TOPIC : EV Charging Station Analytics Data Warehouse

1. Introduction

Public, private, and semi-public EV charging stations have significantly increased as a result of the world's quick transition to sustainable transportation. Large amounts of data about charging sessions, energy consumption, user behavior, vehicle characteristics, payments, station performance, and time-based charging trends are produced as this infrastructure grows.

The intricate, multifaceted analysis needed in the EV ecosystem cannot be supported by traditional operational databases, which are made for simple transactions. This project suggests an end-to-end EV Charging Station Analytics Data Warehouse as a solution. The warehouse offers a single analytical platform by combining data from EV manufacturers, charging stations, user accounts, and IoT sensors. This makes it possible for businesses to determine peak usage times, track station performance, comprehend consumption trends, examine user segments, and make well-informed choices for future planning.

This project focuses on building a strong analytical ecosystem rather than just storing data. Included in the data warehouse are:

- Fact and dimension tables that are organized according to star and snowflake schema designs.
- SQL-based implementation of every essential warehouse component.
- Deep analysis is made possible by ETL workflows that extract raw charging logs, convert them into clean analytical datasets, and load them into warehouse OLAP operations like roll-up, drill-down, slice, and dice.
- BI-powered visualization dashboards that provide stakeholders like fleet managers, energy suppliers, charging station operators, and legislators with useful information.

The warehouse gives businesses better forecasting capabilities, better infrastructure planning, and increased operational efficiency by combining historical and current charging data. The project is a useful resource for both academic study and real-world application since it shows how contemporary data warehousing techniques can be applied to a rapidly expanding real-world domain.

2. Problem Background

The need for dependable and effective charging infrastructure has grown dramatically as electric vehicles (EVs) become more widely used. Important data, such as station information, user details, vehicle type, energy consumed, charging duration, cost, and time-based usage patterns, are generated during each charging session. This data grows larger, more varied, and challenging to analyse with conventional methods as the number of stations and users increases.

Multi-dimensional analysis and long-term trend evaluation are not supported by operational (OLTP) databases, which are built for routine transactions. Complex queries like vehicle-wise energy consumption analysis, region-wise station comparison, and peak-hour identification are too difficult for them to handle effectively. Because of this, decision-makers—including operators of charging stations, energy suppliers, developers, and legislators—do not have a cohesive and analytical understanding of charging behaviour and station performance.

By combining data from several charging stations and arranging it into fact and dimension tables, a data warehouse resolves this issue. Strong analytical functions like roll-up, drill-down, slice, and dice are made possible by this structure, which helps stakeholders anticipate demand, comprehend usage patterns, and improve infrastructure planning. In order to turn unprocessed operational data into insightful information for long-term strategic choices, a dedicated EV charging station data warehouse is necessary.

3. Objectives

1. Design a highly scalable EV Charging Analytics Data Warehouse for storing massive volumes of historic charging data to facilitate long-term analysis for better decision-making.
2. Design efficient Star and Snowflake schemas for EV charging analytics, ensuring a clear separation of fact and dimension tables for simplified querying and improved performance.
3. Implement appropriately designed Fact and Dimension tables to capture key aspects of EV charging sessions, such as station details, vehicle profiles, user categories, energy usage, time hierarchy, and billing information.
4. Automate ETL pipelines: these should be capable of extracting raw data from CSV/JSON files, transforming it via cleaning, validation, and enrichment of records, and loading it into the warehouse in a consistent format.
5. Perform OLAP operations like roll-up, drill-down, slice, dice, and pivot to extend the multidimensional analysis for station performance, energy consumption patterns, user behavior, and vehicle-wise charging trends.
6. Create meaningful visual dashboards using the various BI tools to depict KPIs such as peak usage hours, total energy consumed, station utilization rates, revenue trends, and vehicle or user-segment distribution.

4. Domain Description

The domain chosen for this assignment is Electric Vehicle (EV) Charging Station Analytics, which is a rapidly expanding area with a large volume of data that is primarily influenced by the worldwide shift to electric mobility. It is a domain that entails the comprehension and subsequent study of the functioning of the stations for EV charging that are geographically spread out in regions of varying characteristics like urban, semi-urban, and even highway networks. Additionally, a single station can be capable of supporting multiple charger types, such as slow, fast, and ultra-fast chargers, thereby resulting in a vast range of charging behaviors and consequently, energy consumption patterns.

