INF FACULTY OF COMPUTER SCIENCE

Documentation VLBA II - System Architectures

Project II: Consulting & Customer Retention

Manu Benny (244058)
Nived Bijukumar (250894)
Atif Harshad (244570)
Ijaaz Muhammed Mullamangalam (244754)
Adem Zarrouki (231613)

Magdeburg, July 3, 2024

Otto-von-Guericke-University Magdeburg
Magdeburg Research and Competence Cluster
Faculty of Computer Science
Department of Technical and Business Information Systems
Very Large Business Applications Lab

Contents

List of Figures

List of Abbreviations

API	Application Programming Interface
BSP	Business System Planning
CCR	Consulting & Customer Retention
CRM	Customer Relationship Management
CSV	comma-separated values
DWC	SAP Data Warehouse Cloud
ERP	Enterprise Resource Planning
ETL	Extract, Transform, Load
GCP	Google Cloud Platform
GCS	Google Cloud Storage
JSON	JavaScript Object Notation
ML	Machine Learning
MPD	Manufacturing & Product Distribution
MS	Maintenance & Service
MWW	Mobility Worldwide
PLM	Product-Lifecycle-Management
RSD	Research & Solution Development
SCM	Supply Chain Management
SOA	Service-Oriented Architectures
SQL	Structured Query Language
SRM	Supplier Relationship Management
UI	User Interface
UML	Unified Modeling Language

1 Introduction

Mobility Worldwide (MWW), headquartered in Magdeburg, stands as a global leader in the mobility sector, renowned for its innovation, production, talent promotion, and comprehensive services. To maintain its competitive edge in a rapidly evolving market, MWW is dedicated to evaluating and integrating cutting-edge technologies into its operations. The company's structure is divided into four strategic departments: Consulting & Customer Retention (CCR), Manufacturing & Product Distribution (MPD), Research & Solution Development (RSD), and Maintenance & Service (MS). Each department functions independently but collaborates synergistically to ensure the overall success of the organization. The Consulting & Customer Retention (CCR) department plays a crucial role in MWW's operational success and external perception. Its responsibilities encompass consulting clients across a spectrum of relationships, from large car dealerships to small, premium-quality driving service providers, and representing the company at trade fairs. Organized into strategic regions, each with multiple branch offices, the CCR department ensures efficient and effective service delivery and client engagement.
This project focuses on analyzing existing business data from the CCR department to evaluate the profitability of its current branch offices and to assess the feasibility of opening a new branch in a strategic region. The analysis will utilize historical data on employees, business trips, maintenance costs, and required business assets, supplemented by external data on hotel rates in frequently visited cities.

1.1 Objectives

The primary objectives of this project are as follows:

1. **Evaluate Average Costs**: Determine the average costs associated with operating a branch office. This includes analyzing rent, personnel expenses, maintenance costs, and other relevant expenditures.

2. **Visualize Business Trips**: Examine and visualize the business trips undertaken by consultants, particularly focusing on those trips that necessitate overnight stays due to the distance traveled.

3. **Assess New Branch Feasibility**: Investigate the potential benefits and feasibility of opening a new branch office in the strategic region. This involves identifying a suitable location and forecasting the potential impact on profitability and operational efficiency.

1.2 Methodology

The project will be conducted using the following methodology:

1.2.1 Data Collection and Preparation

Gather historical data from MWW's internal SAP DWC system and external data sources. The internal data will include information on employees, previous business trips, maintenance costs, and required business assets. External data will provide insights into hotel rates in frequently visited cities.

1.2.2 Cost Evaluation

Analyze the provided data to calculate the average costs associated with running a branch office. This analysis will consider all relevant expenses, such as rent, personnel, maintenance, and operational costs.

1.2.3 Business Trip Visualization

Utilize visualization tools to map out and analyze the business trips of consultants. Special attention will be given to trips requiring overnight stays, leveraging the company's policy of providing lodging for trips that are at least a one-hour drive away.

1.2.4 Feasibility Study for New Branch

Conduct a comprehensive study to assess the feasibility of opening a new branch office. This will include proposing a new location, visualizing the current subdivision of the region, and forecasting the potential impact on profitability if the branch had been opened one year prior.

1.3 Expected Outcomes

The expected outcomes of this project include:

- A detailed report documenting the average costs of operating a branch office.
- Visualizations of the business trips taken by consultants, highlighting those requiring overnight accommodations.
- A feasibility study and recommendation for the potential location of a new branch office, along with a forecast of its impact on profitability.

By providing a thorough analysis of the CCR department's operations, this project aims to equip the MWW board with valuable insights, enabling informed decision-making regarding the optimization of branch office locations and the overall profitability of the CCR department. The integration of advanced analytics and visualization tools will further enhance the strategic planning and operational efficiency of MWW.

2 System Architecture

In this chapter, we will discuss the integration process between SAP DWC and GCP and how the system architecture of our project integrates various components to efficiently manage and analyze data sourced from SAP Data Warehouse Cloud (SAP DWC). We will describe the architecture using UML component diagram notations, justify the design with Business System Planning (BSP) diagrams, and highlight relevant cloud patterns. This comprehensive approach ensures that our architecture adheres to the principles of composability, modern system design, and best practices.

2.1 Integration of SAP Data Warehouse Cloud (DWC) with Google Cloud Platform (GCP)

In this part of the chapter, we will discuss the integration process between SAP Data Warehouse Cloud (DWC) and Google Cloud Platform (GCP). Although the practical implementation was not carried out for our project, understanding the integration process is crucial for replicating and scaling this approach in a real-world scenario.

Overview of Data Integration Process

The interconnection of SAP Data Warehouse Cloud (DWC) and Google Cloud Platform (GCP) is crucial to leverage existing business data to support decision-making processes within the Consulting Customer Retention (CCR) department of Mobility Worldwide (MWW). This integration enables the extraction, transformation, and analysis of historical data to derive insights that can drive strategic decisions, such as evaluating branch office profitability and proposing new locations.

Technologies and tools Used

- SAP Data Warehouse Cloud (DWC): SAP DWC is a comprehensive data management solution that integrates data from various SAP frameworks and external sources and serves as the primary source of internal company data, encompassing various datasets such as branch office details, employee information, business trip records, and branch running costs. These datasets are crucial for comprehensive analysis and decision-making.
- Google Cloud Platform (GCP): GCP provides a robust environment for data storage, analysis, and visualization. The specific GCP services utilized include:
  - Google Cloud Storage (GCS): Acts as an intermediary storage solution for staging data extracted from SAP DWC before loading it into BigQuery.

- BigQuery: A powerful data warehouse solution used for storing and querying large datasets efficiently. It supports advanced analytics and machine learning models (BigQueryML) to generate actionable insights.
- Looker Studio: A visualization tool used for creating interactive dashboards and reports, helping in the visual examination of the current subdivision of the region and business trip visualizations.

Integration Process

Figure 2.1: Theoretical Integration Architecture between SAP DWC and GCP

The integration process between SAP DWC and GCP involves several key steps to ensure seamless data transfer, storage, and analysis.

1. **Data Extraction from SAP DWC**
   - SAP Data Services: Utilize SAP Data Services or SAP Data Intelligence to extract data from the SAP DWC. These tools allow for the creation of data extraction jobs that can pull data from various SAP tables and datasets.
   - ETL (Extract, Transform, Load) Jobs: Create ETL jobs to define how data should be extracted, transformed, and loaded into the target system. The extraction phase involves pulling data from different SAP frameworks like PLM, SRM, CRM and SCM to obtain the structured data tables MwwCcrBranches, MwwCcrBranchRunningCosts, MwwCcrBusinessTrips, MwwCcrEmployees, MwwCcrEquipmentOrderItems, and MwwCcrStrategicRegions.
2. **Data Transfer to Google Cloud Storage (GCS)**:
   - Google Cloud Storage: Use GCS as the intermediary storage for the extracted data. Once the data is extracted from SAP DWC, it is transferred to GCS using tools like Google Cloud Storage Transfer Service or SAP Data Services connectors for GCP.
   - File Formats: Store the extracted data in formats like CSV, JSON, or Parquet, which are compatible with GCP services.
3. **Data Loading into Google BigQuery**

- Google BigQuery: Load the data from GCS into Google BigQuery, a fully-managed data warehouse solution on GCP that supports fast SQL queries and real-time analytics.
- BigQuery Data Transfer Service: Use BigQuery Data Transfer Service or create custom scripts to automate the data loading process from GCS to BigQuery.
- Schema Definition: Define the schema for each table in BigQuery to match the structure of the data extracted from SAP DWC. For instance, the schema for the MwwCcrBranches table in BigQuery will include fields like BranchId, StrategicRegionId, City, PostalCode, Latitude, Longitude, and OfficeSpaces.

4. **Data Transformation and Analysis**
   - BigQuery SQL: Use BigQuery SQL to transform the data as required for analysis. This may involve joining multiple tables, aggregating data, and creating new derived fields.
   - BigQuery ML: Leverage BigQuery ML to build machine learning models directly within BigQuery using SQL syntax, enabling predictive analytics on the integrated data.
5. **Visualization and Reporting**
   - Looker Studio: Utilize Looker Studio (formerly Google Data Studio) for creating interactive dashboards and reports. Connect Looker Studio to BigQuery to visualize data insights and trends.
   - Custom Reports: Design custom reports to display key metrics and findings from the data analysis, such as average costs of branch operations, business trip patterns, and profitability assessments for potential new branch locations.

Integrating SAP Data Warehouse Cloud (DWC) with Google Cloud Platform (GCP) enables MWW to leverage advanced data analytics and visualization capabilities. By extracting data from SAP DWC, transferring it to GCS, and loading it into BigQuery, MWW can perform comprehensive analyzes and generate valuable insights to inform strategic decisions. This integration not only enhances data accessibility and analysis, but also positions MWW to stay competitive in the mobility sector by adopting state-of-the-art cloud technologies.

2.2 Project Architecture

The general design of operational systems often involves a layered architecture, including a presentation layer, an application layer, and a database layer. The SAP client/server architecture, of which SAP DWC is a part, splits tasks or workloads between service providers (servers) and service requesters (clients). This can vary depending on requirements, including monolithic applications, service-oriented architectures (SOA), and multi-tier configurations.

Our project's system architecture integrates various tools from the Google Cloud Platform (GCP) and SAP DWC, which can be represented similarly to the layered presentation of the SAP architecture.

Integration with SAP DWC, BigQuery, and Looker Studio and BigQuery ML

1. **SAP Data Warehouse Cloud (SAP DWC)**
   - Integrates data from various sources, including SAP and non-SAP systems, enabling powerful data analysis and sharing capabilities.
   - Data integration involving ETL (Extract, Transform, Load) processes are performed here to obtain the given dataset.
2. **BigQuery**
   - A fully-managed, serverless data warehouse that enables super-fast SQL queries using Google's infrastructure.
   - Data analysis and transformations are performed on the dataset after it is added to BigQuery from SAP DWC using Google Cloud Storage (GCS).
3. **Cloud Shell and API**
   - A web-based environment for managing GCP resources. It includes essential command-line tools and programming languages.
   - Used to transform required data using Python script and APIs and stored into BigQuery.
4. **Looker Studio**
   - A tool for creating reports and dashboards from BigQuery data.
   - Visualization of the processed data from BigQuery and creation of reports to better represent data and aid in decision-making.
5. **BigQuery ML**
   - Enables the creation and operationalization of machine learning models directly within BigQuery using SQL.
   - Enhances data analysis capabilities by integrating machine learning processes with the data stored in BigQuery.

This architecture, which integrates SAP DWC with BigQuery, Cloud Shell, and Looker Studio and BigQuery ML, leverages the strengths of various modern cloud-based services, creating a robust, scalable, and flexible system for data management and analytics. It offers a comprehensive solution for handling and visualizing large datasets, enhancing the overall efficiency and capability of business operations.

2.3 UML Component Diagram Notation

The system architecture is composed of several components, each playing a crucial role in the overall functionality. **Components and Their Roles**

1. SAP Data Warehouse Cloud (SAP DWC) Integrates data from various sources, including SAP systems, and provides a unified data foundation.
2. BigQuery Dataset: Represents the initial dataset creation in Google Cloud BigQuery, serving as a central data warehouse.
3. BigQuery View: Custom views created in BigQuery for specific tasks, such as data analysis and cost savings analysis.
4. Distance Matrix API: A Google Maps API used for calculating distances between locations.
5. Static Maps API: A Google Maps API used for visualizing locations on maps.
6. Cloud Shell and API: Provides a versatile environment for managing GCP resources and executing integration tasks.
7. Looker Studio: A business intelligence tool for creating reports and dashboards from BigQuery data.
8. BigQuery ML: Facilitates machine learning model creation and application within BigQuery using SQL.

Relationships Between Components

1. SAP DWC to BigQuery Dataset: Data is extracted from SAP DWC and loaded into BigQuery.
2. BigQuery Dataset to BigQuery View: Views are created on the dataset for specific analyses.
3. Distance Matrix API and Static Maps API to BigQuery View: Data from these APIs is processed and stored using Python code in BigQuery for visualization.
4. BigQuery View to Looker Studio: Views in BigQuery are used in Looker Studio for creating dashboards.
5. BigQuery ML: Utilized within BigQuery to enhance data analysis with machine learning models.

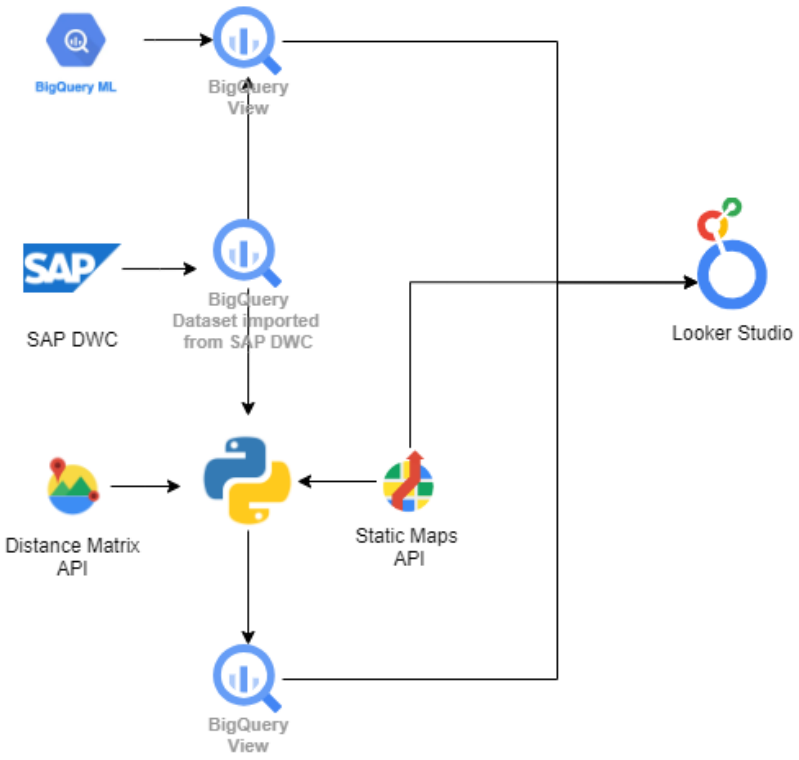

Figure 2.2: UML Diagram representing System Architecture

2.4 Justification of Architecture Design Using BSP Diagrams

BSP diagrams model the architecture in terms of business functions and the data or applications they require, ensuring the architecture aligns with business needs. **Business Functions**

1. Data Integration: Integrating data from various sources (SAP DWC, APIs) into a central data warehouse (BigQuery).
2. Data Analysis: Analyzing data through views in BigQuery and using Looker Studio for creating reports.
3. Machine Learning: Utilizing BigQuery ML for advanced data analysis and prediction models.
4. Visualization and Reporting: Creating interactive dashboards with Looker Studio.

Processes

1. Data Extraction and Loading: Extract data from SAP DWC and load it into BigQuery.
2. Data Transformation: Use APIs to enhance data with additional information and store it in BigQuery.

