CloudWalk, Inc.

11.06.2024

—

Arthur Santos Marinho de Souza
Information Systems @ UFPE -CIn (2023.1-2026.2)
UFPE Finance
Business Monitoring Intelligence Intern Candidate

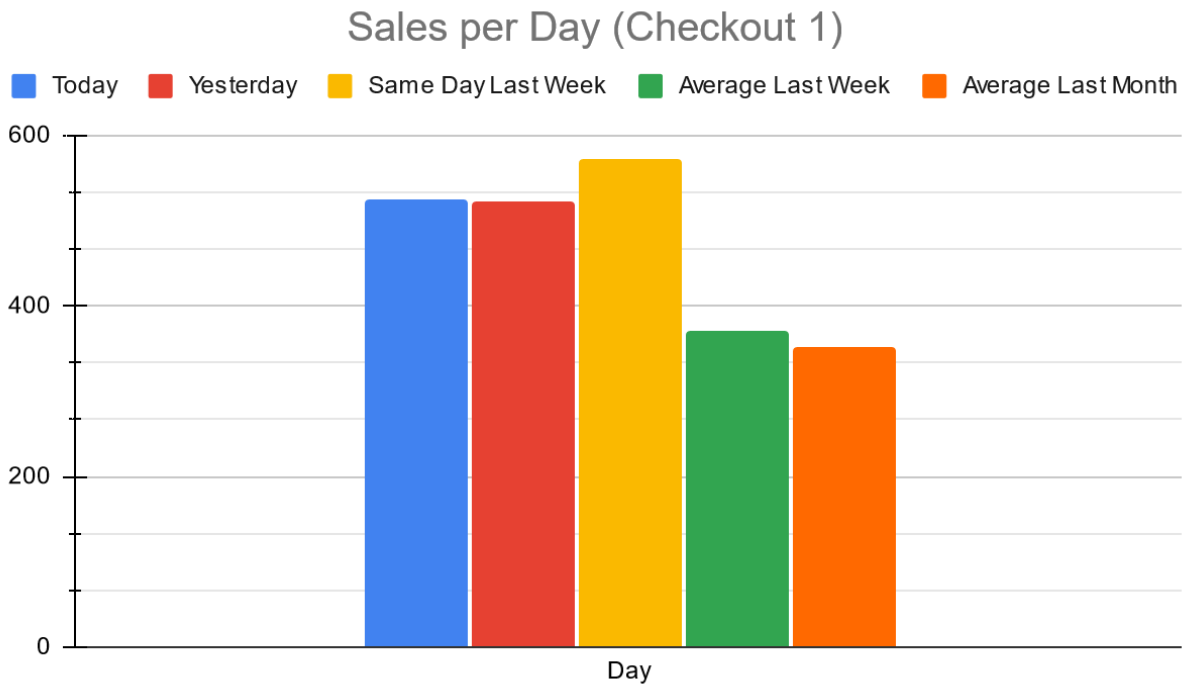
Spreadsheets

Analyzing the data

We are presented with two CSV (Comma-Separated Values) files containing data about the number of sales per hour in the determined Points of Sale (PoS). To analyze the data, I used Google Sheets, since it makes reading the spreadsheets easier and also helps to visualize the data using charts. Below, I list how I achieved the final result (measures were applied to both spreadsheets).

- ➔ Imported the CSV file to Google Sheets
- ➔ Changed Column Names to more readable format (average_last_week > Average Last Week)
- ➔ Created a row of data containing the number of sales for the full day (row 26)
- ➔ Inserted line chart to view the number of sales per hour
- ➔ Inserted column chart to view the total number of sales for the day