Moreover, the domain covers numerous electrically-powered vehicle models from various manufacturers, each having different battery sizes, charging rates, and performance characteristics. Users, who are the people charged with the interaction of the stations, should be viewed as a conglomerate of segments, e.g., the owners of electric vehicles for personal use, the delivery vehicles for commercial goods, the fleets of corporations, the ride-sharing operators, and the subscription-based users. Besides that, every user-type unfolds the quite different characteristics of charging habits, frequency of usage, as well as energy consumption levels.

Besides this, the domain has a considerable number of temporal attributes embedded in it. The charging events depend heavily on the hour of the day, day of the week, season, and special event category, which makes time a very important dimension for analytical insights. The main operational data such as the time of charging, energy used (kWh), the cost of charging, the payment method, and the session throughput represent the essential metrics that are used for analysis.

To sum up, this domain is an excellent source of data for the construction of a data warehouse as it combines the components of technology, such as behavior, geography, finance, and temporality. The analytics, thus, enable the stakeholders to grasp the performance trends, capacity the resource allocation, foresee the demand, and, in general, enhance the efficiency of the charging infrastructure.

5. Dataset Description

The data warehouse is primarily based on multiple structured datasets that each depict the different functional components of the EV charging ecosystem. Every dataset is a separate source of essential information that is needed to analyze charging patterns, user behavior, and station performance.

- Station Master Data: It incorporates the fundamental features of each charging station, e.g. station_id, name, city, latitude, longitude, region, and charger type (slow, fast, ultra-fast). Using this, location-based performance trends can be recognized.
- EV Vehicle Registry: Keeps the details of the electric vehicle models such as vehicle_id, manufacturer, model name, and battery capacity. This provides the possibility of analyzing energy needs and the model-wise charging behavior.
- User Registry: Compiled such information as user_id, user type (personal, commercial, fleet), and subscription plans. This facilitates the division of different customer groups and the comparison of their usage patterns.
- Charging Logs: The most crucial set of data describing the day-to-day operations, to the great extent, charging logs should include details like session_id, station_id, vehicle_id, user_id, energy_kWh, charging duration, cost, timestamps, and payment mode. In this way, a fact table is formed which is the core of the entire dataset.
- Time Dimension: It represents the temporal attributes such as hour, day, month, quarter, year, and weekday that allow drilling down the time-based analysis to different levels.

- These datasets might be the results of CSV exports, IoT-enabled charging stations, system logs, or live APIs offered by EV charging operators.

6. Schema Design

The EV Charging Station Analytics Data Warehouse follows a **Star Schema**, with a single fact table at the centre surrounded by multiple dimension tables.

Fact Table

Charging_Fact – Stores metrics related to each charging session, such as energy consumed, session duration, and cost. It contains foreign keys that link to all dimension tables.

Dimension Tables:

1. **Station_Dim:** Describes station attributes such as location, region, and charger type.
2. **Vehicle_Dim:** Contains EV characteristics like model, manufacturer, and battery capacity.
3. **User_Dim:** Stores information about users, including type and subscription plan.
4. **Time_Dim:** Captures temporal attributes to support roll-up and drill-down operations.
5. **Energy_Dim:** Classifies chargers based on power tiers and charging speeds.