3. Data Analysis and Machine Learning: Perform data analysis and machine learning using BigQuery and BigQuery ML.

4. Data Visualization: Use Looker Studio to create and deploy dashboards.

Components

1. Data Sources: SAP DWC, Distance Matrix API, Static Maps API.

2. Data Warehouse: BigQuery.

3. Management and Automation Tools: Cloud Shell and API.

4. Visualization Tool: Looker Studio.

5. Machine Learning: BigQuery ML.

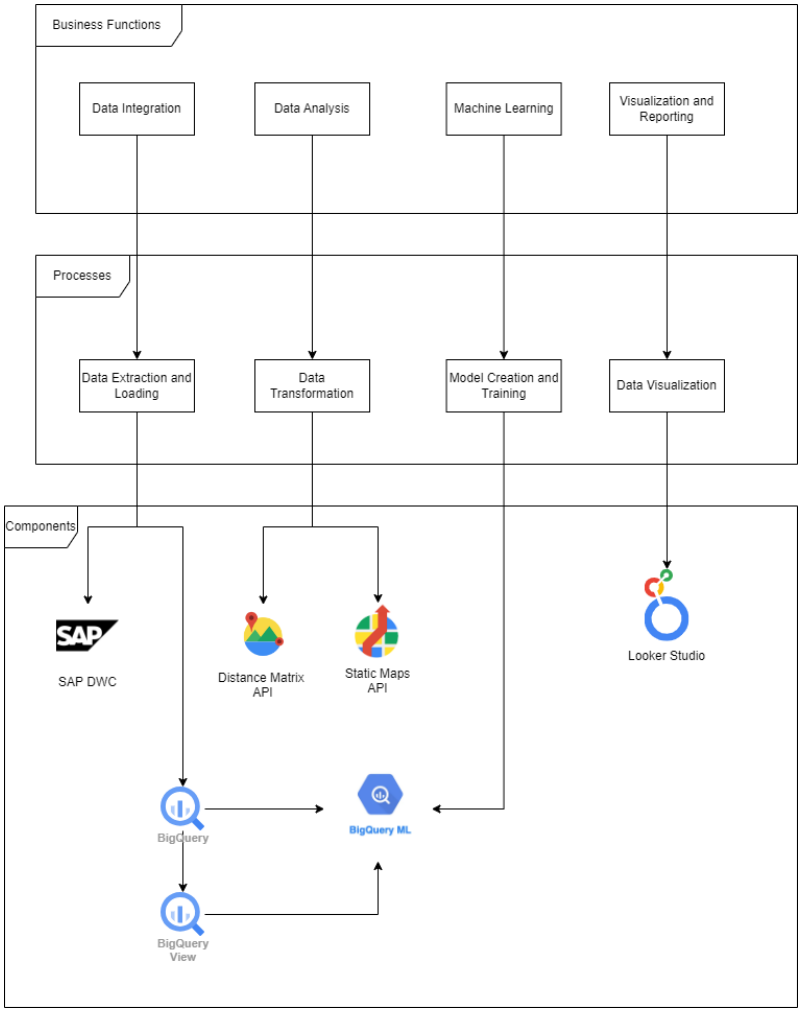

Figure 2.3: UML Diagram representing System Architecture

2.5 Understanding and Highlighting Cloud Patterns

Cloud patterns provide reusable solutions to common problems in cloud-based architecture. Our Project utilizes several cloud computing patterns to achieve its functionality. Here's a breakdown of the cloud patterns

Cloud Offering Patterns

- Implementation:
  - Storage Offering: BigQuery is used as a central data warehouse for storing and managing large datasets.
  - Communication Offering: These APIs provide external data and services necessary for the application.

Cloud Integration Patterns

- Implementation:
  - It involves transforming and organizing data from the BigQuery dataset for specific analytical purposes.
  - The data from APIs is integrated and processed within the BigQuery environment.

Cloud Application Architecture Patterns

- Implementation:
  - Cloud Application Components: Custom views are created to perform specific tasks such as data analysis and cost savings.
  - Presentation: Looker Studio is used to create reports and dashboards for data visualization.

2.6 Summary

By describing the architecture using UML component diagrams, justifying the design with BSP diagrams, and understanding the cloud patterns involved, we have ensured that our project is well-structured, comprehensible, and aligns with best practices in modern system design. This approach not only makes our solution reproducible but also highlights its composability.

3 Data Preparation using BigQuery

In this chapter, we give an overview of the data provided for analysis, the different queries we used to clean and transform the data, and the tool used to visualize it. The goal of this chapter is to evaluate the average costs for a branch office, which solve task 1 of the given scenario.

3.1 Description of the Provided Datasets

The following six datasets are provided from the SAP Data Warehouse Cloud (DWC):

- MwwCcrBranches: Contains information on the branch offices in each region (see Figure 3.1).

Row	BranchId	StrategicRegionId	City	PostalCode	Latitude	Longitude	OfficeSpaces
1	B100001	GER100001	Frankenroda	99826	51.0787394	10.3351003	22
2	B100002	GER100001	Obermarchtal	89611	48.2337188	9.5698554	21
3	B100003	GER100001	Artern	6556	51.3667759	11.2919115	22

Figure 3.1: MwwCcrBranches

- MwwCcrBranchRunningCosts: All costs associated with the operation of the branch offices (see Figure 3.2).

Row	FinanceProcessId	BranchId	Date	ProcessType	Costs	Currency
1	FP100003	B100001	2000-01-01	Water bill	507.12	EUR
2	FP100073	B100001	2000-02-01	Water bill	486.97	EUR
3	FP100132	B100001	2000-03-01	Water bill	306.97	EUR

Figure 3.2: MwwCcrBranchRunningCosts

- MwwCcrBusinessTrips: Historical records of business trips made by employees (see Figure 3.3).

Row	BusinessTripId	StartDate	EndDate	EmployeeId	PostalCode	City	OvernightStays	HotelCosts	TravelCosts	Currency
1	BT104986	1998-02-08	1998-02-08	E107718	55559	Bretzenheim	0	0.0	0.0	EUR
2	BT105058	1998-02-25	1998-02-25	E107718	55559	Bretzenheim	0	0.0	0.0	EUR
3	BT106442	1998-09-20	1998-09-20	E107718	55559	Bretzenheim	0	0.0	0.0	EUR

Figure 3.3: MwwCcrBusinessTrips

- MwwCcrEmployees: Details of currently employed persons, including their branch assignments (see Figure 3.4).

Row	EmployeeId	BranchId	Position	HiredOn	Salary	Currency
1	E109013	B100001	Manager	2019-03-29	127200.0	EUR
2	E107935	B100002	Manager	2001-07-11	214901.72	EUR
3	E108999	B100003	Manager	2018-07-01	134832.0	EUR

Figure 3.4: MwwCcrEmployees

- MwwCcrEquipmentOrderItems: Orders placed by employees for office supplies and equipment (see Figure 3.5).

Row	EquipmentOrderId	Date	ProductName	UnitOfMeasure	Quantity	CostPerItem	Costs	Currency	EmployeeId
1	E0106311	1995-03-17	Printer cartridge (black)	ST	1	7.5	7.5	EUR	E107543
2	E0106453	1995-09-29	Printer cartridge (black)	ST	1	7.5	7.5	EUR	E107543
3	E0108712	1997-05-22	Printer cartridge (black)	ST	1	7.5	7.5	EUR	E107543

Figure 3.5: MwwCcrEquipmentOrderItems

- MwwCcrStrategicRegions: divides Germany into regions based on postal codes (see Figure 3.6).

Row	StrategicRegionId	CountryKey	CountryName	PostalCodeRang	PostalCodeRang	PostalCodeRang
1	GER100001	DE	Germany	9	0	null
2	GER100002	DE	Germany	1	2	null
3	GER100004	DE	Germany	6	7	null

Figure 3.6: MwwCcrStrategicRegions

3.2 Import the Data to Google Cloud Platform

The data for this analysis were provided in the form of CSV files. We imported these CSV files into Google Cloud Platform (GCP) using Google Cloud Storage and BigQuery, so that we can access the data, analyze it, and do a visualization. We performed the following steps using the BigQuery Web UI:

- open BigQuery web UI (see Figure 3.7)
- In the Explorer pane, we created a new dataset (see Figure 3.8)
- the following parameters were entered: ID: branches, Location type: Region, Data location: europe-west3 (see Figure 3.9)
- for each provided csv file, we created a table with the same name (see Figure 3.10)

3.3 Views

Task 1: The average costs for a branch office must be evaluated. Consider all given data regarding rent, personnel, maintenance costs etc.
We created several views in BigQuery to streamline the data analysis process and enhance query performance. We took a look on the given dataset, so that we we can easily extract important information. The following views are created:

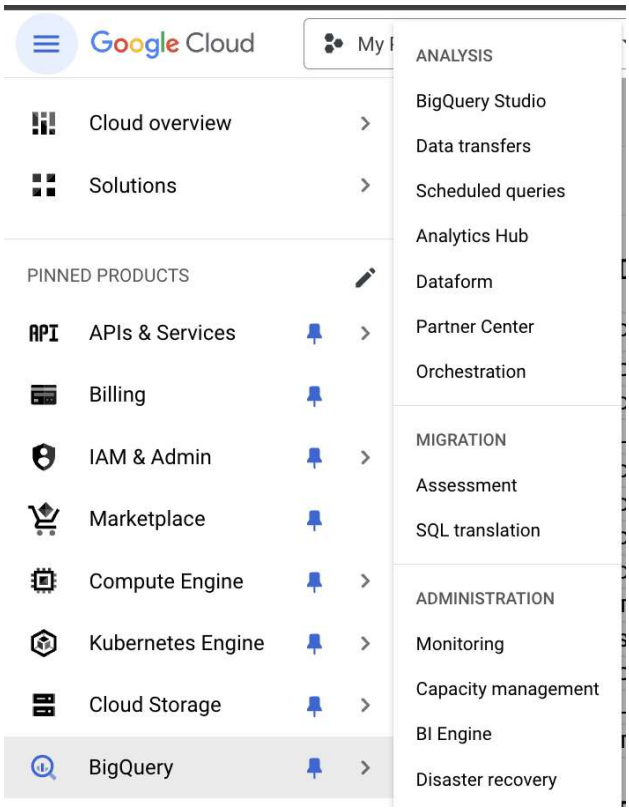

Figure 3.7: BigQuery

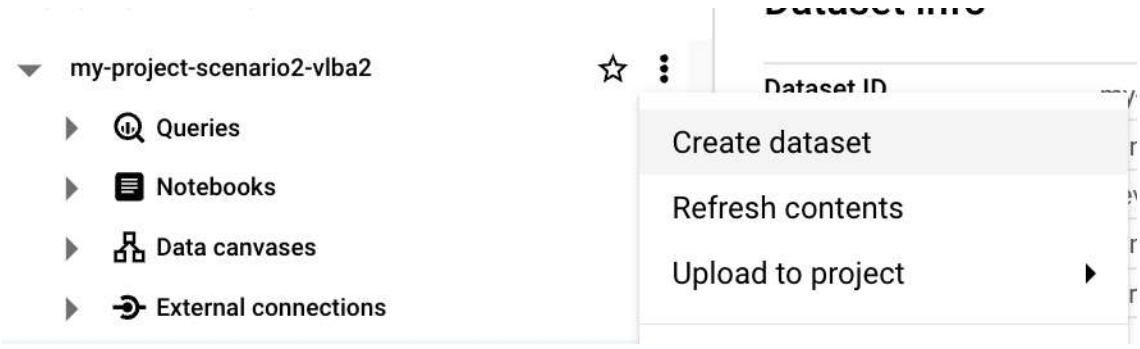

Figure 3.8: create a dataset

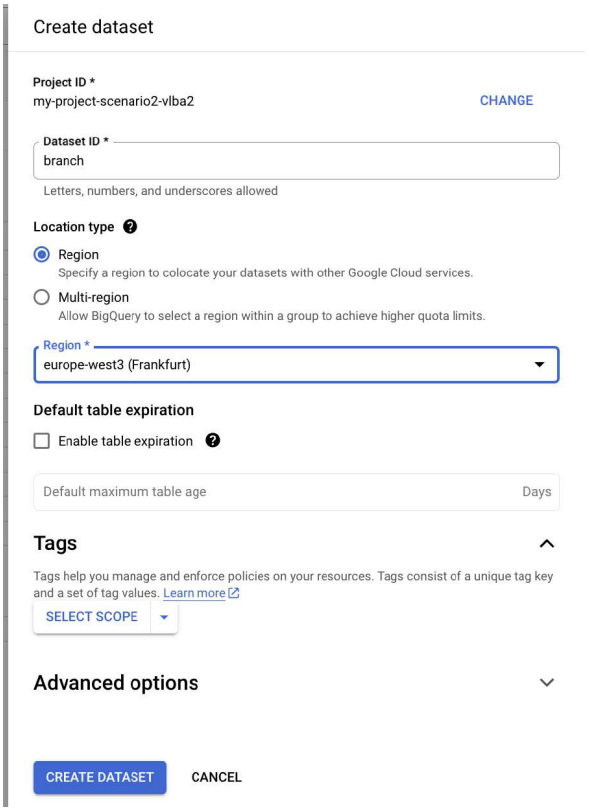

Figure 3.9: dataset details

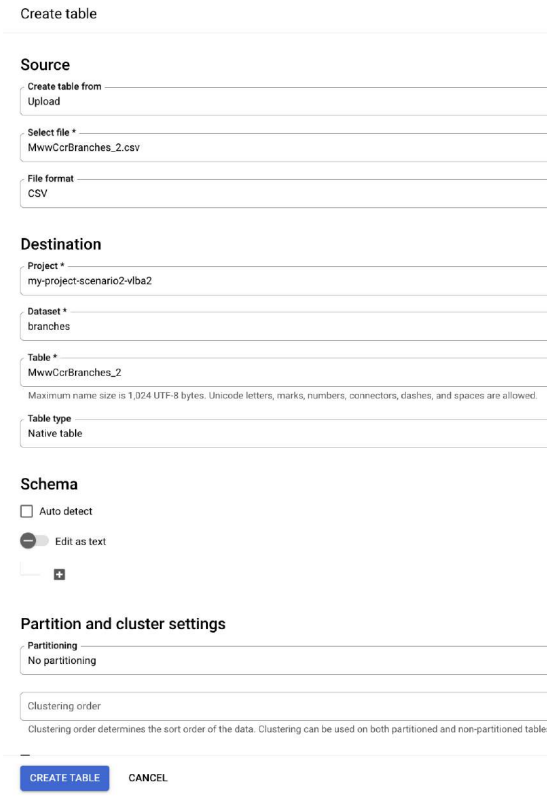

Figure 3.10: create a table

- Cost types: a view to list types of costs in MwwCcrBranchRunningCosts, so that the board members has an overview, which types of costs are included in a branch.

  Following costs are identified: Water bill, Electrician, Office rent, Heating costs, Minor repairs, Vehicle repair, Heating engineer, Plumbing service, Construction work, Technical support, Ventilation repair, Window replacement, Electricity prepayment, Smoke detector control, Electricity reimbursement, Electricity additional payment.

```
SELECT
    ProcessType AS typesOfCosts
FROM
    `my-project-scenario2-vlba2.branches.
       MwwCcrBranchRunningCosts_2`
GROUP BY
    ProcessType
```

- branch running costs categorization and costs: categorizes and averages the running costs of each branch, providing a clear understanding of the financial distribution across different expense categories.

```
SELECT
    BranchId,
    CASE
        WHEN ProcessType IN ('Office rent') THEN '
           Operational Costs'
        WHEN ProcessType IN ('Electrician', 'Minor repairs
           ', 'Vehicle repair', 'Heating engineer', '
           Plumbing service', 'Construction work', '
           Technical support', 'Ventilation repair', '
           Window replacement', 'Smoke detector control')
           THEN 'Maintenance Costs'
        WHEN ProcessType IN ('Water bill', 'Heating costs'
           , 'Electricity prepayment', 'Electricity
           reimbursement', 'Electricity additional payment
           ') THEN 'Utility Costs'
        ELSE 'Other'
    END AS CostCategory,
    SUM(CAST(Costs AS FLOAT64)) AS AvgTotalCosts
FROM
    `my-project-scenario2-vlba2.branches.
       MwwCcrBranchRunningCosts_2`
GROUP BY
    BranchId, CostCategory
ORDER BY
    BranchId, CostCategory;
```

- Types of product to order: As mentioned in MwwCcrEquipmentOrderItems, employees order many items for their work. We created the following view so that the board members has an overview, which types of products are ordered in a branch.