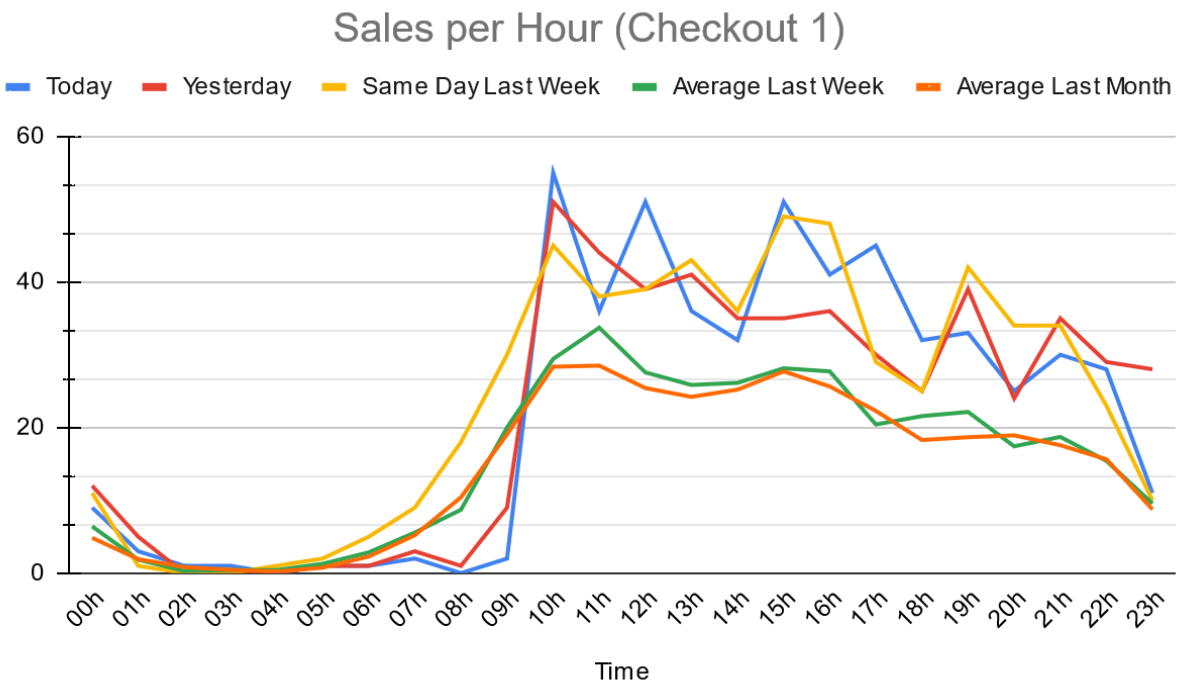
I. Checkout 1



On the chart “Sales per day (Checkout 1)”, we are able to visualize that the values of specific