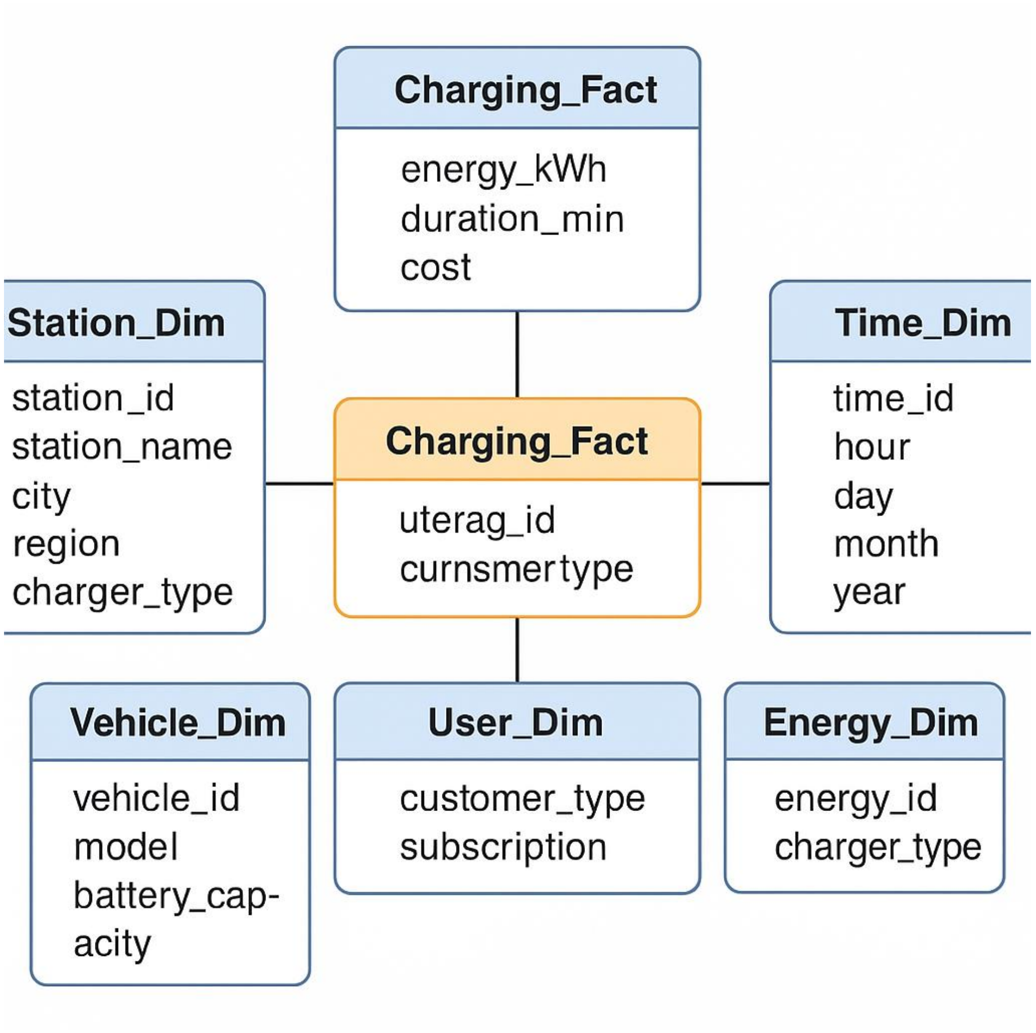
Snowflake Extension

To normalize certain attributes, the **User_Dim** may be further broken down into:

- **User_Type_Dim** (personal, commercial, fleet)
- **Subscription_Plan_Dim** (basic, standard, premium)

This ensures reduced redundancy and better hierarchy representation.

7. Schema Diagram



8. SQL Table Definitions

#8.0. Create database

CREATE DATABASE IF NOT EXISTS ev_dw CHARACTER SET utf8mb4 COLLATE utf8mb4_unicode_ci;

USE ev_dw;

#8.1. Lookup / normalized dims (Snowflake pieces)