  Following products are ordered from employees: Printer cartridge (black), Pen, Paper clips (package), Pencil, Printer paper, Notepad, Printer cartridge (color).

```
SELECT
```

BranchId	CostCategory	TotalCosts
B100001	Maintenance Costs	287027.75999999995
B100001	Operational Costs	2642848.88
B100001	Utility Costs	669746.26000000013

Figure 3.11: branch running costs categorization and costs

```
    ProductName AS Products
FROM
    `my-project-scenario2-vlba2.branches.
       MwwCcrEquipmentOrderItems_2`
GROUP BY
    ProductName;
```

- branch equipment order items categorization costs: categorizes equipment order items by type (Printing Supplies, Office Supplies, and Other) and summarizes the total quantity and costs per branch.

```
SELECT
    e.BranchId,
    CASE
        WHEN o.ProductName IN ('Printer cartridge (black)'
           , 'Printer cartridge (color)', 'Printer paper')
            THEN 'Printing Supplies'
        WHEN o.ProductName IN ('Pen', 'Paper clips (
           package)', 'Pencil', 'Notepad') THEN 'Office
           Supplies'
        ELSE 'Other'
    END AS ProductCategory,
    SUM(o.Quantity) AS TotalQuantity,
    SUM(CAST(Costs AS FLOAT64)) AS SumTotalCost
FROM
    `my-project-scenario2-vlba2.branches.
       MwwCcrEquipmentOrderItems_2` AS o
JOIN
    `my-project-scenario2-vlba2.branches.MwwCcrEmployees_2
       ` AS e
ON
    o.EmployeeId = e.EmployeeId
GROUP BY
    e.BranchId, ProductCategory
ORDER BY
    e.BranchId, ProductCategory;
```

- Salary per employee: helps in decision-making for HR and management.

```
WITH Years AS (
    SELECT year
    FROM UNNEST(GENERATE_ARRAY(1996, 2021)) AS year
)
SELECT
    BranchID,
```

BranchId	ProductCategory	TotalQuantity	SumTotalCost
B100001	Office Supplies	5261	9349.6000000000513
B100001	Printing Supplies	1212	8287.599999999924
B100002	Office Supplies	5491	9881.8100000000359

Figure 3.12: branch equipment order items categorization costs

```
    EmployeeId,
    HiredOn,
    year AS Year,
    Salary
FROM
    my-project-scenario2-vlba2.branches.MwwCcrEmployees_2,
    Years
WHERE
    EXTRACT(YEAR FROM HiredOn) <= year
ORDER BY
    BranchID,
    EmployeeId,
    Year
```

BranchID	EmployeeId	HiredOn	Year	Salary
B100001	E107555	1995-07-15	1996	136507.4
B100001	E107555	1995-07-15	1997	136507.4
B100001	E107555	1995-07-15	1998	136507.4

Figure 3.13: Salary per employee

- total number of employees and salaries per branch:

```
SELECT
  BranchID,
  COUNT(DISTINCT EmployeeId) AS EmployeeCount,
  SUM(Salary) AS TotalSalary
FROM
  `my-project-scenario2-vlba2.branches.SaleryPerEmployee`
GROUP BY
  BranchID
ORDER BY
  BranchID;
```

BranchID	EmployeeCount	TotalSalary
B100001	18	26909737.96999995
B100002	17	29859523.129999895
B100003	18	28766415.479999911

Figure 3.14: Salary per employee

- target regions and number of employees

```
WITH SelectedEmployees AS (
    SELECT
        EmployeeId
    FROM
        `my-project-scenario2-vlba2.branches.
           MwwCcrBusinessTrips_2`
    WHERE
        HotelCosts > 0
),
EmployeeBranch AS (
    SELECT
        se.EmployeeId,
        e.BranchId
    FROM
        SelectedEmployees se
    INNER JOIN
        `my-project-scenario2-vlba2.branches.
           MwwCcrEmployees_2` e ON se.EmployeeId = e.
           EmployeeId
),
EmployeeRegion AS (
    SELECT
        eb.EmployeeId,
        eb.BranchId,
        b.StrategicRegionId
    FROM
        EmployeeBranch eb
    INNER JOIN
        `my-project-scenario2-vlba2.branches.
           MwwCcrBranches_2` b ON eb.BranchId = b.BranchId
)
SELECT
    StrategicRegionId,
    COUNT(DISTINCT EmployeeId) AS NumberOfEmployees
FROM
    EmployeeRegion
GROUP BY
    StrategicRegionId
ORDER BY
    NumberOfEmployees DESC;
```

StrategicRegionId	NumberOfEmployees
GER100003	46

Figure 3.15: target regions

- target branches: as we can see B100007, B100008, B100009 are the target branches.

```
SELECT
  BranchId
FROM
  `my-project-scenario2-vlba2.branches.MwwCcrBranches_2`
WHERE
```

```
  StrategicRegionId = 'GER100003'
```

- total costs of a branch with and without trips and hotel staying

```
WITH running_costs AS (
    SELECT
        BranchId,
        SUM(CAST(Costs AS FLOAT64)) AS SumRunningCost
    FROM
        `my-project-scenario2-vlba2.branches.
           MwwCcrBranchRunningCosts_2`
    GROUP BY
        BranchId
),
business_trip_costs AS (
    SELECT
        e.BranchId,
        SUM(CAST(bt.TravelCosts AS FLOAT64)) AS
           SumTravelCosts,
        SUM(CAST(bt.HotelCosts AS FLOAT64)) AS
           SumHotelCosts
    FROM
        `my-project-scenario2-vlba2.branches.
           MwwCcrBusinessTrips_2` AS bt
    JOIN
        `my-project-scenario2-vlba2.branches.
           MwwCcrEmployees_2` AS e
    ON
        bt.EmployeeId = e.EmployeeId
    GROUP BY
        e.BranchId
),
salary_costs AS (
    SELECT
        BranchId,
        TotalSalary
    FROM
        `my-project-scenario2-vlba2.branches.
           total_employee_and_salaries_per_branch`
),
equipment_order_costs AS (
    SELECT
        e.BranchId,
        SUM(Costs) AS SumEquipmentOrderCost
    FROM
        `my-project-scenario2-vlba2.branches.
           MwwCcrEquipmentOrderItems_2` AS eoi
    JOIN
        `my-project-scenario2-vlba2.branches.
           MwwCcrEmployees_2` AS e
    ON
        eoi.EmployeeId = e.EmployeeId
    GROUP BY
        e.BranchId
)
SELECT
    b.BranchId,
```

```
46      COALESCE(rc.SumRunningCost, 0) AS SumRunningCost,
47      COALESCE(btc.SumTravelCosts, 0) AS SumTravelCosts,
48      COALESCE(btc.SumHotelCosts, 0) AS SumHotelCosts,
49      COALESCE(eoc.SumEquipmentOrderCost, 0) AS
           SumEquipmentOrderCost,
50      TotalSalary,
51      (COALESCE(rc.SumRunningCost, 0) + COALESCE(btc.
           SumTravelCosts, 0) + COALESCE(btc.SumHotelCosts, 0)
            + COALESCE(eoc.SumEquipmentOrderCost, 0) +
           TotalSalary) AS TotalCostWithTrips,
52      (COALESCE(rc.SumRunningCost, 0) + COALESCE(eoc.
           SumEquipmentOrderCost, 0) + TotalSalary) AS
           TotalCostWithoutTrips
53  FROM
54      `my-project-scenario2-vlba2.branches.MwwCcrBranches_2`
            AS b
55  LEFT JOIN
56      running_costs AS rc ON b.BranchId = rc.BranchId
57  LEFT JOIN
58      business_trip_costs AS btc ON b.BranchId = btc.
           BranchId
59  LEFT JOIN
60      equipment_order_costs AS eoc ON b.BranchId = eoc.
           BranchId
61  LEFT JOIN
62      salary_costs AS sc ON b.BranchId = sc.BranchId
63  ORDER BY
64      b.BranchId;
```

BranchId	SumRunningCost	SumTravelCosts	SumHotelCosts	SumEquipmentOrderCost	TotalSalary	TotalCostWithTrips	TotalCostWithoutTrips
B100001	3599622.9000000013	0.0	0.0	17637.199999999921	26909737.96999995	30526998.069999952	30526998.069999952
B100002	3499966.820000005	0.0	0.0	18281.009999999951	29859523.129999895	33377770.9599999	33377770.9599999
B100003	2290298.620000002	0.0	0.0	18375.309999999987	28766415.479999911	31075089.409999914	31075089.409999914
B100004	2437565.4700000021	0.0	0.0	23021.690000000053	38293273.839999832	40753860.999999836	40753860.999999836

Figure 3.16: total costs of a branch with and without trips and hotel staying

- total costs of a branch with trips per year

```
1   WITH running_costs AS (
2       SELECT
3           BranchId,
4           EXTRACT(YEAR FROM DATE) AS Year,
5           SUM(CAST(Costs AS FLOAT64)) AS SUMRunningCost
6       FROM
7           `my-project-scenario2-vlba2.branches.
               MwwCcrBranchRunningCosts_2`
8       GROUP BY
9           BranchId, Year
10  ),
11  business_trip_costs AS (
12      SELECT
13          e.BranchId,
14          EXTRACT(YEAR FROM bt.StartDate) AS Year,
15          SUM(CAST(bt.TravelCosts AS FLOAT64)) AS
               SUMTravelCosts,
```

```
16          SUM(CAST(bt.HotelCosts AS FLOAT64)) AS
               SUMHotelCosts
17      FROM
18          `my-project-scenario2-vlba2.branches.
               MwwCcrBusinessTrips_2` AS bt
19      JOIN
20          `my-project-scenario2-vlba2.branches.
               MwwCcrEmployees_2` AS e
21      ON
22          bt.EmployeeId = e.EmployeeId
23      WHERE
24          bt.HotelCosts > 0.0 OR bt.TravelCosts > 0.0
25      GROUP BY
26          e.BranchId, Year
27  ),
28  equipment_order_costs AS (
29      SELECT
30          e.BranchId,
31          EXTRACT(YEAR FROM eoi.Date) AS Year,
32          SUM(Costs) AS SUMEquipmentOrderCost
33      FROM
34          `my-project-scenario2-vlba2.branches.
               MwwCcrEquipmentOrderItems_2` AS eoi
35      JOIN
36          `my-project-scenario2-vlba2.branches.
               MwwCcrEmployees_2` AS e
37      ON
38          eoi.EmployeeId = e.EmployeeId
39      GROUP BY
40          e.BranchId, Year
41  ),
42  salary_costs AS (
43      SELECT
44          BranchId,
45          Year,
46          SUM(Salary) AS SUMSalaryCosts
47      FROM
48          `my-project-scenario2-vlba2.branches.
               SaleryPerEmployee`
49      GROUP BY
50          BranchId, Year
51  )
52  SELECT
53      b.BranchId,
54      COALESCE(rc.Year, btc.Year, eoc.Year, sc.Year) AS Year
           ,
55      COALESCE(rc.SUMRunningCost, 0) AS SUMRunningCost,
56      COALESCE(btc.SUMTravelCosts, 0) AS SUMTravelCosts,
57      COALESCE(btc.SUMHotelCosts, 0) AS SUMHotelCosts,
58      COALESCE(eoc.SUMEquipmentOrderCost, 0) AS
           SUMEquipmentOrderCost,
59      COALESCE(sc.SUMSalaryCosts, 0) AS SUMSalaryCosts,
60      (COALESCE(rc.SUMRunningCost, 0) + COALESCE(btc.
           SUMTravelCosts, 0) + COALESCE(btc.SUMHotelCosts, 0)
            + COALESCE(eoc.SUMEquipmentOrderCost, 0)+ COALESCE
           (sc.SUMSalaryCosts, 0)) AS TotalCost
61  FROM
```

```
    `my-project-scenario2-vlba2.branches.MwwCcrBranches_2`
        AS b
LEFT JOIN
    running_costs AS rc ON b.BranchId = rc.BranchId
LEFT JOIN
    business_trip_costs AS btc ON b.BranchId = btc.
       BranchId AND rc.Year = btc.Year
LEFT JOIN
    equipment_order_costs AS eoc ON b.BranchId = eoc.
       BranchId AND rc.Year = eoc.Year
LEFT JOIN
    salary_costs AS sc ON b.BranchId = sc.BranchId AND
       COALESCE(rc.Year, btc.Year, eoc.Year) = sc.Year
ORDER BY
    b.BranchId, Year;
```

BranchId	Year	SUMRunningCos	SUMTravelCosts	SUMHotelCosts	SUMEquipmentOrderCos	SUMSalaryCosts	TotalCost
B100001	2000	134879.24	0.0	0.0	232.39	386779.06999999995	521890.69999999995
B100001	2001	131060.530...	0.0	0.0	153.99000000000004	501393.31999999995	632607.84
B100001	2002	144041.310...	0.0	0.0	360.26999999999992	501393.31999999995	645794.9

Figure 3.17: total costs of a branch with trips per year

- employee salaries per year: the provided data MwwCcrEmployees2 provides the salary of an employee who was hired on that particular year. So this new view will help us when calculating the total costs of salaries per branch.

```
WITH Years AS (
    SELECT year
    FROM UNNEST(GENERATE_ARRAY(1996, 2021)) AS year
)

SELECT
    BranchID,
    EmployeeId,
    HiredOn,
    year AS Year,
    Salary
FROM
    my-project-scenario2-vlba2.branches.MwwCcrEmployees_2,
    Years
WHERE
    EXTRACT(YEAR FROM HiredOn) <= year
ORDER BY
    BranchID,
    EmployeeId,
    Year
```

- average cost of a branch

```
SELECT BranchId, AVG(TotalCost) AS AverageCost
 FROM `my-project-scenario2-vlba2.branches.
    ToTALCOSTSOFABRANCHPERYEAR`
GROUP BY BranchId
```

BranchID	EmployeeId	HiredOn	Year	Salary
B100001	E107555	1995-07-15	1996	136507.4
B100001	E107555	1995-07-15	1997	136507.4
B100001	E107555	1995-07-15	1998	136507.4

Figure 3.18: employee salaries per year

BranchId	AverageCost
B100001	1351033.1304545454
B100002	1470326.3504545458
B100003	1376623.8768181817

Figure 3.19: average cost of a branch

- origin and destination for each business trip with driving or hotel staying: this view will be used in visualizing the trips done by employees in task 2

```
SELECT
    bt.BusinessTripId,
    e.EmployeeId,
    e.BranchId AS OriginBranchId,
    b.City AS OriginCity,
    bt.City AS DestinationCity,
    bt.PostalCode AS DestinationPostalCode,
    bt.HotelCosts,
    bt.TravelCosts
FROM
    `my-project-scenario2-vlba2.branches.
       MwwCcrBusinessTrips_2` AS bt
JOIN
    `my-project-scenario2-vlba2.branches.MwwCcrEmployees_2
       ` AS e
ON
    bt.EmployeeId = e.EmployeeId
JOIN
    `my-project-scenario2-vlba2.branches.MwwCcrBranches_2`
        AS b
ON
    e.BranchId = b.BranchId
WHERE
    bt.HotelCosts > 0.0 OR bt.TravelCosts > 0.0
ORDER BY
    bt.BusinessTripId;
```

BT176453	E108496	B100009	Süchteln	Altenbamberg	55585	0.0	33.46
BT176460	E108960	B100009	Süchteln	Garbsen	30823	466.55	29.81
BT176468	E108740	B100009	Süchteln	Garbsen	30823	1596.24	31.52

Figure 3.20: origin and destination for each business trip with driving or hotel staying

- number of trips per city: this view helps us to determine the most visited cities by

employees.

```
SELECT
    City,
    COUNT(*) AS Number_of_trips
FROM
    `my-project-scenario2-vlba2.branches.
       MwwCcrBusinessTrips_2`
WHERE
    HotelCosts > 0
GROUP BY
    City
ORDER BY
    Number_of_trips DESC;
```

City	Number_of_trips
Garbsen	410
Hannover	196
Köln	19
Essen	10

Figure 3.21: number of trips per city

These views will allow us to solve the first task (Evaluation of the average costs for a branch office), and also give an overview to the board members about the costs included in a branch.

3.4 Data Visualization

We use GCP Looker Studio to visualize the average annual expenditures associated with each branch, along with comprehensive breakdowns of particular costs like salaries and business travel expenses. Furthermore, in order to comprehend the distribution of human costs among branches, we also visualize the wage expenses. We also provide expenditure visualizations for corporate travel. Together, these visualizations help us spot cost trends and make informed decisions on whether it would be feasible to open a new branch in a new strategic location.

- **Average running costs a Branch (2000-2021)**

  The figure below shows (See Figure 3.22) the average running costs of the 15 branches, including operational, maintenance, and utility costs. It is evident from the graph that Branch B100003 has the lowest expense and Branch B100001 the highest. Because branches B100007, B100008, and B100009 represent our key region, they are of great interest. The graph Figure 3.23 shows the running expenses for these branches. This emphasis makes it possible to analyse costs in this crucial area in more detail.

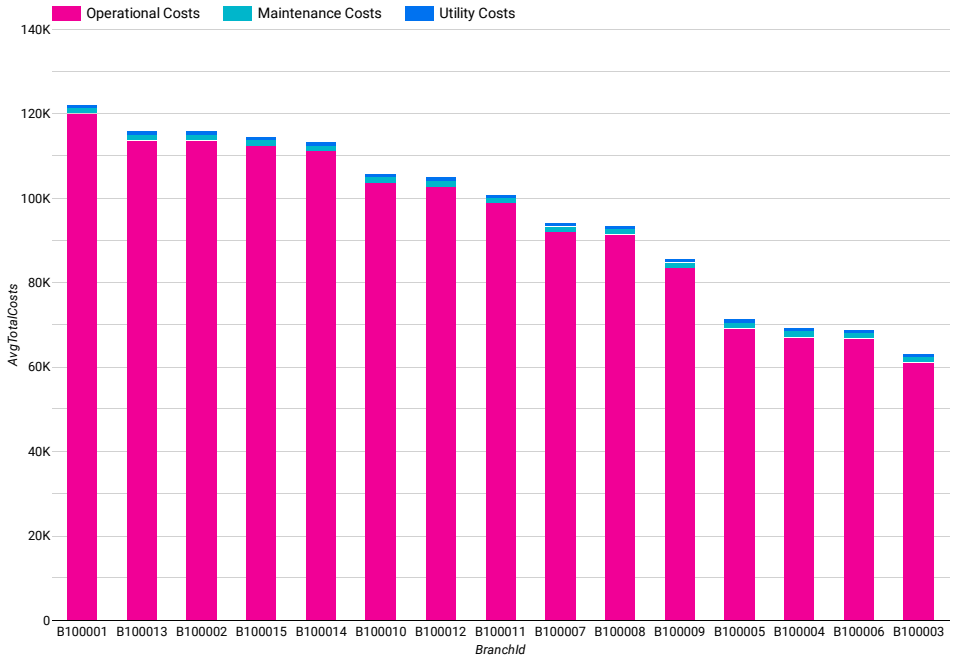

Figure 3.22: Average cost of Running a Branch (2000-2021)

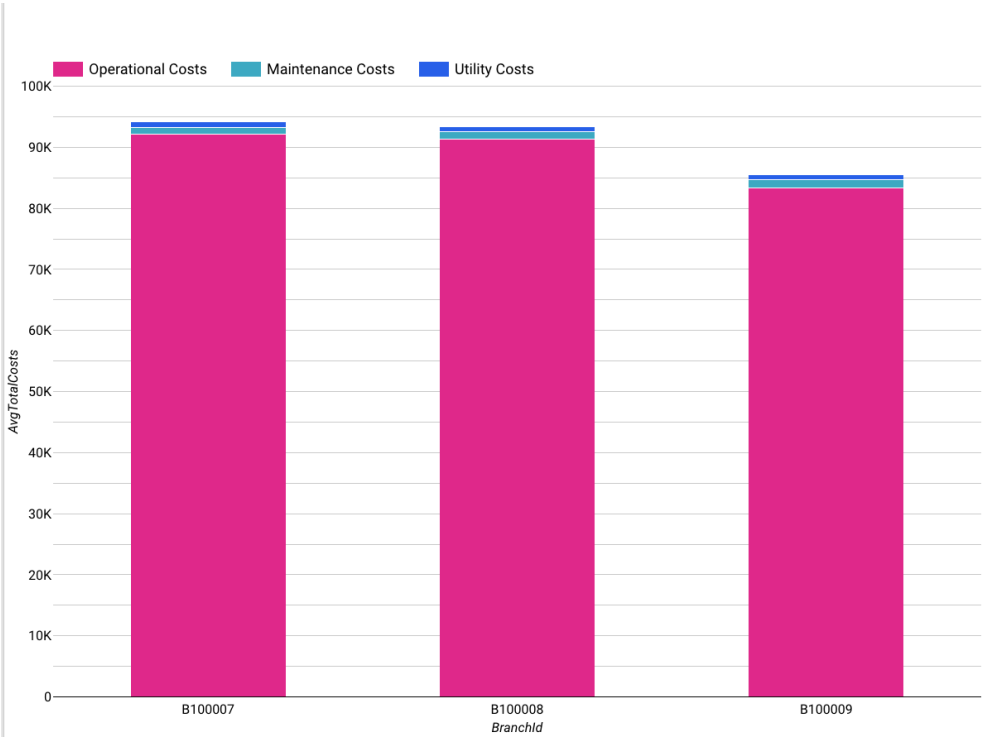

Figure 3.23: Average cost of Running a Branch in the selected Strategic Region (2000-2021)

- **Total Salaries of Different Branches for the year 2021**

  The figure below shows (see Figure 3.24) the total salaries of the employees in each branch for the year 2021. Where B100004 has the highest total salary and B100012

has the lowest total salary. This visualization helps to understand the distribution of human resource costs across different branches and aids in financial planning and decision-making.