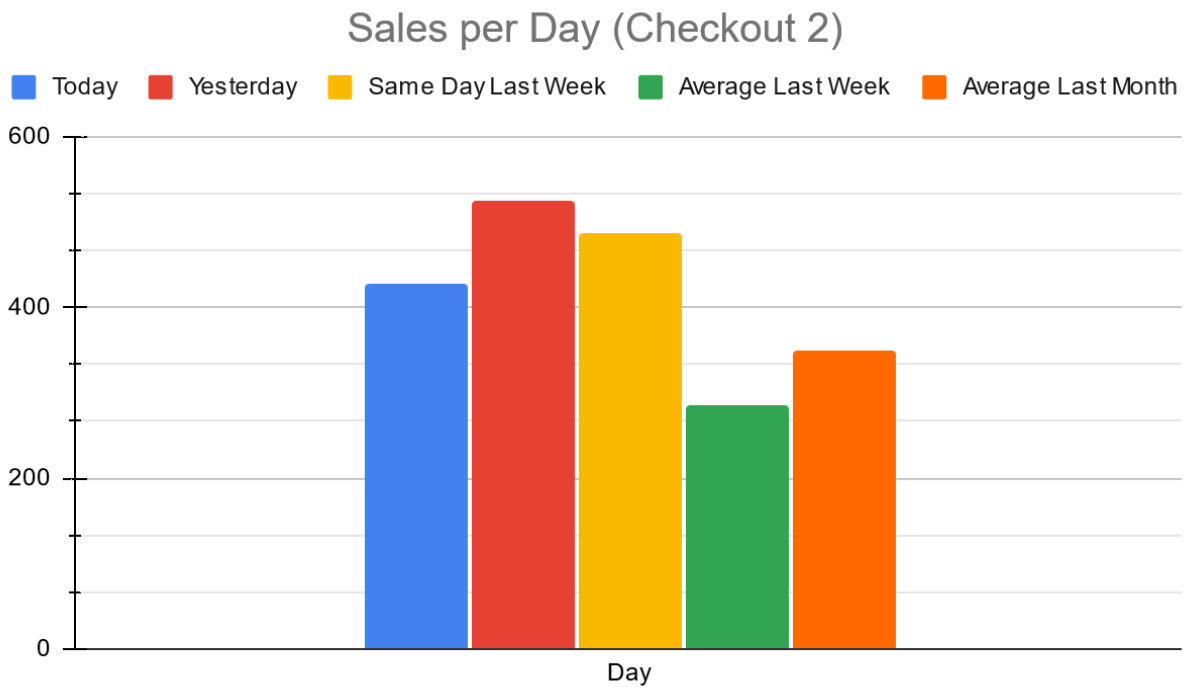
days (“Today”, “Yesterday” and “Same Day Last Week”) are much higher than both averages (“Average Last Week” and “Average Last Month”), suggesting that those are days that receive a higher number of customers. Since weekends tend to be more lucrative for commercial establishments related to leisure (e.g. restaurants, shopping malls, movie theaters, etc.), a plausible explanation would be:

- ➔ “Today” and “Same Day Last Week” are Sundays
- ➔ “Yesterday” is a Saturday
- ➔ The checkouts are from a company with operations related to leisure

