\_trend.csv.
* Growth Rates in ./unpacked/growth\_rates.csv.
* Demand Predictions in ./unpacked/predictions.csv.
* Station Data in ./station\_data/station\_data.csv.
* Station-Level Capacity Bins in ./station\_data/capacity\_bins.csv.
* Station Geographic Distribution in ./station\_data/geographic\_distribution.csv.
* Station-Level Usage in ./unpacked/station\_usage.csv.

Each of these files can be opened or further processed to create visualizations, dashboards, or to be integrated into additional analysis steps.

Summary

Full code pipeline: from downloading data, validating it, transforming it, and performing various analytics and predictive modeling tasks. The output artifacts are CSV files stored locally, and I’ve designed the code so that it can be re-run to refresh data and results. This modular approach makes it easy to build upon as I move to the results and visualizations in Python.

**Results, Analysis, Conclusions, and Challenges**

**Summary of Results**

The project successfully implemented a modular and scalable pipeline for analyzing Bluebikes bikeshare data. Key results include:

1. **Data Acquisition:** Automated download and extraction of monthly trip data and station data from AWS S3 and Bluebikes' GBFS feed.
2. **Data Validation:** Schema enforcement ensured clean and reliable datasets for further analysis. Data validation passed with no critical issues.
3. **Monthly and Quarterly Analysis:** Aggregated metrics such as total trips and average trip duration by month and quarter, saved to CSV files.
4. **Growth Rate Analysis:** Computed month-over-month growth rates, providing insights into ridership trends.
5. **Predictive Modeling:** Developed a regression model using Spark MLlib to predict monthly trips based on features such as month and average duration. The model achieved an RMSE of on the test set.
6. **Station-Level Insights:** Analyzed station capacity and trip patterns, including capacity distribution and geographic distribution. Results were saved as CSV files for visualization.
7. **Geographic Analysis:** Categorized stations into geographic bins and aggregated capacity by region.
8. **Station-Level Usage:** Identified monthly trip counts and average trip durations for each station.

The outputs were saved as CSV files for easy integration into visualization and reporting tools.

(these csv files have been saved in the Input\_and\_Result\_data.zip)

**Potential Options for Scaling to a Distributed System**

To scale this project for larger datasets and distributed systems:

1. **Leverage a Cloud-Based Cluster:**
   * Deploy the Spark jobs on a cloud platform like AWS EMR, Google Dataproc, or Azure HDInsight to utilize distributed processing for larger datasets.
   * Store data in distributed storage solutions like S3, HDFS, or Google Cloud Storage.
2. **Use Kubernetes for Resource Management:**
   * Containerize the Spark applications and deploy them on Kubernetes clusters to enable efficient resource allocation and scalability.
3. **Optimize Spark Configurations:**
   * Tune Spark configurations, including executor memory, partitions, and caching strategies, to improve performance for large-scale data.
4. **Batch vs. Streaming Processing:**
   * Integrate streaming data pipelines using Spark Structured Streaming for real-time analysis of trip and station data.
5. **Database Integration:**
   * Store processed results in distributed databases like Snowflake, BigQuery, or Cassandra for efficient querying and visualization.

**Application of Functional Programming Principles**

The project applied key functional programming principles as follows:

1. **Immutability:**
   * DataFrames in Spark were treated as immutable, with transformations creating new DataFrames rather than modifying existing ones.
2. **Declarative Programming:**
   * Leveraged high-level APIs in Spark, such as select, groupBy, and withColumn, to express transformations declaratively rather than specifying execution steps.
3. **Higher-Order Functions:**
   * Used higher-order functions like map, filter, and reduce to apply transformations to RDDs and DataFrames.
4. **Modularity:**
   * Divided the project into reusable and independent modules (e.g., downloading, validation, analysis) to ensure better maintainability and testability.
5. **Lazy Evaluation:**
   * Spark’s execution model ensured transformations were lazily evaluated, optimizing performance by executing only the necessary computations.

**Challenges with the Project and What to Do Differently Next Time**

**Challenges:**

1. **File Overwriting Issues:**
   * Encountered errors due to existing output files when writing results. Resolved by adding mode("overwrite") to save operations.
2. **Large Log Outputs:**
   * Excessive log outputs made debugging difficult. Managed by redirecting logs to files and adjusting log levels.
3. **Schema Mismatches:**
   * Ensuring consistent schemas across all modules required extra validation steps.
4. **Resource Limitations:**
   * Running Spark jobs locally was resource-intensive, limiting scalability.

**Improvements for Future Iterations:**

1. **Automated Testing:**
   * Implement unit tests for each module to catch errors early.
2. **Centralized Logging:**
   * Use centralized logging solutions (e.g., ELK Stack) for better log management.
3. **Cluster Deployment:**
   * Deploy the project on a cloud-based Spark cluster for better performance and scalability.
4. **Real-Time Processing:**
   * Extend the pipeline to handle real-time data streams for dynamic insights.
5. **Enhanced Visualizations:**
   * Integrate visualization libraries directly into the pipeline for immediate results.