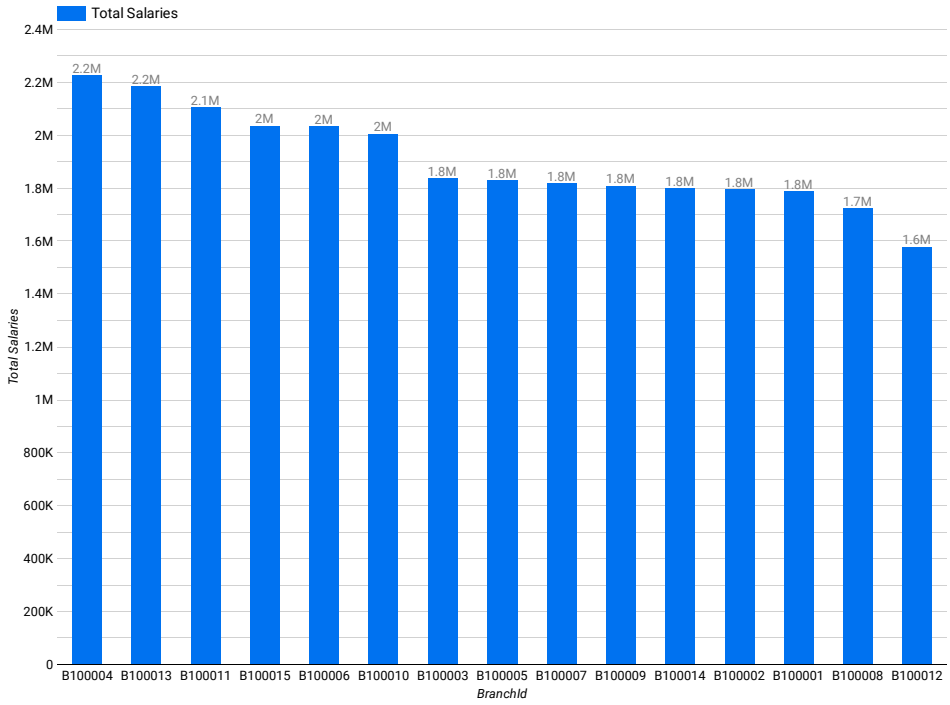

Figure 3.24: Total Salaries by Branch for the year 2021

- **Breakdown of average running costs of a Branch**

  The figure below shows (see Figure 3.25) the breakdown of the average running costs of a branch. Rent is the largest expense, constituting 68.6% of the total costs. This highlights the significance of rent in the overall cost structure.14.5% constituting Electricity bill and the remaining consists of Water Bill, Heating Costs, Construction Work, Plumbing Service, Vehicle Repair, Ventilation Repair, Heating Engineer and Other Costs.

- **Average total cost of Branch**

  The figure below illustrates (see Figure 3.26) the average total cost of running a branch, encompassing running costs, travel expenses, hotel costs, equipment order costs, and salaries. Of the two branches, Branch B100012 has the lowest average total expense and Branch B100013 the highest average total expense. The second graph (See Figure 3.27), which shows the overall operating costs for our strategically located branches B100007, B100008, and B100009, is very noteworthy. This emphasis makes it possible to analyse costs in this crucial area in more detail.

- **Yearwise Average total cost of branches 2007-2021**

  The figure below illustrates (see Figure 3.28) the average total cost of running a

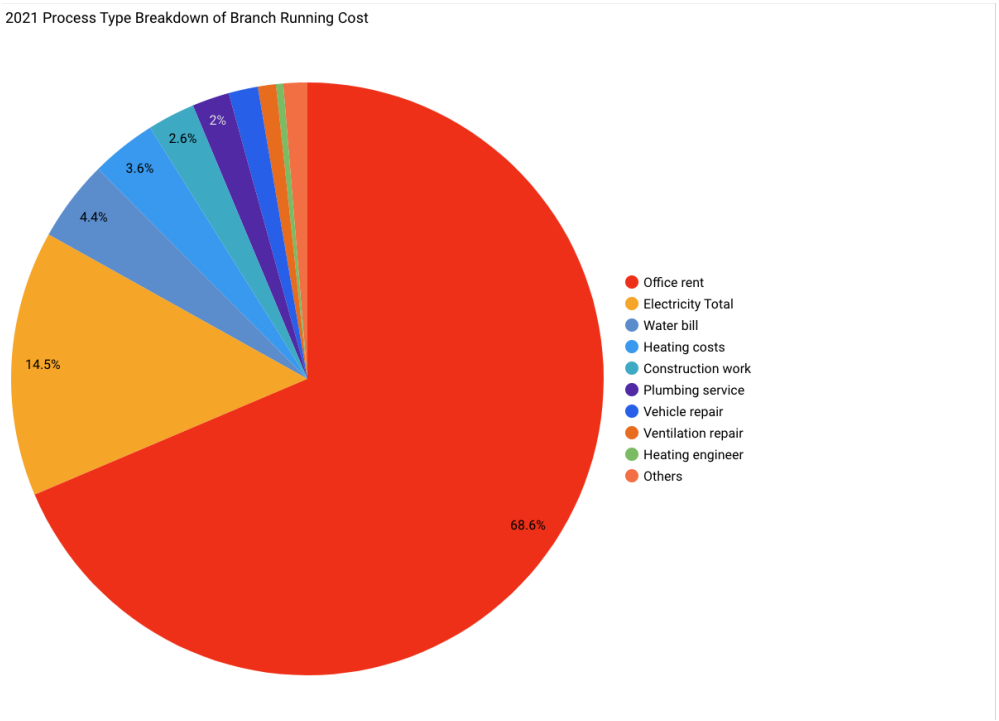

Figure 3.25: Breakdown of average running costs of a Branch

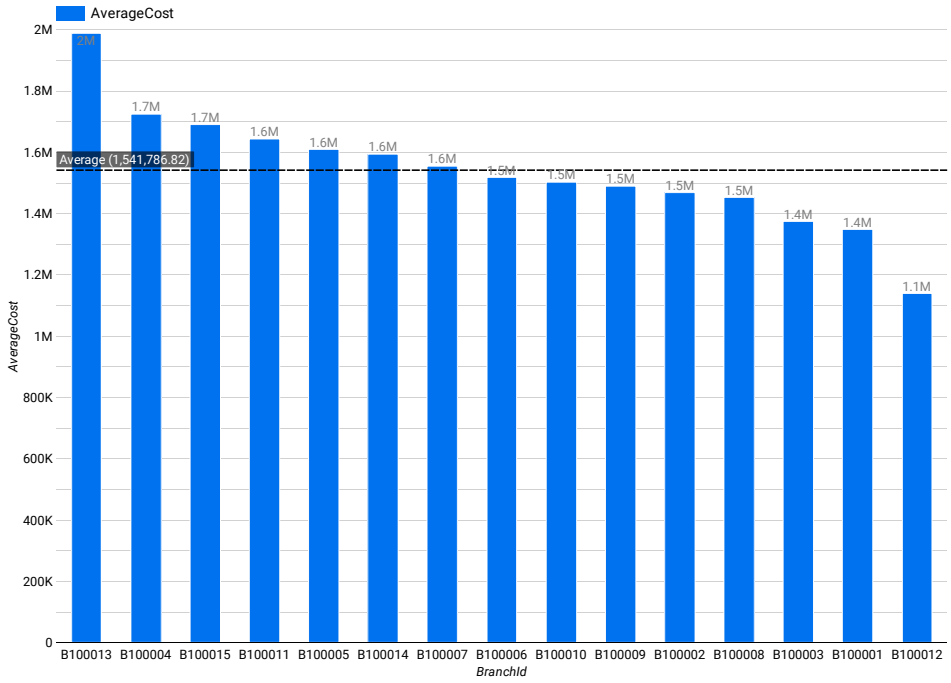

Figure 3.26: Average total cost of all the branches

branch, encompassing running costs, travel expenses, hotel costs, equipment order costs, and salaries. Of the two branches, Branch B100012 has the lowest average total expense and Branch B100013 the highest average total expense.

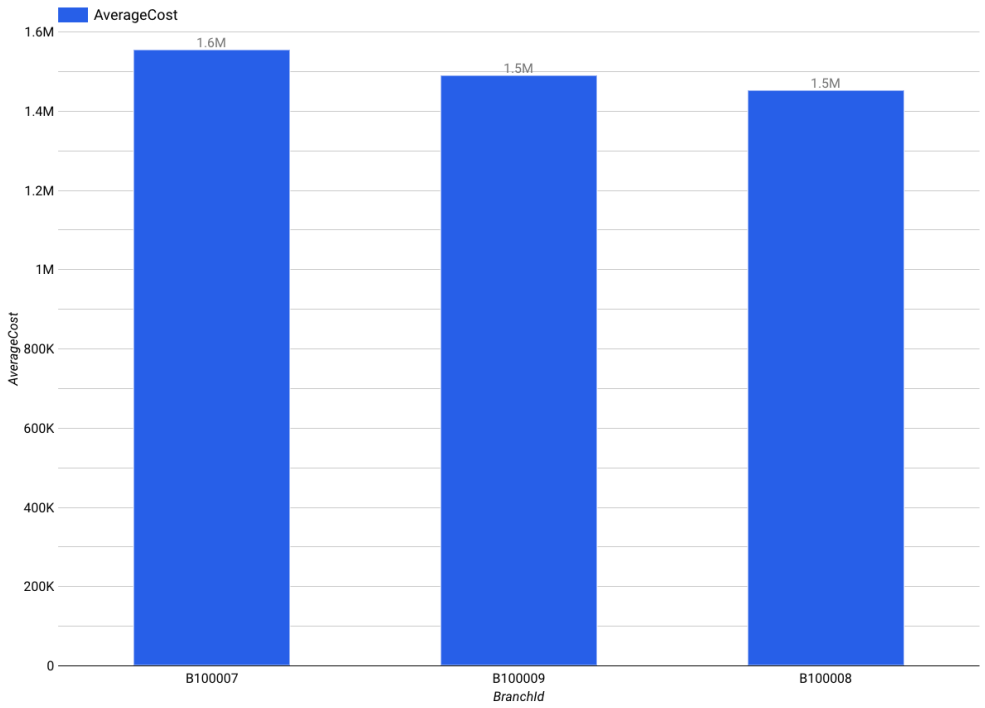

Figure 3.27: Average total cost of selected branches

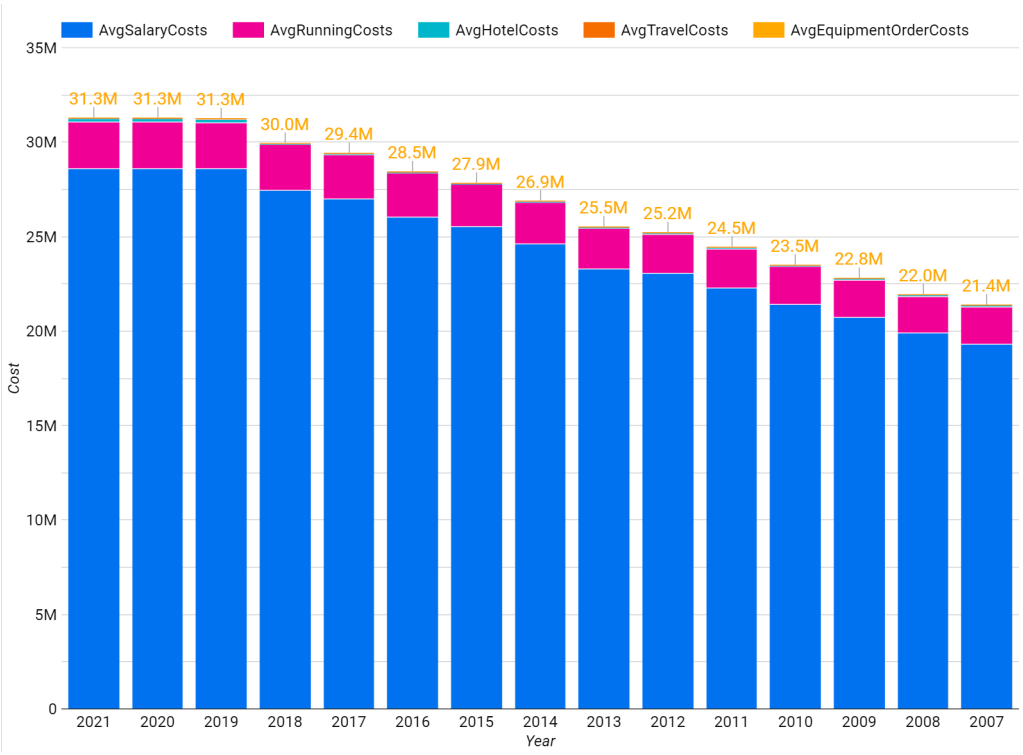

Figure 3.28: Year wise average cost of all branches (2007-2021)

- **Travel and Hotel Cost**

  The figure below illustrates (see Figure 3.29) the total travel and hotel costs incurred

by the three branches, B100007, B100008, and B100009, over the past year. The graph highlights that Branch B10009 had the highest expenses related to travel and accommodation, while Branch B10007 incurred the lowest costs in this category. This visualization provides insights into the travel spending patterns of these specific branches.

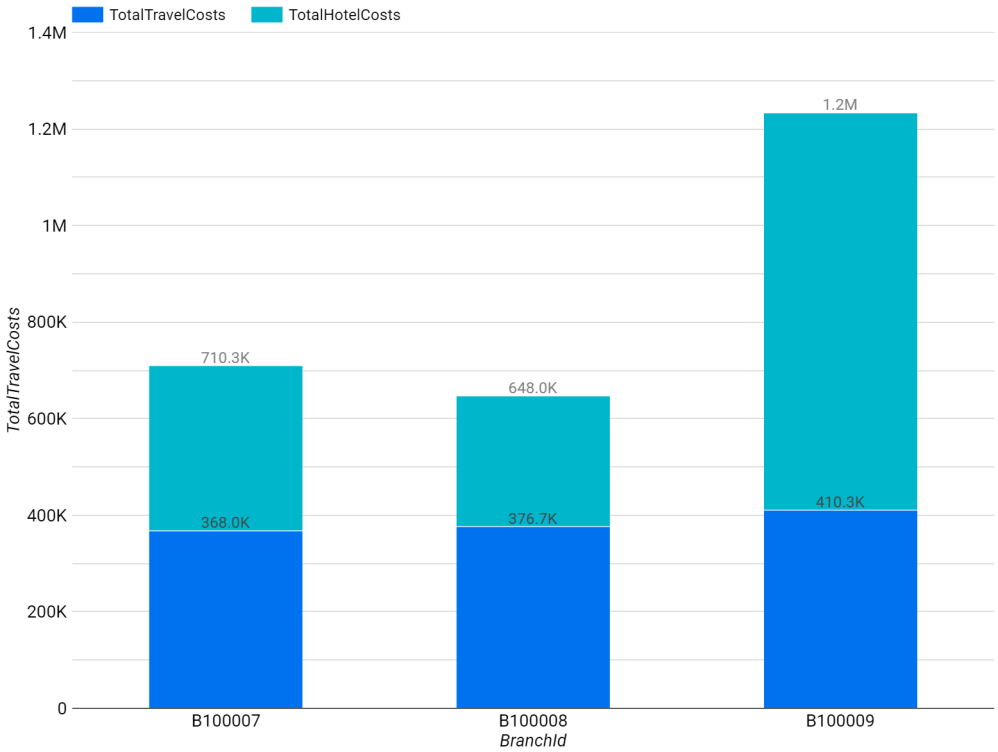

Figure 3.29: Travel and Hotel Cost

- **Average Cost of each Branches in 2021**

  The graph below displays (see Figure 3.30) the average total cost for each branch in 2021, broken down into five categories: Average Travel Costs, Average Hotel Costs, Average Salary Costs, Average Equipment Order Costs, and Average Running Costs. Each branch has a significant portion of its total cost attributed to salary costs (shown in blue), with running costs (in pink) contributing varying amounts across branches. Branches B100013, B100007, and B100004 have the highest total costs, exceeding 2 million. Travel and hotel costs are generally low compared to other categories, with only a few branches showing noticeable expenses in these areas.

