Business Problem

Project name (recommended): GPS Fleet Data - Driver Risk & Route Anomaly Detection
Alternative short names: DriverRisk-DBRA24, Fleet Anomalies-Snowflake

Main business problem :
Build a system that identifies risky drivers and flags anomalous trips so fleet managers can prioritize interventions and reduce incident risk.

What business issues this solves:

- proactive safety monitoring (prevent accidents)
- operational cost savings (less fuel/wear from risky driving)
- compliance & route theft/geofence alerting
- performance-based driver coaching and incentives

Secondary analyses (add-on):

1. Route deviation analysis and hotspot mapping (why did deviation happen: traffic, weather, etc.)
2. Driver behavioral consistency score & trend (who's improving/worsening)
3. Clustering of trip types or driver segments (group similar driving patterns)

Blueprint

Business Problem (unchanged): Build a system that identifies risky drivers and flags anomalous trips so fleet managers can prioritize interventions and reduce incident risk.

OVERVIEW — What we will produce (concrete deliverables)

1. `fleet_db.raw.trips_raw` — original file(s) copied to RAW (exact CSV content).
2. `fleet_db.clean.trips_clean` — canonical table with correct types, parsed timestamp, cleaned categories.
3. `fleet_db.analytics.driver_daily` — driver-day aggregated features (used for ML + dashboards).
4. `fleet_db.analytics.trip_summary` — trip-level summary with route deviation & anomaly flags.
5. Colab notebook `model_training.ipynb` — EDA, features, baseline + XGBoost training, evaluation, predictions.
6. `fleet_db.analytics.predictions` — trip-level predicted risk probability + label.
7. Power BI pbix (or screenshots) — executive dashboard (leaderboard, map, time-series, drill-down).
8. GitHub repo with SQL scripts, Colab notebook, pbix/screenshots, README, small sample CSV.

STAGE A — INITIAL NAVIGATION IN SNOWSIGHT (exact steps)

1. Open Chrome → login to Snowflake → left sidebar → **Query data** → **Worksheets** → **+ Worksheet**.
2. Top bar of Worksheet: set **Role = ACCOUNTADMIN** (or SYSADMIN if preferred), **Warehouse = GPSFLEET_WH**, leave Database/Schema blank until created.

3. Create Database / Schemas (paste and run):

```
CREATE OR REPLACE DATABASE fleet_db;
CREATE OR REPLACE SCHEMA fleet_db.raw;
CREATE OR REPLACE SCHEMA fleet_db.clean;
CREATE OR REPLACE SCHEMA fleet_db.analytics;
```

4. Create file format and stage (paste and run):

```
CREATE OR REPLACE FILE FORMAT csv_fmt
  TYPE = 'CSV'
  FIELD_OPTIONALLY_ENCLOSED_BY = '"'
  SKIP_HEADER = 1;

CREATE OR REPLACE STAGE my_stage FILE_FORMAT = csv_fmt;
```

Why: Stage gives you a safe place for raw files so you can reload/reprocess. File format tells Snowflake how to parse CSV.

STAGE B — UPLOAD CSV (UI steps) + VERIFY

1. Left sidebar → **Data** → expand `Stages` → click `MY_STAGE` → **Upload** → select CSV from your laptop.

2. After upload, verify:

```
LIST @my_stage;
```

3. Quick preview (first 10 rows):

```
SELECT $1,$2,$3,$4,$5
FROM @my_stage (FILE_FORMAT => 'csv_fmt') LIMIT 10;
```

Why: Ensure file is present and parsed; `$1` etc are the raw columns. If header rows mismatch, adjust `SKIP_HEADER`.

STAGE C — CREATE RAW TABLE (exact SQL)

Key principle: RAW = a faithful copy of the CSV (minimal transformation). Use STRING where unsure.

```
CREATE OR REPLACE TABLE fleet_db.raw.trips_raw (
  trip_id STRING,
```

```
  driver_id STRING,
  vehicle_id STRING,
  timestamp STRING,
  latitude STRING,
  longitude STRING,
  speed STRING,
  acceleration STRING,
  steering_angle STRING,
  heading STRING,
  trip_duration STRING,
  trip_distance STRING,
  fuel_consumption STRING,
  rpm STRING,
  brake_usage STRING,
  lane_deviation STRING,
  weather_conditions STRING,
  road_type STRING,
  traffic_condition STRING,
  stop_events STRING,
  geofencing_violation STRING,
  anomalous_event STRING,
  route_anomaly STRING,
  route_deviation_score STRING,
  acceleration_variation STRING,
  behavioral_consistency_index STRING
);
```

Then load from stage (replace your filename):

```
COPY INTO fleet_db.raw.trips_raw FROM @my_stage/your_file.csv
FILE_FORMAT=(FORMAT_NAME='csv_fmt') ON_ERROR='CONTINUE';
```

Why RAW as STRING: CSV carries no types; keeping RAW non-strict preserves raw input so you can re-run transformations later without risking load failures.

STAGE D — CLEAN TABLE (exact SQL with explanations inline)

Goal: Convert strings → correct datatypes, parse timestamps, standardize categories, map 0/1 to booleans, handle missing values. This is the canonical table for analysis and ML.