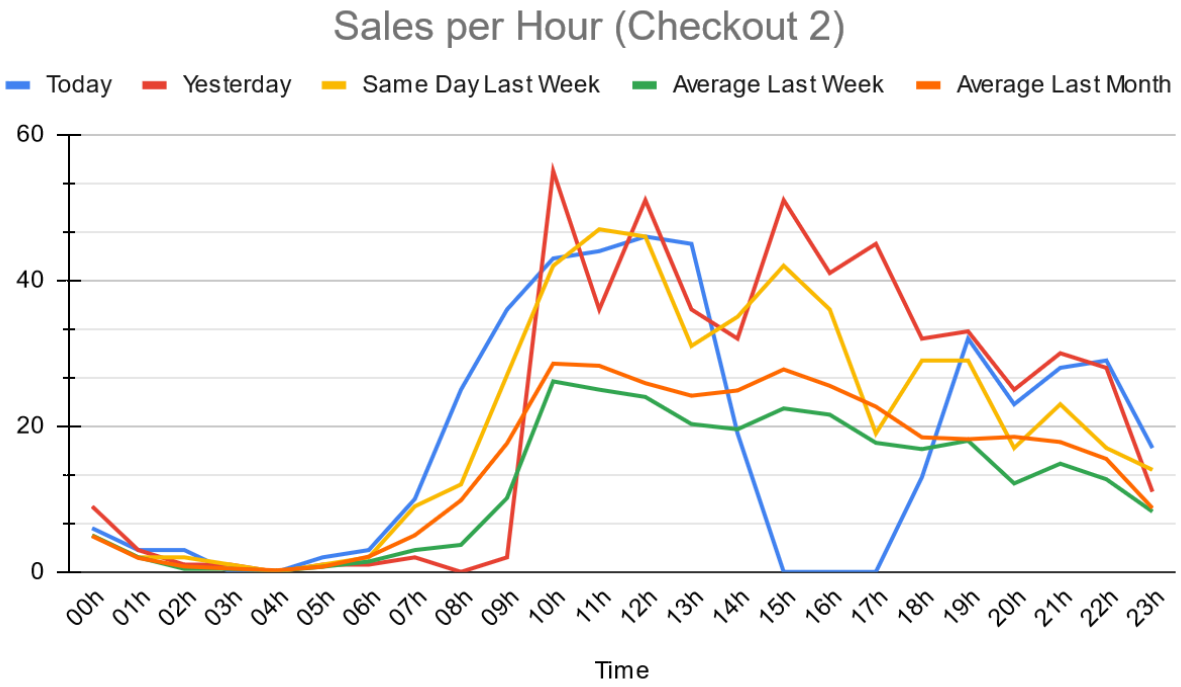


The chart “Sales per Hour (Checkout 1)”, contains the hourly rate of sales. The store has seen a strong decrease in sales from the 6h to 9h mark and an increase in sales around 10h, possibly indicating a later opening time. This would explain both phenomena, with the early sales being moved to 10h.

II. Checkout 2



In the above chart, we are able to see a similar trend as pointed out in the analysis of Chart 1, with the difference that there is a noticeable variation between “Today”, “Yesterday” and “Same Day Last Week”. This could be explained by some seasonal event, such as a soccer match that drew fans to the location yesterday.



In “Sales per Hour (Checkout 2)”, we see a weird behavior of “Today” compared to both the other days at “Checkout 2” and “Today” at “Checkout 1”, with an increased number of sales early in the morning and no sales between 15h and 17h, suggesting that the payment gateway was likely down during that period.

III. Conclusion

This could be a delivery restaurant and the Checkouts are delivery apps, such as Rappi and iFood. I believe that since there are few payments made while the restaurant is closed (you are able to schedule an order, but that is uncommon, and the time of the payments would match the flow of order of a restaurant focused on selling food for breakfast, but also lunch (a bakery or a coffee shop).

Database

Creating Tables

To create SQL Tables, I have used MySQL and JavaScript. The Javascript file contains the connection standards in addition to the SQL Queries. A JS file was needed instead of a simple SQL one due to the import of a system file (the CSV). Here is an overview of the code:

Module Imports: The script imports necessary modules to interact with the file system and MySQL database.

Database Connection: Establishes a connection to a MySQL database using specified credentials.

Create Table: Defines and executes an SQL query to create a table with columns for time (primary key), today's data, yesterday's data, data from the same day last week, the average data for the last week, and the average data for the last month.

File Path Load Data Query: Specifies the path to the CSV file that contains the data to be loaded into the table.

Load Data Query: Defines an SQL query to load data from this CSV file into the table, specifying how fields and lines are terminated and instructing the database to ignore the first line of the file, which is assumed to contain headers.

Execute Load Data Query: Executes the query to load data from the CSV file into the table and uses a file stream to read the CSV file and logs any errors or results from this operation.

Close Database Connection: Closes the connection to the database after completing the data loading operation.

Analyzing Anomaly Behavior

I have used python to plot the graphs analyzing the behavior. Here is a summary of the code:

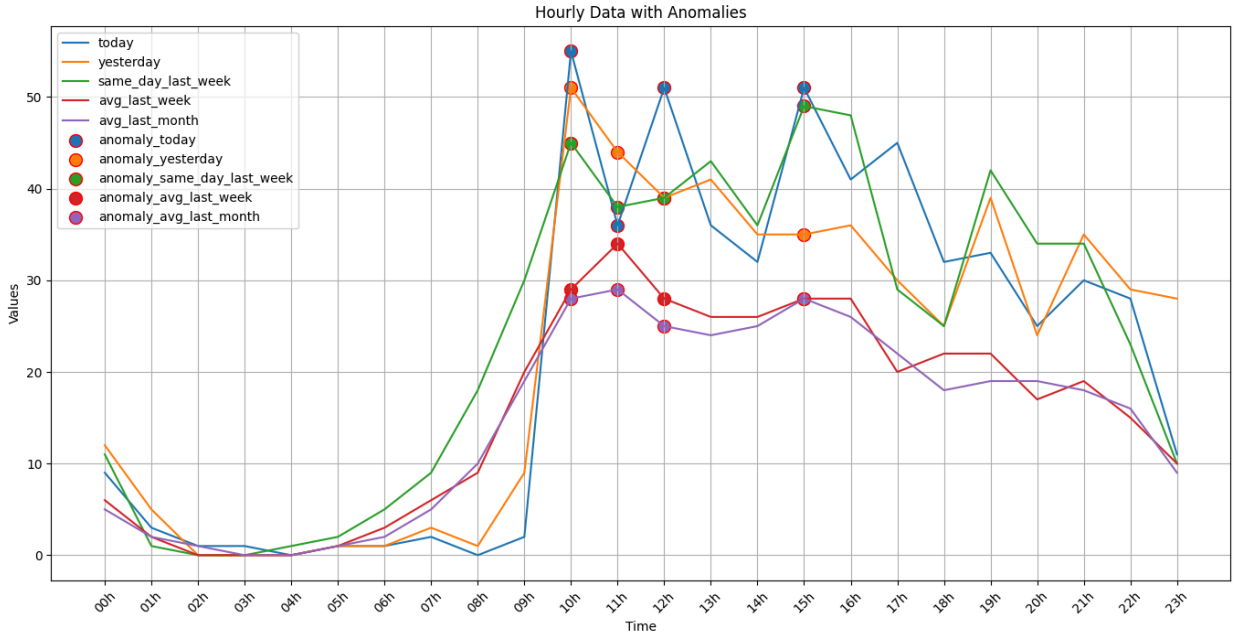
Database Operations: Connects to a MySQL database and retrieves data.

Z-Score Calculation: Computes Z-scores to standardize the data for anomaly detection.

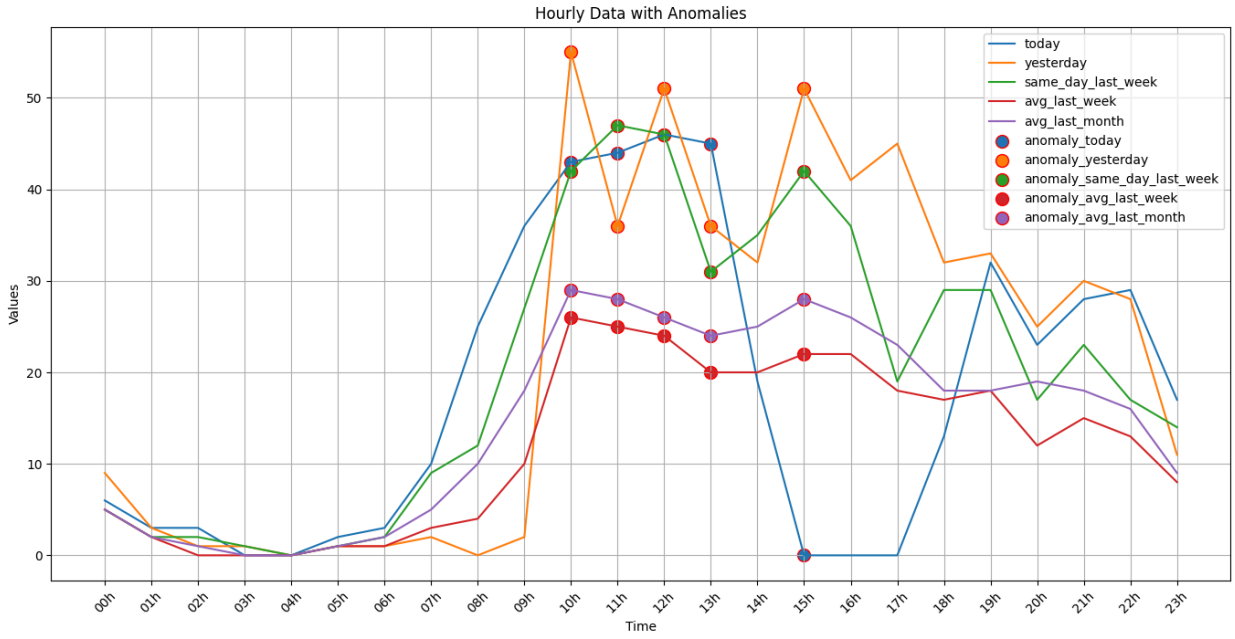
Anomaly Identification: Finds anomalies based on Z-score thresholds.