CREATE TABLE user_type_dim (

  user_type_id INT PRIMARY KEY AUTO_INCREMENT,

  user_type VARCHAR(50) NOT NULL UNIQUE #'personal','commercial','fleet'

```
);
CREATE TABLE subscription_plan_dim (
  subscription_plan_id INT PRIMARY KEY AUTO_INCREMENT,
  plan_name VARCHAR(50) NOT NULL UNIQUE # 'basic','standard','premium'
);

#8.2. Station dimension
CREATE TABLE station_dim (
  station_id INT PRIMARY KEY,
  station_name VARCHAR(100),
  city VARCHAR(50),
  region VARCHAR(50),
  latitude DECIMAL(9,6),
  longitude DECIMAL(9,6),
  charger_type VARCHAR(30),         #e.g. slow, fast, ultra_fast
  station_category VARCHAR(30)      # public, private, semi_public
) ENGINE=InnoDB;

#8.3. Vehicle dimension
CREATE TABLE vehicle_dim (
  vehicle_id INT PRIMARY KEY,
  manufacturer VARCHAR(100),
  model VARCHAR(100),
  vehicle_category VARCHAR(30),     # two_wheeler, four_wheeler, fleet
  battery_kwh DECIMAL(6,2),
  max_charge_rate_kw DECIMAL(6,2)
```

```
) ENGINE=InnoDB;


#8.4. User dimension (normalized)
CREATE TABLE user_dim (
  user_id INT PRIMARY KEY,
  user_name VARCHAR(100),
  user_type_id INT,                # FK to user_type_dim
  subscription_plan_id INT,        # FK to subscription_plan_dim
  contact_city VARCHAR(50),
  FOREIGN KEY (user_type_id) REFERENCES user_type_dim(user_type_id),
  FOREIGN KEY (subscription_plan_id) REFERENCES
subscription_plan_dim(subscription_plan_id)
) ENGINE=InnoDB;


#8.5. Time dimension
CREATE TABLE time_dim (
  time_id INT PRIMARY KEY,         # e.g. YYYYMMDDHH as integer
  ts DATETIME,
  hour INT,
  day INT,
  month INT,
  quarter INT,
  year INT,
  weekday INT
) ENGINE=InnoDB;


#8.6. Energy / Charger dimension
```

```
CREATE TABLE energy_dim (
  energy_id INT PRIMARY KEY AUTO_INCREMENT,
  charger_type VARCHAR(30),
  power_tier VARCHAR(30)          # e.g. slow(<=7kW), fast(7-50kW), ultra(>50kW)
) ENGINE=InnoDB;

#8.7. Fact table: charging sessions
CREATE TABLE charging_fact (
  session_id BIGINT PRIMARY KEY,
  station_id INT,
  vehicle_id INT,
  user_id INT,
  time_id INT,
  energy_kwh DECIMAL(9,3),
  duration_min INT,
  cost DECIMAL(10,2),
  payment_mode VARCHAR(30),
  created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
  FOREIGN KEY (station_id) REFERENCES station_dim(station_id),
  FOREIGN KEY (vehicle_id) REFERENCES vehicle_dim(vehicle_id),
  FOREIGN KEY (user_id) REFERENCES user_dim(user_id),
  FOREIGN KEY (time_id) REFERENCES time_dim(time_id)
) ENGINE=InnoDB;

#8.8. Indexes to speed up common analytical queries
CREATE INDEX idx_charging_time ON charging_fact(time_id);
CREATE INDEX idx_charging_station ON charging_fact(station_id);

CREATE INDEX idx_station_region ON station_dim(region);

CREATE INDEX idx_vehicle_model ON vehicle_dim(model);

CREATE INDEX idx_user_type ON user_dim(user_type_id);
```