- **Year wise average Total Cost in the selected Strategic Region**

  The graph below displays (see Figure 3.31) the average total cost in three selected strategic regions: B100007, B100008, and B100009, over the period from 2000 to 2021. Initially, all three regions show a steady increase in average total cost, with notable growth from 2000 to 2005. After 2005, B100009 experiences more rapid growth compared to B100007 and B100008, ultimately reaching the highest average

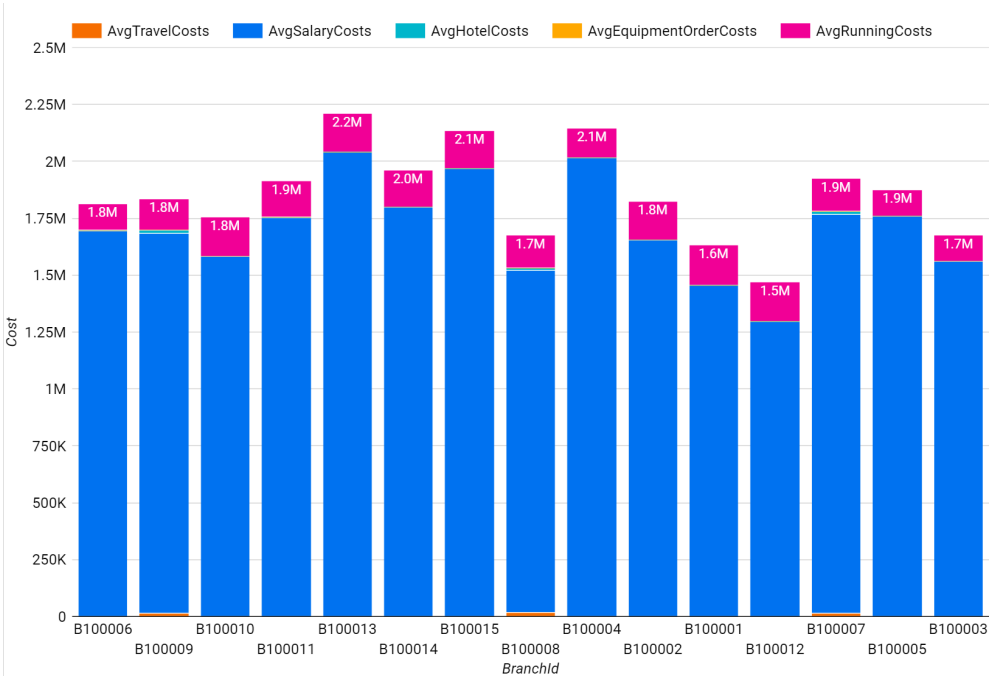

Figure 3.30: Average Cost of each Branches in 2021

total cost by 2021. B100009's total cost surpasses the other two regions starting around 2018 and continues to lead through 2021, indicating its strategic significance and higher expenditures over the years.

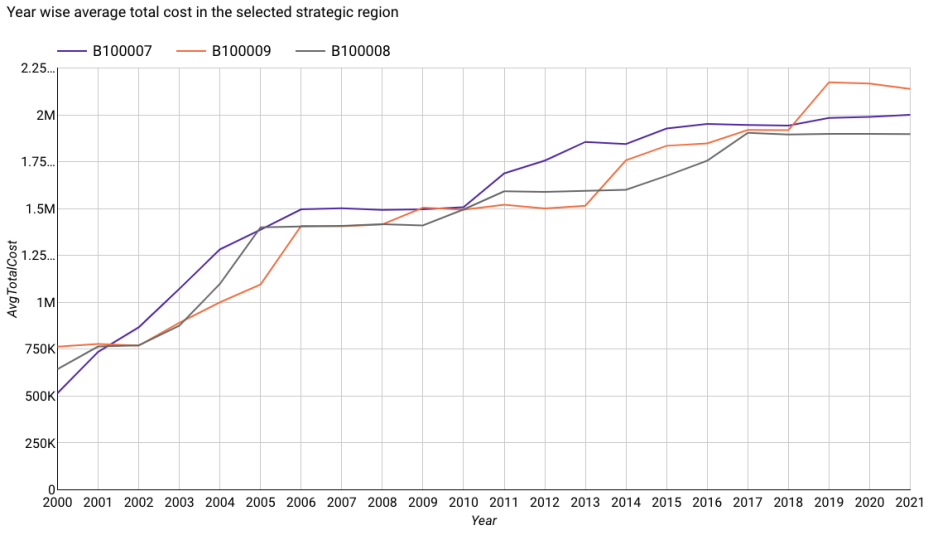

Figure 3.31: Year wise average Total Cost in the selected Strategic Region

4 Visualization of Business trips

In this chapter, we introduce how we used the Google Distance Matrix API to calculate the travel distances between the branches and the destinations of the different trips made by the consultants. The visualization of them will also be shown.

4.1 Data Preparation

we created the following view **BusinessTripsSummaryNew** to extract the origin city with its latitude and longitude and the destination city for each trip. In Figure 4.1 we can see the data lineage of this view. The MwwCcBuisnessTrips2 contains only the origin branch ID and not the city name. That's why we joined the table with the MwwCcr-Branches2 that contains the city name of each branch. We also used the STRUCT data type to organize both latitude and longitude of each origin city to use it later in calculating the distance between cities.
STRUCT is a data type in SQL that represents a collection of elements, where each element can be of a different data type [Bis23].

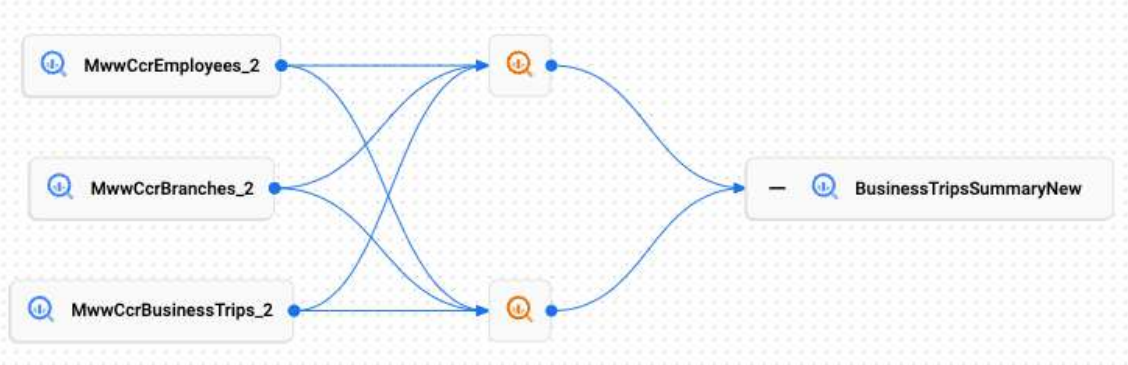

Figure 4.1: data lineage of BusinessTripsSummaryNew

The following SQL is used to create the view:

```
SELECT
  O.TripId,
  O.EmployeeId,
  O.BranchId AS OriginBranchId,
  b.City AS OriginCity,
  STRUCT(b.Latitude, b.Longitude) AS OriginCityCoordinates,
  O.DestinationCity,
  O.DestinationPostalCode,
  O.HotelCosts,
```

```
  O.TravelCosts
FROM
  (
    SELECT
      bt.EmployeeId,
      bt.BusinessTripId AS TripId,
      e.BranchId,
      bt.City AS DestinationCity,
      bt.PostalCode AS DestinationPostalCode,
      bt.HotelCosts,
      bt.TravelCosts
    FROM
      `my-project-scenario2-vlba2.branches.
         MwwCcrBusinessTrips_2` bt
    LEFT JOIN
      `my-project-scenario2-vlba2.branches.MwwCcrEmployees_2`
         e ON bt.EmployeeId = e.EmployeeId
  ) AS O
LEFT JOIN
  `my-project-scenario2-vlba2.branches.MwwCcrBranches_2` b ON
     O.BranchId = b.BranchId
  where  O.HotelCosts> 0
```

TripId	EmployeeId	OriginBranchId	OriginCity	Ori... Latitude	O... Longitude	DestinationCity	DestinationPostalCo	HotelCosts	TravelCosts
BT149160	E108093	B100007	Wahlbach	50.0250712	7.6335463	Gemünden	35329	172.93	25.72
BT159644	E108601	B100007	Wahlbach	50.0250712	7.6335463	Gladenbach	35075	167.24	25.78
BT121687	E107948	B100008	Bekond	49.8389074	6.8519481	Kevelaer	47623	166.07	29.14

Figure 4.2: BusinessTripsSummaryNew

4.2 Calculation of Travel Distances

After creating the view **BusinessTripsSummaryNew**, we used the Google Distance Matrix API to calculate both distance and driving distance for each trip done by a consultant between the origin city (the city where the branch is) and the destination city (the city where the client is).
The Distance Matrix API is a service that accepts an HTTPS request containing origins and destinations for a given mode of transport. For each combination of origin and destination, it returns travel distance and duration [Goo24a].
These steps are done to enable the service in the project:

- In the Navigation Menu select Marketplace (see Figure 4.3)
- search for distance matrix api and select it (see Figure 4.4)
- Click on enable (see Figure 4.5)

Before using this service, which is provided by the GCP, we created an API key associated with the project and restricted it to this API so that it adds security to the project by protecting it from unwarranted requests.
These steps are done to generate the key and to restrict it:

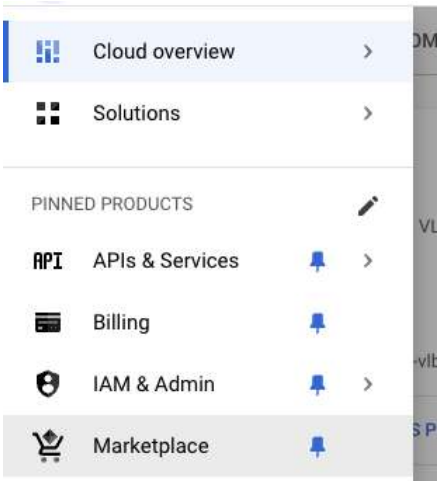

Figure 4.3: Marketplace

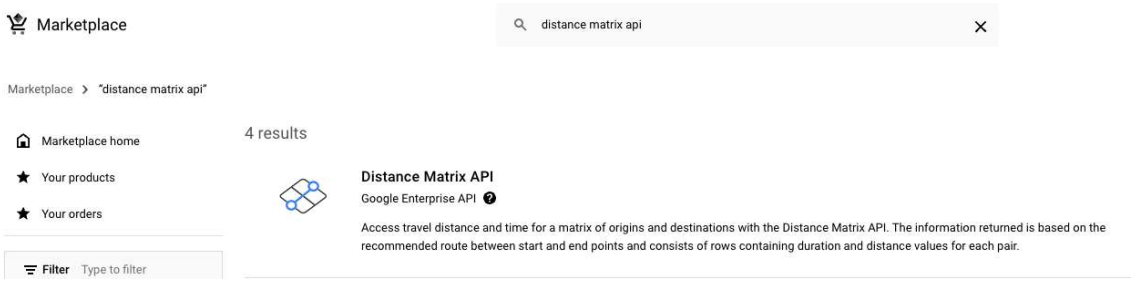

Figure 4.4: search for API

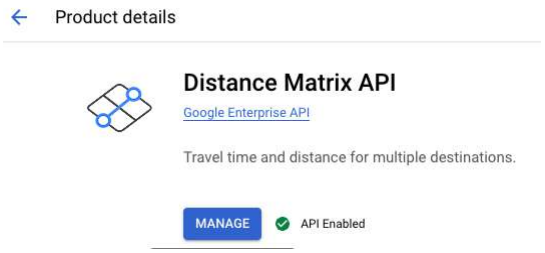

Figure 4.5: Enable API

- In the Navigation Menu select Google Maps Platform and then Keys & Credentials (see Figure 4.6)

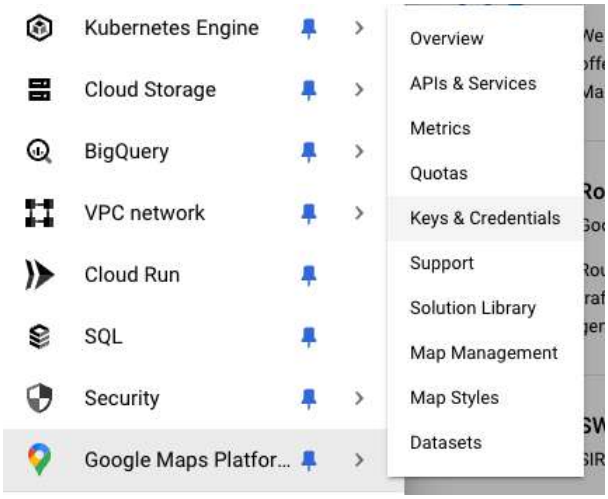

Figure 4.6: Keys and Credentials

- click Create credentials and choose API key (see Figure 4.7)

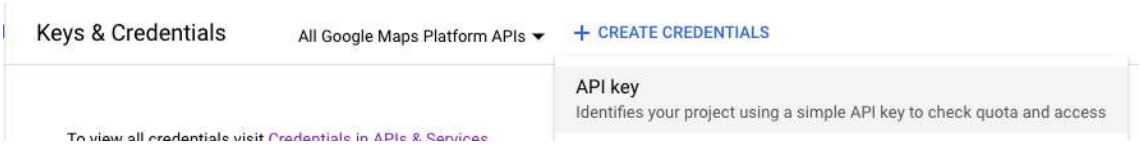

Figure 4.7: create Key

- Click on the generated key and restrict it to the following APIs (see Figure 4.8)

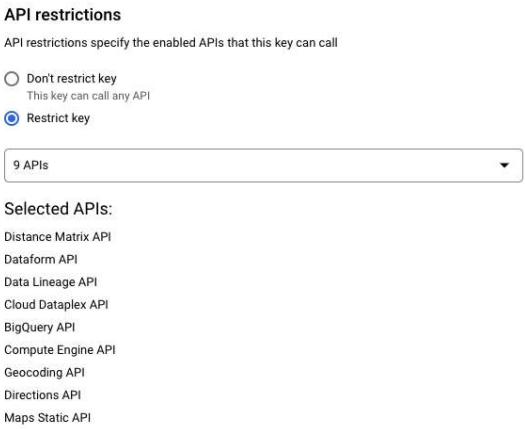

Figure 4.8: restrict key

After these steps we are now able to use the Distance Matrix API to answer the task 2: **Visualize the business trips of the consultants. If a client is at least a 60-minute**

drive away, the company allows the consultants to book an accommodation for the entire trip.

The view BusinessTripsSummaryNew is imported to the Cloud Shell Editor so that we can run the script that we wrote to calculate the distance between the cities. the required libraries are installed in the Cloud Shell environment (using pip3 install googlemaps pandas). Functions are defined to geocode locations and calculate travel time and distance using the Google Maps API. The origin and destination cities are appended with ", Germany" to ensure accurate geocoding. Geocoding is applied to these cities, and travel times and distances are calculated for each trip. The distances are converted from meters to kilometers, and the travel times are converted from seconds to minutes. Also the latitude and longitude of the destination city are added to help in the visualisation later. Finally, the updated DataFrame is saved to a new CSV file called UpdatedBusinessTripsSummaryNew.csv.

BusinessTripsAnalysis.py

```
import googlemaps
from datetime import datetime
import pandas as pd
import logging

gmaps_client = googlemaps.Client(key='
   AIzaSyBymKny0pV9H3AN253jw6ksnotK7QfKQ1g')

logging.basicConfig(level=logging.INFO)

def geocode_location(location):
    try:
        geocode_result = gmaps_client.geocode(location)
        if geocode_result:
            return geocode_result[0]['formatted_address']
        else:
            logging.warning(f"Location not found: {location}")
            return None
    except googlemaps.exceptions.ApiError as e:
        logging.error(f"Geocoding error for {location}: {e}")
        return None

def calculate_travel_time_and_distance(source, destination):
    try:
        now = datetime.now()
        direction_result = gmaps_client.directions(
            source, destination, mode="driving",
               departure_time=now
        )
        if direction_result:
            leg = direction_result[0]['legs'][0]
            distance_km = leg['distance']['value'] / 1000
            duration_mins = leg['duration']['value'] / 60


            destination_lat = leg["end_location"]["lat"]
            destination_lng = leg["end_location"]["lng"]
            destination_coordinates = (destination_lat,
               destination_lng)
```

```
37
38              return duration_mins, distance_km,
                    destination_coordinates
39          else:
40              logging.warning(f"No route found from {source} to
                    {destination}")
41              return None, None
42      except googlemaps.exceptions.ApiError as e:
43          logging.error(f"Error with Google Maps API: {e}")
44          return None, None
45
46  business_trips = pd.read_csv('BusinessTripsSummaryNew.csv')
47
48
49  business_trips['OriginCity'] = business_trips['OriginCity'] +
        ', Germany'
50  business_trips['DestinationCity'] = business_trips['
        DestinationCity'] + ', Germany'
51
52  business_trips['OriginCity'] = business_trips['OriginCity'].
        apply(geocode_location)
53  business_trips['DestinationCity'] = business_trips['
        DestinationCity'].apply(geocode_location)
54
55  business_trips = business_trips.dropna(subset=['OriginCity', '
        DestinationCity'])
56
57  business_trips[['TravelTimeMinutes', 'DistanceKm', '
        DestinationCoordinates']] = business_trips.apply(
58      lambda row: pd.Series(calculate_travel_time_and_distance(
            row['OriginCity'], row['DestinationCity'])), axis=1)
59
60  business_trips['TravelTimeMinutes'] = business_trips['
        TravelTimeMinutes'].fillna(-1)
61  business_trips['DistanceKm'] = business_trips['DistanceKm'].
        fillna(-1)
62
63  business_trips.to_csv('UpdatedBusinessTripsSummaryNew.csv',
        index=False)
64
65  print("Travel times and distances updated successfully")
```

When trying uploading this new file to the Bigquery we received the following error: "Failed to create table: Field name 'OriginCityCoordinates.Latitude' is not supported by the current character map. Please change your field name or use character map V2 to let the system modify the field names."