Visualization: Plots the data and highlights anomalies.

Output: Prints anomalies to the console.



This is for checkout_1



This is for checkout_2

Monitoring Alert System

Analyzing the data

Firstly, we need to study the spreadsheets and understand what would be considered an anomaly. There are two main ways to detect them: the z-score and whiskers. The z-score calculates how many standard deviations the value is from the mean and the whiskers are data points placed 1.5 interquartile range below Q1 and above Q3. Since we are trying to find positive anomalies, we will be using z-scores above 3 and the upper whisker as our measures.

Library Imports: Imports the pandas library for data manipulation and analysis.

Loading the CSV file: Loads transaction data from a CSV file (transactions_1.csv) into a DataFrame named transaction_data.

Creating a Pivot Table: Creates a pivot table from the DataFrame with time as rows, status as columns, and the count of transactions (f0_) as values.

Adding Row Sum: Adds a new column (Row_Sum) to the pivot table, which is the sum of transactions for each time period.

Handling Missing Values: Fills any NaN values in the pivot table with 0.

Calculating Transaction Ratios: Creates a ratio table where each value is the ratio of transactions of each type to the total transactions at each time period.

Dropping Row Sum: Drops the Row_Sum column from the ratio table.

Calculating Whisker Values: Calculates the first quartile, third quartile, and interquartile range for each transaction type, and determines the whisker value (upper bound for potential outliers).

Filtering Outliers: Filters the ratio table to find time periods where the ratio of denied, failed, or reversed transactions exceeds the respective whisker values, storing the result in filtered_whisker.

Calculating Z-Scores: Computes the Z-score for each value in the ratio table and stores it in the z_score_table.

Filtering Extreme Z-Scores: Filters the Z-score table to find time periods where the Z-score for denied, failed, or reversed transactions exceeds 3, indicating potential anomalies, and stores the result in filtered_scores.

Endpoint

We set up an endpoint using flask that will upload our csv with the data needed to perform future operations and make it so there is a way to get the data back

Flask Setup: Initializes a Flask application to handle HTTP requests.

CSV Loading: Loads a CSV file named `transactions_1.csv` at application startup into a global DataFrame `df`.

Error Handling: Provides appropriate error messages and status codes for different failure scenarios (e.g., file not loaded, no matching rows).

Application Execution: Loads the CSV and starts the Flask server in debug mode.

Making requests

Getting the data that matches our requirements back

Parameters: `base_url` (the base URL of the Flask application) and `time_value` (The time value to filter rows by).

Endpoint: `/rows`

URL Construction: Constructs the full URL by appending the endpoint to the base URL.

GET Request: Sends a GET request with `time` as a query parameter.

Error Handling: Raises an exception for HTTP errors and prints a descriptive message and handles any other exceptions and prints a generic error message.

Response Handling: Converts the JSON response into a pandas DataFrame if the request is successful.

Writing email

As the last part of our Monitoring Alert System, we will be sending an email from our newly created monitoringanalysttest@outlook.com to a receiver of our choice.

Library Imports: Imports necessary libraries for handling SMTP connections (smtplib) and constructing email content (MIMEMultipart, MIMEText).

Function Definition: Defines the function write_email(body) to send an email with the provided body content.

Email Details: Sets the sender email (monitoringanalysttest@outlook.com), receiver email (arthursmds@gmail.com), and email subject (Anomaly detected).

SMTP Server Configuration: Specifies SMTP server details including server address (smtp-mail.outlook.com), port (587), and sender's email password.

Email Construction: Creates an email object using MIMEMultipart, sets the 'From', 'To', and 'Subject' headers, and attaches the email body as plain text using MIMEText.