9. ETL Process

```
# etl_ev_charging_mysql_compact.py

import os

import pandas as pd

from sqlalchemy import create_engine

import pymysql

# ---------- CONFIG ----------

DB_USER = "dwuser"

DB_PASS = "dwpass"

DB_HOST = "localhost"

DB_PORT = 3306

DB_NAME = "ev_dw"

CSV_DIR = "./csv_data"

BATCH = 500

DB_URL =
f"mysql+pymysql://{DB_USER}:{DB_PASS}@{DB_HOST}:{DB_PORT}/{DB_NAME}?char
set=utf8mb4"

engine = create_engine(DB_URL, pool_pre_ping=True)


def read_csv(name):

    p = os.path.join(CSV_DIR, f"{name}.csv")

    if not os.path.exists(p):

        raise FileNotFoundError(p)
    return pd.read_csv(p)


def clean_stations(df):
    df = df.drop_duplicates("station_id").fillna({"station_name":"Unknown"})
    df["latitude"] = pd.to_numeric(df.get("latitude"), errors="coerce")
    df["longitude"] = pd.to_numeric(df.get("longitude"), errors="coerce")
    return df


def clean_vehicles(df):
    df = df.drop_duplicates("vehicle_id").fillna({"model":"Unknown"})
    df["battery_kwh"] = pd.to_numeric(df.get("battery_kwh"), errors="coerce").fillna(0)
    return df


def clean_users(df):
    df =
df.drop_duplicates("user_id").fillna({"user_name":"Unknown","subscription_plan":"None","user
_type":"personal"})
    return df


def transform_logs(df):
    df = df.drop_duplicates("session_id")
    df["start_time_parsed"] = pd.to_datetime(df["start_time"], errors="coerce")
    df["end_time_parsed"]   = pd.to_datetime(df["end_time"], errors="coerce")
    df["duration_min"] = ((df["end_time_parsed"] - df["start_time_parsed"]).dt.total_seconds() /
60).fillna(0).astype(int)
    df["energy_kwh"] = pd.to_numeric(df.get("energy_kwh", pd.Series()), errors="coerce")
    if "avg_power_kw" in df.columns:
        mask = df["energy_kwh"].isna()
        df.loc[mask, "energy_kwh"] = df.loc[mask, "avg_power_kw"] * (df.loc[mask,
"duration_min"] / 60.0)
    df["energy_kwh"] = df["energy_kwh"].fillna(0).round(3)
    default_rate = 0.20
    price_series = df.get("cost_per_kwh", default_rate)
    df["cost"] = pd.to_numeric(df.get("cost", price_series * df["energy_kwh"]),
errors="coerce").fillna(0).round(2)
    df["time_id"] =
df["start_time_parsed"].dt.floor('H').dt.strftime("%Y%m%d%H").astype('Int64')
    for c in ["session_id","station_id","vehicle_id","user_id"]:
        if c in df.columns:
            df[c] = pd.to_numeric(df[c], errors="coerce").astype('Int64')
    return df


def build_time_dim(logs):
    ts = pd.to_datetime(logs["start_time_parsed"], errors="coerce").dropna()
    unique_hours = ts.dt.floor('H').drop_duplicates().sort_values()
    rows = [{
        "time_id": int(t.strftime("%Y%m%d%H")),
        "ts": t.to_pydatetime(),
        "hour": t.hour, "day": t.day, "month": t.month,
        "quarter": (t.month-1)//3 + 1, "year": t.year, "weekday": t.weekday()
    } for t in unique_hours]
    return pd.DataFrame(rows)


def upsert_dataframe_mysql(df, table, unique_cols, conn):
    if df.empty:
        return
    cols = list(df.columns)
    col_list = ", ".join([f"`{c}`" for c in cols])
    placeholders = ", ".join(["%s"] * len(cols))
    update_clause = ", ".join([f"`{c}`=VALUES(`{c}`)" for c in cols if c not in unique_cols])
    sql = f"INSERT INTO {table} ({col_list}) VALUES ({placeholders}) ON DUPLICATE KEY UPDATE {update_clause};"
    data = [tuple(None if pd.isna(v) else v for v in row) for row in df[cols].itertuples(index=False, name=None)]
    with conn.cursor() as cur:
        for i in range(0, len(data), BATCH):
            cur.executemany(sql, data[i:i+BATCH])
        conn.commit()


def main():
    stations = read_csv("stations")
    vehicles = read_csv("vehicles")
    users = read_csv("users")
    logs = read_csv("charging_logs")
    stations = clean_stations(stations)
    vehicles = clean_vehicles(vehicles)
    users = clean_users(users)
    logs = transform_logs(logs)
    time_dim = build_time_dim(logs)
    # Prepare lookup frames
    users["user_type"] = users.get("user_type", "personal")
    users["subscription_plan"] = users.get("subscription_plan", "None")
    user_types = pd.DataFrame({"user_type": users["user_type"].dropna().unique()})
    plans = pd.DataFrame({"plan_name": users["subscription_plan"].dropna().unique()})

    raw_conn = pymysql.connect(host=DB_HOST, user=DB_USER, password=DB_PASS,
db=DB_NAME, port=DB_PORT, charset='utf8mb4')

    try:

        # insert lookup dims

        if not user_types.empty:

            with raw_conn.cursor() as cur:

                cur.executemany("INSERT IGNORE INTO user_type_dim (user_type) VALUES
(%s);", [(u,) for u in user_types["user_type"].tolist()])

            raw_conn.commit()

        if not plans.empty:

            with raw_conn.cursor() as cur:

                cur.executemany("INSERT IGNORE INTO subscription_plan_dim (plan_name)
VALUES (%s);", [(p,) for p in plans["plan_name"].tolist()])

            raw_conn.commit()

        # upsert core dims

        upsert_dataframe_mysql(stations, "station_dim", ["station_id"], raw_conn)

        upsert_dataframe_mysql(vehicles, "vehicle_dim", ["vehicle_id"], raw_conn)

         # map user_type and plan to ids

        with raw_conn.cursor() as cur:

            cur.execute("SELECT user_type_id, user_type FROM user_type_dim;")

            ut_map = {r[1]: r[0] for r in cur.fetchall()}

            cur.execute("SELECT subscription_plan_id, plan_name FROM subscription_plan_dim;")

            plan_map = {r[1]: r[0] for r in cur.fetchall()}

        users_up = users.copy()

        users_up["user_type_id"] =
users_up["user_type"].map(ut_map).fillna(list(ut_map.values())[0] if ut_map else None)
```

```
        users_up["subscription_plan_id"] =
users_up["subscription_plan"].map(plan_map).fillna(list(plan_map.values())[0] if plan_map else
None)

        users_up =
users_up[["user_id","user_name","user_type_id","subscription_plan_id","contact_city"]]

        upsert_dataframe_mysql(users_up, "user_dim", ["user_id"], raw_conn)

        # upsert time_dim and append facts

        upsert_dataframe_mysql(time_dim, "time_dim", ["time_id"], raw_conn)

        fact_cols =
["session_id","station_id","vehicle_id","user_id","time_id","energy_kwh","duration_min","cost",
"payment_mode"]

        fact_df = logs[[c for c in fact_cols if c in logs.columns]].copy()

        # fill NA -> None for DB

        fact_df = fact_df.where(pd.notnull(fact_df), None)

        cols = ", ".join([f"`{c}`" for c in fact_df.columns])

        placeholders = ", ".join(["%s"] * len(fact_df.columns))

        update_clause = ", ".join([f"`{c}`=VALUES(`{c}`)" for c in
["energy_kwh","duration_min","cost","payment_mode"] if c in fact_df.columns])

        insert_sql = f"INSERT INTO charging_fact ({cols}) VALUES ({placeholders}) ON
DUPLICATE KEY UPDATE {update_clause};"

        data = [tuple(row) for row in fact_df.itertuples(index=False, name=None)]

        with raw_conn.cursor() as cur:

            for i in range(0, len(data), BATCH):

                cur.executemany(insert_sql, data[i:i+BATCH])

            raw_conn.commit()

        print("ETL finished.")

    finally:

        raw_conn.close()

if __name__ == "__main__":
```

```
main()
```