This problem raised because we used the STRUCT when creating the view BusinessTripsSummaryNew. For this reason we create another python script to clean the data and avoid the error. It creates a new column, OriginCoordinates, by combining the latitude and longitude columns of the origin city and drops the original latitude and longitude columns. The created CSV file has the name **FinalBusinessTripsSummaryNotCleaned.csv**.

CleanData.py

```
1  import pandas as pd
```

```
import re

file_path = 'UpdatedBusinessTripsSummaryNew.csv'
df = pd.read_csv(file_path)

df['OriginCoordinates'] = list(zip(df['
    OriginCityCoordinatesLatitude'], df['
    OriginCityCoordinatesLongitude']))

df.drop(columns=['OriginCityCoordinatesLatitude', '
    OriginCityCoordinatesLongitude'], inplace=True)

columns = list(df.columns)
origin_city_index = columns.index('OriginCity')
columns.insert(origin_city_index + 1, columns.pop(columns.
    index('OriginCoordinates')))
df = df[columns]

output_file_path = 'FinalBusinessTripsSummaryNotCleaned.csv'
df.to_csv(output_file_path, index=False)

print(f"Updated file saved to {output_file_path}")
```

The created file is uploaded into the BigQuery so that it can be used to visualize the different trips.

TripId	EmployeeId	OriginBranchId	OriginCity	OriginCoordinates	DestinationCity	DestinationPostalCode	HotelCosts	TravelCosts	TravelTimeMinutes	DistanceKm	DestinationCoordinates
BT113326	E107755	B100007	Wahlbach	(50.0250712, 7.6335463)	Heiligenroth	56412	1314.11	25.66	59.383333333333333	75.442	(50.4484845, 7.862651)
BT113475	E107755	B100007	Wahlbach	(50.0250712, 7.6335463)	Dermbach-Neidhartshausen	36452	464.52	30.98	170.51666666666668	246.145	(50.6787013, 10.1260453)
BT127273	E107755	B100007	Wahlbach	(50.0250712, 7.6335463)	Heinade	37627	545.13	24.43	243.18333333333331	365.636	(51.8383479, 9.633577599999999)
BT128594	E107755	B100007	Wahlbach	(50.0250712, 7.6335463)	Betzdorf	57518	575.31	28.62	99.666666666666686	121.769	(50.78700660000001, 7.87272560...

Figure 4.9: FinalBusinessTripsSummaryNotCleaned

Although the successful upload, we had a problem in the visualization. Looker Studio requires that the format of the coordination of the city should be in the form "{latitude}, {longitude}". For that, we created a python script to adapt the coordination's of both origin and destination cities. The created file has the name **FinalBusinessTripsSummaryWithNewCoordinates.csv**.

ReformatCoordinations.py

```
import pandas as pd

df = pd.read_csv('FinalBusinessTripsSummaryNotCleaned.csv')

df['LatitudeOrigin'] = '{' + df['OriginCoordinates'].str.
    extract(r'\((.*?),')[0] + '}'
df['LongitudeOrigin'] = '{' + df['OriginCoordinates'].str.
    extract(r', (.*?)\)')[0] + '}'

df['LatitudeDestination'] = '{' + df['DestinationCoordinates'
    ].str.extract(r'\((.*?),')[0] + '}'
df['LongitudeDestination'] = '{' + df['DestinationCoordinates'
    ].str.extract(r', (.*?)\)')[0] + '}'

df['FormattedOriginCoordinates'] = df.apply(lambda row: f"{row
    ['LatitudeOrigin']}, {row['LongitudeOrigin']}", axis=1)
```

```
df['FormattedDestinationCoordinates'] = df.apply(lambda row: f"{row['LatitudeDestination']}, {row['LongitudeDestination']}", axis=1)


df.drop(['LatitudeOrigin', 'LongitudeOrigin'], axis=1, inplace=True)

df = df[['TripId', 'EmployeeId', 'OriginBranchId', 'OriginCity', 'FormattedOriginCoordinates',
         'DestinationCity', 'DestinationPostalCode', 'HotelCosts', 'TravelCosts',
         'TravelTimeMinutes', 'DistanceKm', 'FormattedDestinationCoordinates']]

df.to_csv('FinalBusinessTripsSummaryWithNewCoordinates.csv', index=False)

print(f"Coordination are reformated")
```

The created file can now be uploaded again to BigQuery with the required coordination

TripId	EmployeeId	OriginBranchId	OriginCity	FormattedOriginCoordinates	DestinationCity	DestinationPostalCode	HotelCosts	TravelCosts	TravelTimeMinu	DistanceKm	FormattedDestinationCoordinates
BT113326	E107755	B100007	55 Wahlbach, Germany	(50.0250712), (7.6335463)	56 Heiligenroth, Germany	56412	1314.11	25.66	59.3833333...	75.442	(50.4484845), (7.862651)
BT113475	E107755	B100007	55 Wahlbach, Germany	(50.0250712), (7.6335463)	36466 Dermbach-Neidhartshausen, Germany	36452	464.52	30.98	170.516666...	246.145	(50.6787013), (10.1260453)
BT127273	E107755	B100007	55 Wahlbach, Germany	(50.0250712), (7.6335463)	37627 Heinade, Germany	37627	545.13	24.43	243.183333...	365.636	(51.8383479), (9.63357759999...
BT128594	E107755	B100007	55 Wahlbach, Germany	(50.0250712), (7.6335463)	57 Betzdorf, Germany	57518	575.31	28.62	99.6666666...	121.769	(50.78700660000001), (7.8727...
BT120662	E107755	B100007	55 Wahlbach, Germany	(50.0250712), (7.6335463)	Am bhf, 57612 Ingelbach, Germany	57612	903.7	28.56	84.75	107.081	(50.6881548), (7.7247418)
BT156193	E107755	B100007	55 Wahlbach, Germany	(50.0250712), (7.6335463)	39 Werben, Germany	39615	1695.25	32.4	388.7	598.833	(52.86120529999999), (11.982...

Figure 4.10: FinalBusinessTripsSummaryWithNewCoordinates

4.3 Trips Visualization

Utilizing Looker Studio, we visualize business travel data efficiently. Using tools like bubble maps for trip origins, heat maps for destination density, and cost distribution maps for travel and lodging expenses, we derive insights to optimize branch locations based on travel patterns and costs.

- **Bubble Map visualization** Firstly we visualized the businness trips of our consultants to various clients , based on their corresponding branch Id , as you can see (See Figure 4.11) we used a bubble map visualization and the corresponding colours in the legend shows their corresponding trips from the respective Origin Branch Id.

- **Heat Map**

  According to the heat map below figure (See Figure 4.12), Hanover and Garbsen are the primary destinations for business travel. Garbsen has a higher density of business trip records, indicating that it is a frequently visited location for business trips. Similar to Garbsen, Hanover also sees a lot of business travel, albeit to a slightly lesser degree. This focus emphasizes how crucial these sites are to general patterns of business travel and implies that Garbsen would be a great place to open a new branch.

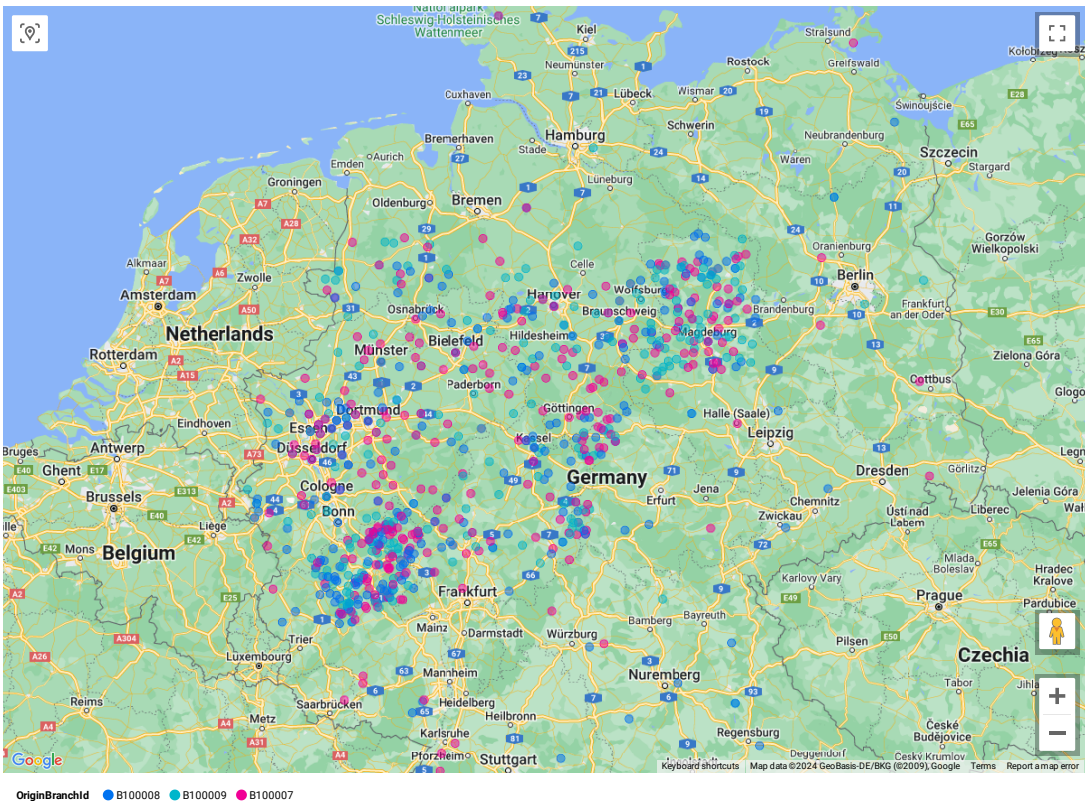

Figure 4.11: Bubble Maps Visualization

- **Travel Cost Distribution**

  The map below illustrates (See Figure 4.13)the distribution of travel costs associated with different origin branches across various parts of Germany. Within a strategic zone, each colour denotes a distinct branch, and each point represents the cost of travel to that location. The areas with the highest concentration of travel expenses are Garbsen and Hanover, respectively, as indicated by the bigger blue and red circles. This visualization helps identify areas where travel costs are high, making it easier to make strategic judgements about branch placement.

- **Hotel Cost Distribution**

  The map below illustrates (See Figure 4.14) the distribution of hostel costs associated with different origin branches across various parts of Germany. Each color represents the hostel costs from different branches to that location within a strategic region. The large red and blue circles highlight Garbsen and Hanover, respectively, as areas where the majority of hostel costs are concentrated. This visualization aids in pinpointing regions with high hostel expenses, with Garbsen standing out with the highest hostel costs at 366,294.93. Such insights facilitate strategic decisions for optimizing branch locations.

Figure 4.12: Heat Map Visualization

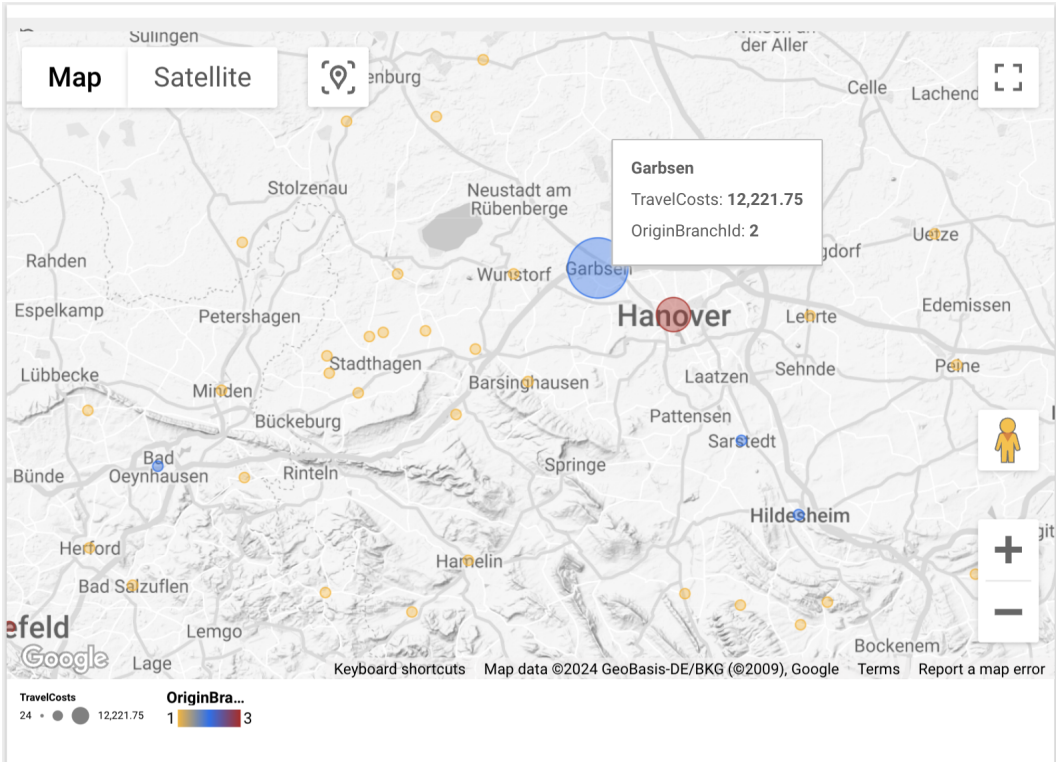

Figure 4.13: Travel Cost Distribution

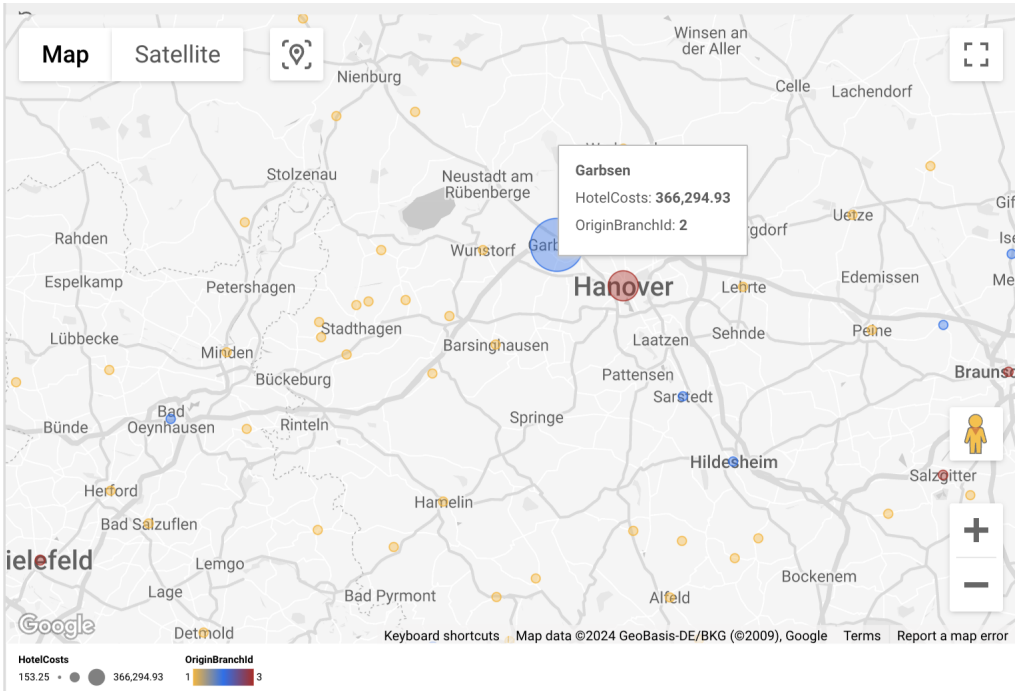

Figure 4.14: Hotel Cost Distribution

5 Feasibility Study and Recommendations for New Branch Office Location

In this section, we detail the procedure undertaken to determine the most sensible location for opening a new branch office based on the analysis of business trip data.