**YouTube Video URL**

[**https://youtu.be/azFLVl8BtNg**](https://youtu.be/azFLVl8BtNg)

**Bonus: Visualization**

**Titles, Descriptions, and Takeaways for Final Plots**

(these plots were built in Jupyter Notebooks using the csv outputs made by the scala files. The jupyter notebook has been saved as an ipynb file named Scala\_Project\_Visuals and has been saved in the Input\_and\_Result\_data.zip)

**Plot Title: Hourly Trip Trends by Geographic Region**

**Description:** This line chart visualizes hourly trip trends across different geographic regions: Northeast, Northwest, Southeast, and Southwest. The x-axis represents hours of the day, while the y-axis shows the total number of trips. **Takeaways:**

* The Southeast region experiences the highest trip volume, especially during afternoon and evening hours.
* The Northeast region shows consistently low trip volumes throughout the day.
* Trip volumes peak around 6 PM, indicating high activity during the evening rush hour across all regions.A graph of different colored lines

  Description automatically generated

**Plot Title: Station Peak Usage Times**

**Description:** A map with labeled stations indicating their peak usage times (Morning, Afternoon, Evening, or Night). Stations without a specific peak time are labeled as "No Peak Time." **Takeaways:**

* Evening is the most common peak time, especially in central locations.
* Morning and Afternoon peaks are more scattered geographically, possibly influenced by commuting patterns.
* "No Peak Time" stations represent less-utilized areas.

A map of a city

Description automatically generated

**Plot Title: Hourly Trip Trends for Top Stations**

**Description:** A line chart showing hourly trip volumes for the top five stations with the highest total trip counts. **Takeaways:**

* All top stations exhibit a sharp increase in activity during afternoon and evening hours.
* MIT at Mass Ave / Amherst St consistently has the highest trip volumes throughout the day, peaking at 6 PM.A graph of different colored lines

  Description automatically generated

**Plot Title: Monthly Trends: Total Trips and Average Duration by Year-Month**

**Description:** A combination bar and line chart, where bars represent the total trips and the line shows the average trip duration, plotted over time across multiple years. **Takeaways:**

* Total trips peak during the summer months (July–September), likely due to favorable weather conditions.
* Average trip durations spike during summer months, possibly indicating recreational or non-commuter usage.
* Both metrics exhibit seasonality, underscoring weather's influence on ridership patterns.
* Total trip number is increasing but average trip duration is decreasing, potentially showing a change in overall rider behavior such as more people using bikes for short trips and the average user going less far. This could also be due to an increase in number of available stations. Users don’t have to travel as far to redock a bike. A graph with blue and orange lines

  Description automatically generated

**Plot Title: Top 10 Stations by Trip Count**

**Description:** A bar chart displaying the top 10 stations based on total trip counts. **Takeaways:**

* MIT at Mass Ave / Amherst St is the most popular station, with significantly higher trip counts compared to others.
* Stations near major hubs or academic institutions dominate the list, highlighting their importance in the network.

A graph of a number of people

Description automatically generated with medium confidence

**Plot Title: Month-over-Month Growth Rates in Total Trips**

**Description:** A line chart showing the month-over-month growth rates in total trips for a specific year. **Takeaways:**

* Strong growth is observed during spring and early summer months, followed by a decline in the fall.
* Negative growth in winter months could reflect reduced ridership due to weather conditions.
* This analysis provides insight into seasonal fluctuations in demand.

A graph with green line

Description automatically generated

**Plot Title: Station Capacity by Geographic Region**

**Description:** This heatmap shows the distribution of station capacity across different geographic regions, categorized by latitude and longitude bins. The darker the color, the higher the station capacity in that region.

**Takeaways:**

* Central regions (both East and West) have the highest station capacities, likely reflecting dense urban areas.
* Northern and Southern regions have significantly smaller capacities, indicating less demand or population density in those areas.

A graph of a station capacity

Description automatically generated

**Plot Title: Number of Stations by Capacity Bin**

**Description:** This bar chart categorizes all stations into three capacity bins: Small, Medium, and Large. It provides an overview of the capacity distribution across the network.

**Takeaways:**

* Most stations fall into the "Medium" capacity category, suggesting a standardized design for station infrastructure.

A graph of a number of stations

Description automatically generated

**Plot Title: Month-over-Month Growth Rates in Total Trips by Year**

**Description:** This line chart compares month-over-month growth rates in total trips across different years. Each line represents a separate year, highlighting seasonal trends and growth patterns.

**Takeaways:**

* Significant seasonal spikes in growth are observed around spring and summer months (e.g., May–June).
* Some years, such as 2020, show unusual patterns, likely influenced by external factors like the COVID-19 pandemic.

A graph with lines and dots

Description automatically generated

**Plot Title: Actual vs. Predicted Total Trips Over Time**

**Description:** This time series plot compares actual total trips across years with predicted trips for the latest year. The actual data is broken down by year, while predictions are shown as dashed lines for the latest period.

**Takeaways:**

* The predictive model predicts some of the season pattern, such as when there being more trips in the summer versus winter. However it doesn’t account for the growth year to year, and predicts the year 2022 to look similar to 2021 when in reality it is a larger year in total.
* The model could be fine-tuned for periods of lower activity, as variability increases in these months. The model has much more noise than the actual data.

A graph with colored lines and numbers

Description automatically generated