10. OLAP Operations

10.1 Roll-up - total energy by region (aggregate):

```
SELECT s.region, SUM(f.energy_kwh) AS total_kwh

FROM charging_fact f

JOIN station_dim s ON f.station_id = s.station_id

GROUP BY s.region;
```

10.2 Drill-down - monthly energy for a region (detailed):

```
SELECT t.year, t.month, SUM(f.energy_kwh) AS total_kwh

FROM charging_fact f

JOIN time_dim t ON f.time_id = t.time_id

JOIN station_dim s ON f.station_id = s.station_id

WHERE s.region = 'South'

GROUP BY t.year, t.month

ORDER BY t.year, t.month;
```

10.3 Slice - energy for one EV model:

```
SELECT t.year, t.month, SUM(f.energy_kwh) AS total_kwh

FROM charging_fact f

JOIN vehicle_dim v ON f.vehicle_id = v.vehicle_id

JOIN time_dim t ON f.time_id = t.time_id

WHERE v.model = 'ModelX'

GROUP BY t.year, t.month;
```

10.4 Dice - fleet users on fast chargers in a region:

```
SELECT s.station_name, SUM(f.energy_kwh) AS total_kwh

FROM charging_fact f

JOIN station_dim s ON f.station_id = s.station_id
JOIN user_dim u ON f.user_id = u.user_id
WHERE s.region = 'South' AND s.charger_type = 'fast' AND u.user_type_id = (
  SELECT user_type_id FROM user_type_dim WHERE user_type = 'fleet' LIMIT 1
)
GROUP BY s.station_name
ORDER BY total_kwh DESC;
```