5.1 Analyzing Business Trip Data

Using SQL queries in BigQuery, we analyzed the uploaded dataset presented in section 4.2 FinalBusinessTripsSummaryWithNewCoordinates to determine the total number of trips made by consultants to various cities. The query aggregated the trip data by city, providing us with a clear view of the most frequently visited destinations. The following query is used:

```
SELECT
  DestinationCity,
  count (*) AS total_trips
FROM
  `my-project-scenario2-vlba2.branches.
     FinalBusinessTripsSummaryWithNewCoordinates`
WHERE
  HotelCosts > 0
GROUP BY
  DestinationCity
ORDER BY
  total_trips DESC
```

DestinationCity	total_trips
30 Garbsen, Germany	410
30 Hanover, Germany	196
Cologne, Germany	19

Figure 5.1: most frequently visited destinations

From the analysis (Figure 5.1), we observed that Garbsen and Hanover were the cities with the highest number of business trips. This leads to the conclusion that these locations were prime candidates for opening a new branch office to reduce travel costs and improve operational efficiency.

5.2 Proposal for a New Branch Office in the Hanover/Garbsen Region

This section presents a comparative analysis of two approaches to evaluate the feasibility of opening a new branch office in the Hanover/Garbsen region. The goal is to determine the most cost-effective and strategically advantageous solution for serving clients in this area.

5.2.1 Approach 1: Direct Cost Comparison

The first approach involves a straightforward comparison of the average running costs of our existing branches (B100007, B100008, B100009) for the latest year, 2021, with the combined travel and hotel costs incurred for trips to Garbsen and Hanover in 2021.

- **Existing Branches**: The average running cost for the branches B100007, B100008, and B100009 in 2021 was 157.266K (see Figure 5.2). This figure is derived from the running cost expenses incurred by these branches throughout the year, including operational expenditures necessary to maintain branch functionality.

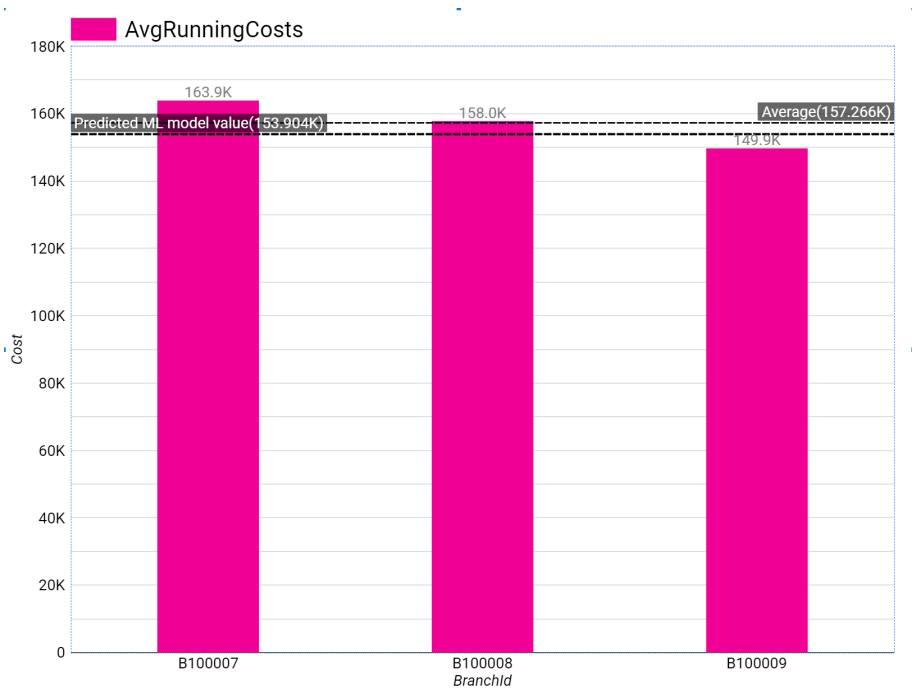

Figure 5.2: Average running costs of B100007, B100008, B100009

- **Garbsen and Hanover**: In contrast, the combined travel and hotel costs for trips to Garbsen and Hanover alone in 2021 amounted to 165.426K (see Figure 5.3). This cost represents the financial burden of sending consultants to these cities, including transportation and lodging expenses.

This direct comparison reveals a crucial insight: the travel and hotel expenses for serving clients in Garbsen and Hanover exceed the average cost of operating an entire branch.

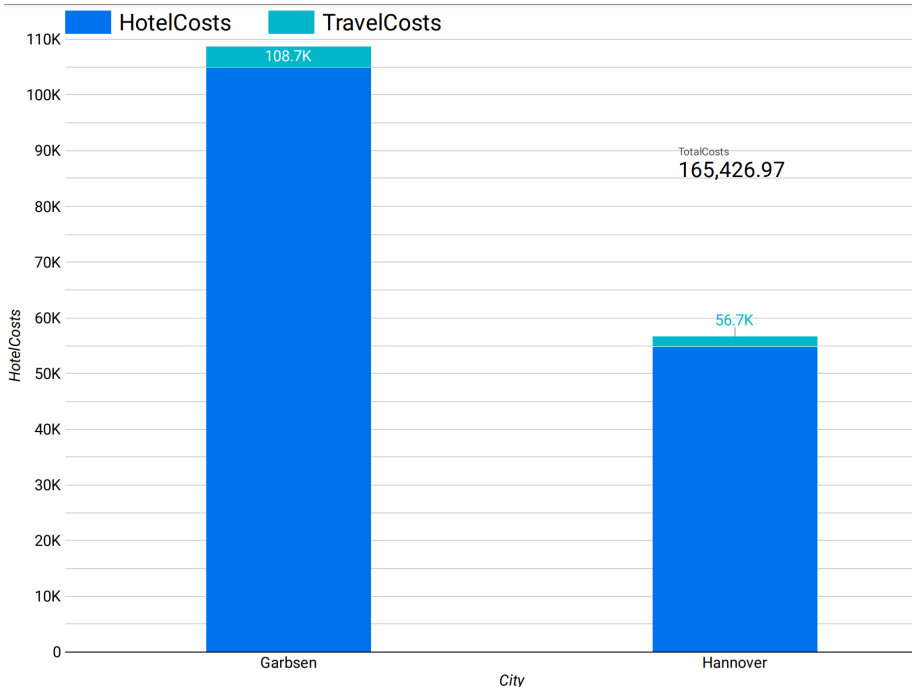

Figure 5.3: Total Hotel and Travel Costs for Garbsen and Hanover

This suggests that opening a new branch office in a strategic location between these two cities could lead to significant cost savings. By reducing the need for extensive travel and overnight stays, the company can optimize operational efficiency and reduce expenses. We also solidified our approach with the next method as well.

5.2.2 Approach 2: Comparison with ML Predicted Value

The second approach leverages predictive analytics using a Machine Learning (ML) model to estimate the running costs of a new branch office. This method provides a forward-looking perspective on potential savings and efficiencies.

5.2.2.1 Model Creation and Training

We used BigQuery ML to create and train a linear regression model based on historical running cost data from our branches B100007, B100008, and B100009, spanning from the year 2000 to 2021. The training process firstly involved the creation of a model using the following steps:

```
CREATE OR REPLACE MODEL branches.avg_running_cost_model
OPTIONS(model_type='linear_reg', input_label_cols=['
    AvgRunningCost']) AS
SELECT BranchId, AVG(SUMRunningCost) as AvgRunningCost, Year
FROM branches.ToTALCOSTSOFABRANCHPERYEAR
WHERE BranchId IN ('B100007', 'B100008', 'B100009')
GROUP BY BranchId, Year;
```

5.2.2.2 Model Evaluation

To evaluate the model, we used the following query:

```
SELECT *
FROM ML.EVALUATE(MODEL branches.avg_running_cost_model, (
  SELECT BranchId, AVG(SUMRunningCost) AS AvgRunningCost, Year
  FROM branches.ToTALCOSTSOFABRANCHPERYEAR
  WHERE BranchId IN ('B100007', 'B100008', 'B100009')
  GROUP BY BranchId, Year
));
```

5.2.2.3 Cost Prediction for New Branch

We then used the model to predict the average running cost for a hypothetical new branch (B100016) in the year 2022:

```
SELECT BranchId, Year, predicted_AvgRunningCost
FROM ML.PREDICT(MODEL branches.avg_running_cost_model, (
  SELECT 'B100016' AS BranchId, 2022 AS Year
));
```

Row	BranchId	Year	predicted_AvgRunningCost
1	B100016	2022	153904.80761904761

Figure 5.4: Predicted running costs of new branch

The predicted running cost for the new branch was 153.9K (see Figure 5.4). This value is significantly lower than the combined travel and hotel costs for Garbsen and Hanover (165.426K), further supporting the potential profitability of opening a new branch.
To enhance clarity and understanding, we have also visualized a cost comparison (see Figure 5.5).

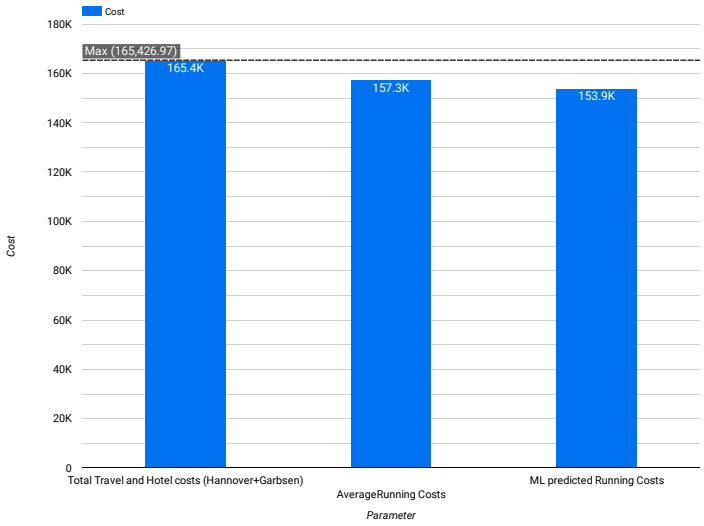

Figure 5.5: Comparison of the values obtained

This figure illustrates the substantial difference between the combined travel and hotel costs incurred for business trips in Hannover and Garbsen in 2021 (165.4K) and the average running costs of existing branches (157.3K). Additionally, the figure shows the predicted average running cost for a hypothetical new branch in the same region (153.9K) using our machine learning model.
The stark contrast between the high travel and hotel costs and the lower predicted running costs underscores the potential financial benefits of establishing a new branch. This strategic move could also decrease travel times for consultants, improving service delivery efficiency and client satisfaction.

5.3 New Branch Office Location

We have now two candidates cities to open the new branch. To further refine our decision, we calculated the travel distances and driving times between the identified cities (Garbsen and Hanover). The calculations were performed using the Google Maps API with the following python script:

```
import googlemaps
from datetime import datetime

gmaps_client = googlemaps.Client(key="
   AIzaSyBymKny0pV9H3AN253jw6ksnotK7QfKQ1g")
now = datetime.now()
source = "Garbsen"
destination = "Hanover"

try:
    direction_result = gmaps_client.directions(
        source + ", Germany", destination + ", Germany", mode=
           "driving", departure_time=now
    )

    leg = direction_result[0]['legs'][0]
    distance_km = leg['distance']['value'] / 1000
    duration_mins = leg['duration']['value'] / 60

    print(f"Distance: {distance_km:.2f} km")
    print(f"Duration: {duration_mins:.0f} minutes")

except googlemaps.exceptions.ApiError as e:
    print(f"Error with Google Maps API: {e}")
```

As a result we got that the Distance between Garbsen and Hanover is 12.69 km with driving time 18 minutes, which is less then 60 minutes. Given this finding, it was determined that opening a new branch in Garbsen alone would be sufficient to cover the demand in both cities. This decision would optimize resource allocation without incurring unnecessary costs.

5.3.1 Native Approach

To evaluate the potential cost savings of opening a new branch in Garbsen, we developed a Python script to calculate the driving distance and time between Garbsen and various other cities listed in MwwCcrBusinessTrips2. The script uses the Google Maps API to determine driving distances and times for each trip.

GarbsenToAll.py

```
import googlemaps
from datetime import datetime
import pandas as pd
import logging

gmaps_client = googlemaps.Client(key='
   AIzaSyBymKny0pV9H3AN253jw6ksnotK7QfKQ1g')

logging.basicConfig(level=logging.INFO)

def geocode_location(location):
    try:
        geocode_result = gmaps_client.geocode(location)
        if geocode_result:
            return geocode_result[0]['formatted_address']
        else:
            logging.warning(f"Location not found: {location}")
            return None
    except googlemaps.exceptions.ApiError as e:
        logging.error(f"Geocoding error for {location}: {e}")
        return None

def calculate_travel_time_and_distance(destination):
    try:
        now = datetime.now()
        direction_result = gmaps_client.directions(
            'Garbsen, Germany', destination, mode="driving",
               departure_time=now
        )
        if direction_result:
            leg = direction_result[0]['legs'][0]
            distance_km = leg['distance']['value'] / 1000
            duration_mins = leg['duration']['value'] / 60

            return duration_mins, distance_km
        else:
            logging.warning(f"No route found from {source} to
               {destination}")
            return None, None
    except googlemaps.exceptions.ApiError as e:
        logging.error(f"Error with Google Maps API: {e}")
        return None, None

business_trips = pd.read_csv('formatted_file2.csv')


business_trips['OriginCity'] = 'Garbsen , Germany'
business_trips['DestinationCity'] = business_trips['
```

```
    DestinationCity'] + ', Germany'

business_trips['OriginCity'] = business_trips['OriginCity'].
    apply(geocode_location)
business_trips['DestinationCity'] = business_trips['
    DestinationCity'].apply(geocode_location)

business_trips = business_trips.dropna(subset=['
    DestinationCity'])

business_trips[['TravelTimeMinutes', 'DistanceKm']] =
    business_trips.apply(
     lambda row: pd.Series(calculate_travel_time_and_distance(
        row['DestinationCity'])), axis=1)

business_trips['TravelTimeMinutes'] = business_trips['
    TravelTimeMinutes'].fillna(-1)
business_trips['DistanceKm'] = business_trips['DistanceKm'].
    fillna(-1)

business_trips.to_csv('gg.csv', index=False)

print("Travel times and distances calculated successfully")
```

TripId	EmployeeId	OriginBranchId	OriginCity	FormattedOriginCoordinates	DestinationCity	DestinationPost	HotelCosts	TravelCosts	TravelTimeMinu	DistanceKm	FormattedDestinationCoordinates
BT113326	E107755	B100007	30 Garbsen, Germany	(50.0250712), (7.6335463)	56 Heiligenroth, Germany	56412	1314.11	25.66	234.316666...	372.27	(50.4484845), (7.862651)
BT113475	E107755	B100007	30 Garbsen, Germany	(50.0250712), (7.6335463)	36466 Dermbach-Neidhartshausen, Germ...	36452	464.52	30.98	194.183333...	256.937	(50.6787013), (10.1260453)
BT127273	E107755	B100007	30 Garbsen, Germany	(50.0250712), (7.6335463)	37627 Heinade, Germany	37627	545.13	24.43	84.8833333...	84.656	(51.8383479), (9.63357759999...
BT128594	E107755	B100007	30 Garbsen, Germany	(50.0250712), (7.6335463)	57 Betzdorf, Germany	57518	575.31	28.62	211.883333...	302.047	(50.78700660000001), (7.8727...
BT120662	E107755	B100007	30 Garbsen, Germany	(50.0250712), (7.6335463)	Am bhf, 57612 Ingelbach, Germany	57612	903.7	28.56	232.3	348.982	(50.6881548), (7.7247418)

Figure 5.6: distance between Garbsen and all other cities

We then created a query to calculate the number of cities and their costs with driving times less than 60 minutes from Garbsen. This analysis helps us estimate the potential cost savings by reducing the need for accommodations on these short trips.

```
SELECT
  SUM(a.HotelCosts+a.TravelCosts) AS TotalSavingCost
FROM
  `my-project-scenario2-vlba2.branches.GarbsenToAllNative`  AS
      a
LEFT JOIN
  `my-project-scenario2-vlba2.branches.MwwCcrBusinessTrips_2`
     AS b
ON
  a.TripId = b.BusinessTripId
WHERE
 TravelTimeMinutes < 60 AND EXTRACT(YEAR FROM b.StartDate)  =
    2021
```

There are around 42 cities that are less then 60 minutes driving time far from Garbsen. The previous query shows that opening a branch in Garbsen will save approximately 165427 euro in total in the year 2021.

5.3.2 Machine Learning Approach

In this approach we determine the optimal location for a new branch based on the travel patterns of our employees. By leveraging historical data on business trips, we applied k-means clustering to group locations into clusters and identify the cluster centroid with the highest number of trips. This centroid represents the ideal new branch location.

Model Training

We trained a k-means clustering model using BigQuery ML to categorize the destinations into five clusters.

```
CREATE OR REPLACE MODEL branches.client_location_clustering
OPTIONS(model_type='kmeans', num_clusters=5) AS
SELECT
    CAST(SPLIT(CleanedDestinationCoordinates, ',')[OFFSET(0)]
       AS FLOAT64) AS Latitude,
    CAST(SPLIT(CleanedDestinationCoordinates, ',')[OFFSET(1)]
       AS FLOAT64) AS Longitude
FROM
    branches.FinalBusinessTripsSummaryCleanedCoordinates
WHERE CleanedDestinationCoordinates != 'nan, nan';
```

Clusters and Centroids

After training the model, we retrieved the centroids for each cluster, representing the central points of each group of locations.

```
WITH CentroidCoordinates AS (
    SELECT
        centroid_id AS cluster,
        MAX(CASE WHEN feature = 'Latitude' THEN
           numerical_value END) AS CentroidLatitude,
        MAX(CASE WHEN feature = 'Longitude' THEN
           numerical_value END) AS CentroidLongitude
    FROM
        ML.CENTROIDS(MODEL branches.client_location_clustering
           )
    GROUP BY
        centroid_id
),
CountLocations AS (
    SELECT
        CENTROID_ID AS cluster,
        COUNT(*) AS TotalLocations
    FROM
        ML.PREDICT(MODEL branches.client_location_clustering,
            (
                SELECT
                    Latitude,
                    Longitude
                FROM
                    branches.client_locations
            )
        )
    GROUP BY
        CENTROID_ID
)
```

```
SELECT
    cc.cluster,
    cc.CentroidLatitude,
    cc.CentroidLongitude,
    cl.TotalLocations
FROM
    CentroidCoordinates cc
JOIN
    CountLocations cl ON cc.cluster = cl.cluster;
```

The Query provides the results as follows. Based on the results, the cluster with the centroid at latitude 52.392908618435023 and longitude 9.55142128527852 has the highest number of trips (750), making it the ideal location for the new branch.

Row	cluster	CentroidLatitude	CentroidLongitude	TotalLocations
1	1	50.18369084104...	7.6981568511194016	279
2	2	50.85693517218...	10.065753033774834	150
3	3	51.39581200434...	7.2107609865217412	226
4	4	52.29978564744...	11.572560654591836	195
5	5	52.39290861843...	9.55142128527852	750

Figure 5.7: Cluster centroids and the total count of locations.

We can also visualize the locations clustered by the model using the coordinates to plot in a map.

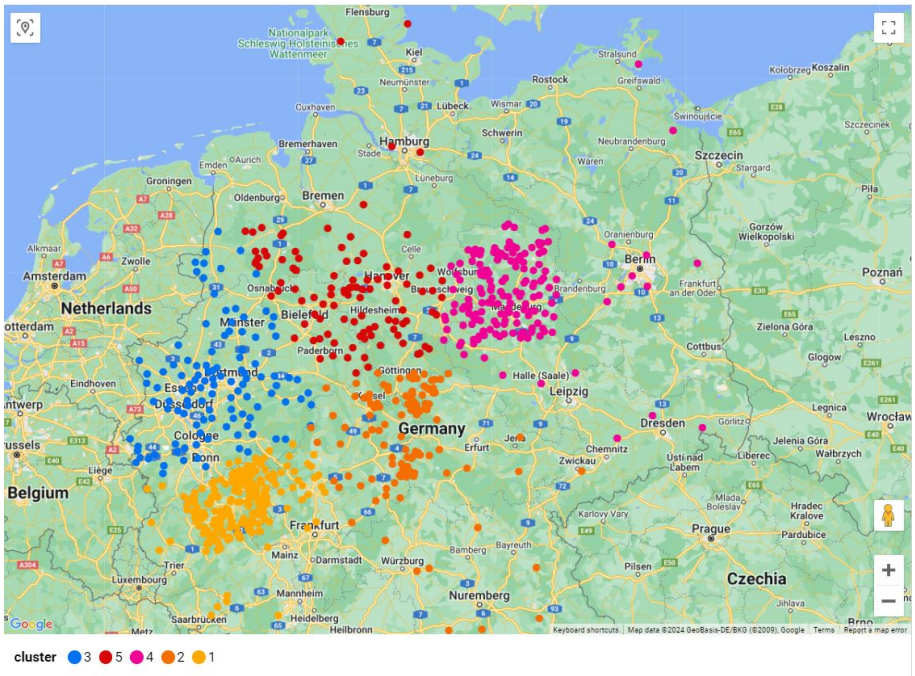

Figure 5.8: Cluster of business trip locations.

For the obtained coordinates we use the Python Script to calculate the driving distance and time between the cluster 5 centroid to other cities around it.

CentroidToAll.py

```
import googlemaps
from datetime import datetime
import pandas as pd
import logging

gmaps_client = googlemaps.Client(key='AIzaSyBymKny0pV9H3AN253jw6ksnotK7QfKQ1g')

logging.basicConfig(level=logging.INFO)

def geocode_location(location):
    try:
        geocode_result = gmaps_client.geocode(location)
        if geocode_result:
            return geocode_result[0]['formatted_address']
        else:
            logging.warning(f"Location not found: {location}")
            return None
    except googlemaps.exceptions.ApiError as e:
        logging.error(f"Geocoding error for {location}: {e}")
        return None

def calculate_travel_time_and_distance(destination):
    try:
        now = datetime.now()
        direction_result = gmaps_client.directions(
            (52.392908618435023, 9.55142128527852),
               destination, mode="driving", departure_time=now
        )
        if direction_result:
            leg = direction_result[0]['legs'][0]
            distance_km = leg['distance']['value'] / 1000
            duration_mins = leg['duration']['value'] / 60

            return duration_mins, distance_km
        else:
            logging.warning(f"No route found from {source} to
               {destination}")
            return None, None
    except googlemaps.exceptions.ApiError as e:
        logging.error(f"Error with Google Maps API: {e}")
        return None, None

business_trips = pd.read_csv('formatted_file2.csv')

business_trips['OriginCity'] = 'Garbsen , Germany'
business_trips['DestinationCity'] = business_trips['DestinationCity'] + ', Germany'

business_trips['OriginCity'] = business_trips['OriginCity'].apply(geocode_location)
business_trips['DestinationCity'] = business_trips['DestinationCity'].apply(geocode_location)

business_trips = business_trips.dropna(subset=['DestinationCity'])
```

```
50
51 business_trips[['TravelTimeMinutes', 'DistanceKm']] =
      business_trips.apply(
52     lambda row: pd.Series(calculate_travel_time_and_distance(
          row['DestinationCity'])), axis=1)
53
54 business_trips['TravelTimeMinutes'] = business_trips['
      TravelTimeMinutes'].fillna(-1)
55 business_trips['DistanceKm'] = business_trips['DistanceKm'].
      fillna(-1)
56
57 business_trips.to_csv('gg2.csv', index=False)
58
59 print("Travel times and distances updated successfully")
```

TripId	EmployeeId	OriginBranchId	OriginCity	FormattedOriginCoordinates	DestinationCity	DestinationPost	HotelCosts	TravelCosts	TravelTimeMinutes	DistanceKm	FormattedDestinationCoordinates
BT113326	E107755	B100007	30 Garbsen, Germany	{50.0250712}, {7.6335463}	56 Heiligenroth, Germany	56412	1314.11	25.66	228.28333333333333	365.018	{50.4484845}, {7.862651}
BT113475	E107755	B100007	30 Garbsen, Germany	{50.0250712}, {7.6335463}	36466 Dermbach-Neidhartshau...	36452	464.52	30.98	196.96666666666667	265.269	{50.6787013}, {10.1260453}
BT127273	E107755	B100007	30 Garbsen, Germany	{50.0250712}, {7.6335463}	37627 Heinade, Germany	37627	545.13	24.43	86.583333333333329	99.137	{51.8383479}, {9.63357759999999}
BT128594	E107755	B100007	30 Garbsen, Germany	{50.0250712}, {7.6335463}	57 Betzdorf, Germany	57518	575.31	28.62	205.85	294.795	{50.78700660000001}, {7.872725600000001}
BT120662	E107755	B100007	30 Garbsen, Germany	{50.0250712}, {7.6335463}	Am bhf, 57612 Ingelbach, Ger...	57612	903.7	28.56	226.26666666666668	341.73	{50.6881548}, {7.7247418}
BT156193	E107755	B100007	30 Garbsen, Germany	{50.0250712}, {7.6335463}	39 Werben, Germany	39615	1695.25	32.4	180.86666666666667	265.459	{52.86120529999999}, {11.9823923}

Figure 5.9: number of cities close to Centroid and their costs

After calculating the distance and travel time between the centroid (Garbsen) and various cities, we performed a query to identify the cities that are less than 60 minutes driving from Garbsen.

```
1 SELECT
2   FormattedDestinationCoordinates AS ClientCloseToGarbsen
3 FROM
4   `my-project-scenario2-vlba2.branches.GardsenToAllML`
5 WHERE
6   TravelTimeMinutes < 60
7 GROUP BY
8   FormattedDestinationCoordinates
```

ClientCloseToGarbsen
{52.3303732}, {9.10602}
{52.5006189}, {9.8925889}
{52.2657521}, {9.3392588}
{52.0764867}, {8.75570149999...
{52.0513942}, {9.2522064}
{52.2677917}, {10.5245329}
{52.7093025}, {9.0708205}
{52.3267677}, {10.3904652}

Figure 5.10: cities close to Centroid

The filtered results from BigQuery were then uploaded to Google Cloud Shell. We used the Maps Static API to visualize the position of the centroid and the cities from the previous

query.

Maps Static API is a tool provided by the Google Cloud Platform that let us embed a Google Maps image without requiring JavaScript or any dynamic page loading. The Maps Static API service creates a map based on URL parameters sent through a standard HTTP request and returns the map as an image [Goo24b].

We create a python script called GenerateMap.py that visualizes the proximity of various cities to Garbsen on a map, highlighting Garbsen with a distinct marker, and showing all cities within a 60-minute drive.

GenerateMap.py

```
import pandas as pd
import googlemaps
import requests

gmaps_client = googlemaps.Client(key='
    AIzaSyBymKny0pV9H3AN253jw6ksnotK7QfKQ1g')

df = pd.read_csv('inputformaps.csv')
garbsen_coordinates = (52.392908618435023, 9.55142128527852)
base_url = "https://maps.googleapis.com/maps/api/staticmap?"
params = {
    "size": "800x600",
    "key": 'AIzaSyBymKny0pV9H3AN253jw6ksnotK7QfKQ1g'
}
garbsen_marker = f"color:red|size:mid|label:C|{
    garbsen_coordinates[0]},{garbsen_coordinates[1]}"

city_markers = [garbsen_marker]
for idx, row in df.iterrows():
    coordinates = row['FormattedDestinationCoordinates'].
        replace("{", "").replace("}", "").split(", ")
    lat, lng = float(coordinates[0]), float(coordinates[1])
    city_marker = f"size:mid|color:blue|{lat},{lng}"
    city_markers.append(city_marker)

markers_param = '&'.join([f"markers={marker}" for marker in
    city_markers])
map_url = f"{base_url}{markers_param}&key={params['key']}&size
    ={params['size']}"

print("Generated Map URL:", map_url)

response = requests.get(map_url)
if response.status_code == 200:
    with open('mapNew.png', 'wb') as f:
        f.write(response.content)
    print("Map image saved as mapNew.png")
else:
    print("Failed to retrieve the map image")
```

The results of this analysis are represented in Figure 5.11. The central point marked in red represents Garbsen, the proposed location of a new branch office. The blue markers indicate the cities that are within a 60-minute driving distance from Garbsen. This visual

representation demonstrates the strategic advantage of opening a new branch in Garbsen, as it shows the significant number of cities that are easily accessible, potentially reducing travel costs and time for business trips.

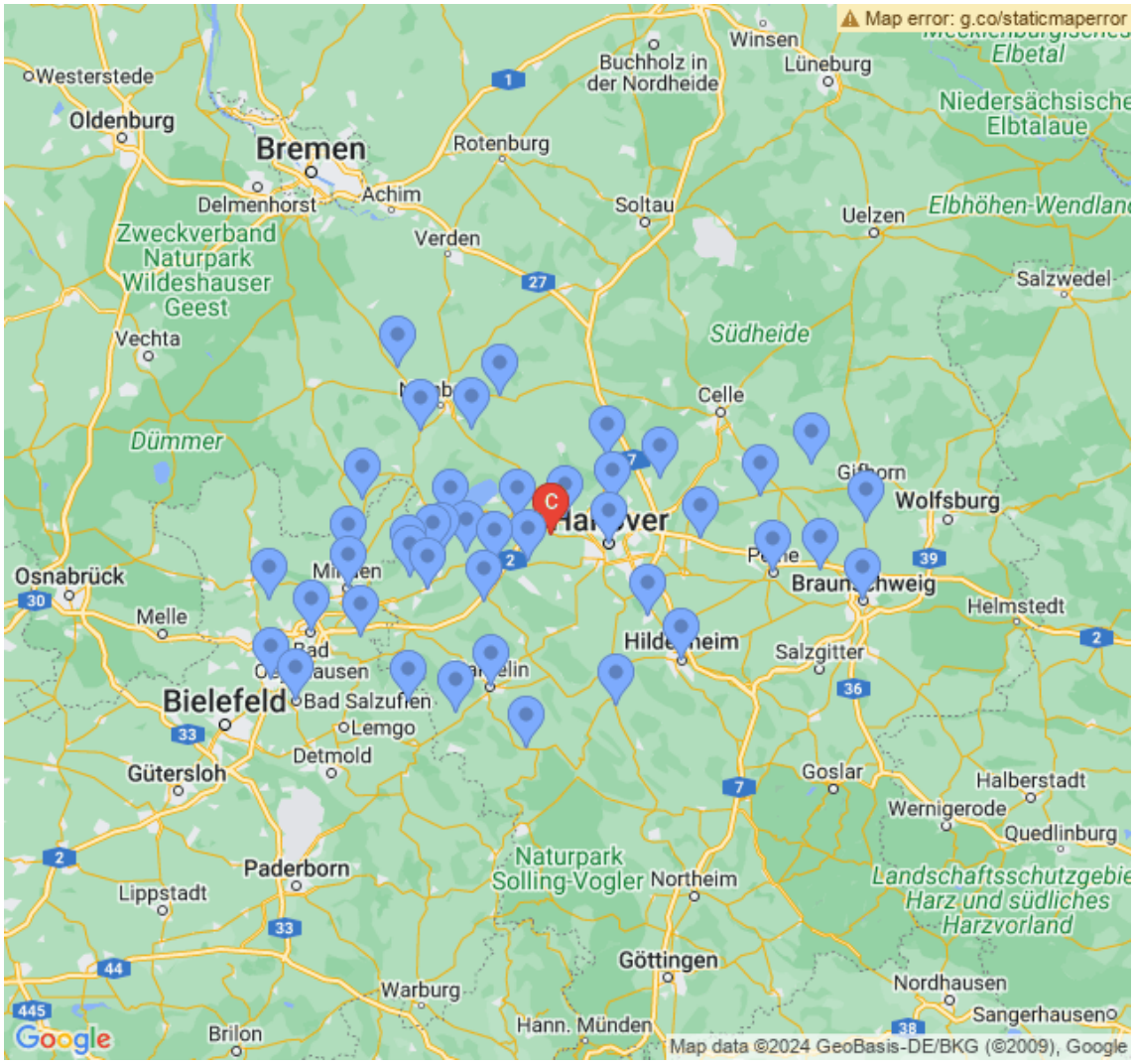

Figure 5.11: Map visualization

6 Conclusion

This project examines Mobility Worldwide's Consulting & Customer Retention (CCR) department, offering valuable insights into branch office financials and the possibility of setting up a new branch.
We started by analyzing the average costs of running a branch office, and then looking at factors like staff salaries, equipment order costs, and other operational expenses. We found that costs varied significantly across branches. We also tried to drill down and concentrate mainly on the branches in our strategic region GER100003(B100007, B100008,B100009) to find more insights and mostly concentrated on the latest year, 2021.
Next, we examined the business trips taken by consultants, particularly those requiring overnight stays due to distance. Using the Google Distance Matrix API, we calculated travel distances and times, revealing patterns and costly destinations.
Building on this, we came to the conclusion of opening a new branch office in the Hanover/-Garbsen region, since that regions had the most number of trips to . We used two approaches to evaluate if it is feasible to open a new branch:

- Direct Cost Comparison: We observed that travel and hotel costs incurred in the Hanover and Garbsen regions were already higher than the average cost of running a branch, pointing to potential savings with a new branch office.
- Predictive Cost Modeling: We created a machine learning model to predict the running costs of a new branch in the region. The predicted running cost was much lower than the current travel and hotel expenses, further supporting the idea of a new branch.

Both approaches provide solid evidence that establishing a new branch office in the Hanover/-Garbsen region would be financially viable. This would not only save money but also dramatically reduce travel times for our consultants.
In addition, to find the location of our new branch, the most probable answer was in the Garbsen region, as this region had the most number of trips. We also wanted to strengthen our claim that the selected region is accurate so we used a clustering approach using BigQuery ML. In that approach, we determined the optimal location for a new branch based on the travel patterns of our employees. By leveraging historical data on business trips, we applied k-means clustering to group locations into clusters and identify the cluster centroid with the highest number of trips. This centroid represents the ideal new branch location, and the found centroid location also happens to be in the Garbsen region.

We then finalized our new branch office location to be in Garbsen region and then we calculated the distance and travel time between the centroid (Garbsen) and various cities, to identify the client locations that are less than 60 minutes driving distance from Garbsen and identified potential 42 client locations that are within a 60min travel time from the supposed new branch office location, Garbsen.
Opening a branch in Garbsen as we saw before will reduce much costs for the company, especially with regard to trips and hotel costs. As Garbsen is close to many client's locations (around 42 cities) within a 60-minute drive, it is an ideal location to save traveling costs and time. The company could have potentially saved around €165427 in travel and hotel costs if a branch had been opened in 2021. So, it would be sensible to open a new branch office. Establishing a new branch eliminates the possible cost incurred as Travel and Hotel Costs to the nearby identified 42 cities in the upcoming years from the new branch .

Bibliography

[Bis23] Iqra Bismi. What is struct in sql?, 2023. [Online; accessed 30-June-2024].

[Goo24a] Google. Distance matrix api overview, 2024. [Online; accessed 30-June-2024].

[Goo24b] Google. Maps static api overview, 2024. [Online; accessed 01-July-2024].