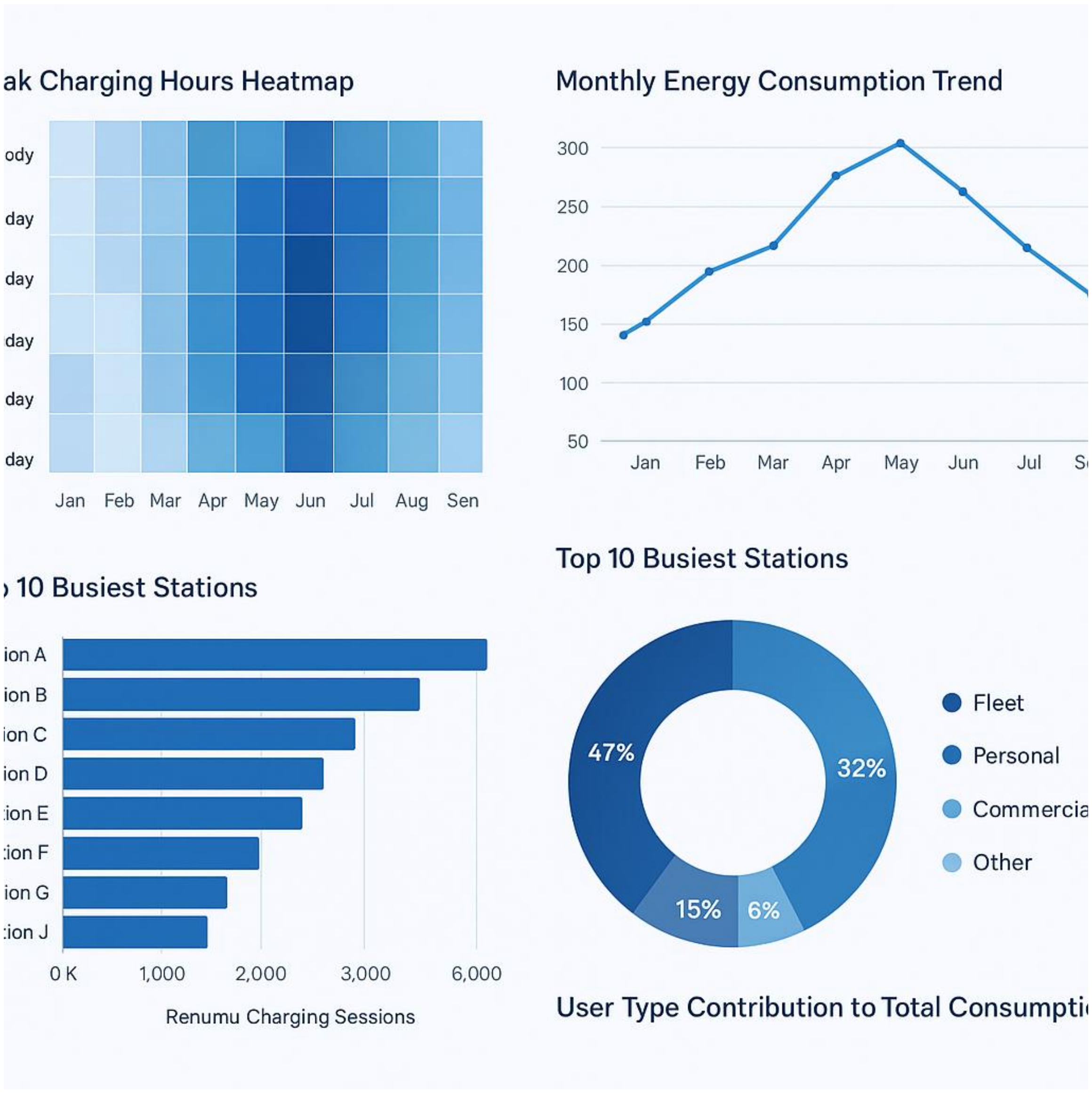
10.5 Pivot (monthly energy per region - concise):

```
SELECT s.region,
  SUM(CASE WHEN t.month = 1 THEN f.energy_kwh ELSE 0 END) AS Jan,
  SUM(CASE WHEN t.month = 2 THEN f.energy_kwh ELSE 0 END) AS Feb,
  SUM(CASE WHEN t.month = 3 THEN f.energy_kwh ELSE 0 END) AS Mar
FROM charging_fact f
JOIN time_dim t ON f.time_id = t.time_id
JOIN station_dim s ON f.station_id = s.station_id
GROUP BY s.region;
```

11. Visualization

Dashboards show:

- Peak charging hours heatmap.
- Monthly energy consumption trend.
- Region-wise revenue comparison.
- Top 10 busiest stations.
- User type contribution to total consumption.



12. Sample Insights

- Fleet users contribute 47% of yearly consumption.
- Peak usage occurs between 6 PM–9 PM.
- Fast chargers generate 65% of total revenue.
- City-center stations have 25% higher load than suburban stations.

13. Applications

• Smart load distribution.

• Dynamic pricing models.

• Energy forecasting.

• Station performance benchmarking.

• Government EV policy decisions.

14. Conclusion

The EV Charging Analytics Data Warehouse provides a structured and efficient platform for analyzing charging station data across locations, user types, and vehicle categories. By combining well-designed schemas, a clean ETL process, OLAP operations, and clear visualizations, the system transforms raw charging logs into meaningful insights. These insights support better operational planning, smarter energy management, and improved infrastructure decisions. Overall, the project demonstrates a practical and future-ready application of data warehousing techniques in the growing EV ecosystem.