Paste the block and run:

```
CREATE OR REPLACE TABLE fleet_db.clean.trips_clean AS
```

```
SELECT
  TRIM(trip_id) AS trip_id,
  TRIM(driver_id) AS driver_id,
  TRIM(vehicle_id) AS vehicle_id,
  -- parse timestamp (adjust format if needed)
  TRY_TO_TIMESTAMP(timestamp) AS ts,
  TRY_TO_DOUBLE(latitude) AS latitude,
  TRY_TO_DOUBLE(longitude) AS longitude,
  TRY_TO_DOUBLE(speed) AS speed,
  TRY_TO_DOUBLE(acceleration) AS acceleration,
  TRY_TO_DOUBLE(steering_angle) AS steering_angle,
  TRY_TO_DOUBLE(heading) AS heading,
  TRY_TO_DOUBLE(trip_duration) AS trip_duration,
  TRY_TO_DOUBLE(trip_distance) AS trip_distance,
  TRY_TO_DOUBLE(fuel_consumption) AS fuel_consumption,
  TRY_TO_DOUBLE(rpm) AS rpm,
  -- brake_usage is count → integer
  TRY_TO_INT(brake_usage) AS brake_usage,
  TRY_TO_DOUBLE(lane_deviation) AS lane_deviation,
  LOWER(TRIM(weather_conditions)) AS weather_conditions,
  LOWER(TRIM(road_type)) AS road_type,
  LOWER(TRIM(traffic_condition)) AS traffic_condition,
  TRY_TO_INT(stop_events) AS stop_events,
  -- booleans mapped from 1/0 or 'true'/'false'
  IFF(TRY_TO_INT(geofencing_violation)=1, TRUE, FALSE) AS geofencing_violation,
  IFF(TRY_TO_INT(anomalous_event)=1, TRUE, FALSE) AS anomalous_event,
```

IFF(TRY_TO_INT(route_anomaly)=1, TRUE, FALSE) AS route_anomaly,

TRY_TO_DOUBLE(route_deviation_score) AS route_deviation_score,

TRY_TO_DOUBLE(acceleration_variation) AS acceleration_variation,

TRY_TO_DOUBLE(behavioral_consistency_index) AS behavioral_consistency_index

FROM fleet_db.raw.trips_raw;

How this helps / why each transform:

- `TRY_TO_TIMESTAMP` → timestamps are essential for temporal aggregation; using TRY avoids errors if format differs.
- `TRY_TO_DOUBLE` / `TRY_TO_INT` → numeric conversions allow aggregation, statistical metrics, and ML features.
- `LOWER(TRIM(...))` → category standardization prevents duplicates like `Highway` vs `highway`.
- `IFF(...)=1` → convert 0/1 strings to booleans for logic & ML labels.

Validation queries to run next:

SELECT COUNT(*) FROM fleet_db.clean.trips_clean;

SELECT COUNT(CASE WHEN ts IS NULL THEN 1 END) AS null_timestamps FROM fleet_db.clean.trips_clean;

SELECT MIN(ts), MAX(ts) FROM fleet_db.clean.trips_clean;

SELECT DISTINCT weather_conditions, road_type, traffic_condition FROM fleet_db.clean.trips_clean LIMIT 50;

STAGE E — ANALYTICS TABLES (exact SQL + WHY & Analysis Questions)

We create both trip-level summary and driver-day aggregates. These are what you'll pull into Colab and use in Power BI.

1) Trip summary (one row per trip)

CREATE OR REPLACE TABLE fleet_db.analytics.trip_summary AS

```
SELECT
  trip_id,
  driver_id,
  vehicle_id,
  DATE_TRUNC('day', ts) AS day,
  MIN(ts) AS trip_start_ts,
  MAX(ts) AS trip_end_ts,
  MAX(trip_distance) AS trip_distance,
  MAX(trip_duration) AS trip_duration,
  AVG(speed) AS avg_speed,
  MAX(speed) AS max_speed,
  SUM(brake_usage) AS total_brake_events,
  AVG(acceleration) AS avg_acceleration,
  MAX(route_deviation_score) AS max_route_deviation,
  BOOL_OR(anomalous_event) AS any_anomalous_event,
  BOOL_OR(route_anomaly) AS any_route_anomaly
FROM fleet_db.clean.trips_clean
GROUP BY trip_id, driver_id, vehicle_id, DATE_TRUNC('day', ts);
```

Why: Trip-level aggregates let us label trips and feed trip features to a trip-level ML model (predict anomalous_event).

Business questions supported: Which trips are high-risk? Which trips have high route deviation? Which trips require immediate review?

2) Driver daily aggregates (driver-day features suitable for ML)

```
CREATE OR REPLACE TABLE fleet_db.analytics.driver_daily AS
SELECT
  driver_id,
```

```
  day,
  AVG(avg_speed) AS avg_speed_day,
  STDDEV(avg_speed) AS sd_speed_day,
  SUM(total_brake_events) AS total_brakes_day,
  SUM(CASE WHEN any_anomalous_event THEN 1 ELSE 0 END) AS anomalies_count_day,
  AVG(max_route_deviation) AS avg_route_dev_day,
  COUNT(DISTINCT trip_id) AS trips_count_day,
  AVG(behavioral_consistency_index) AS avg_behavioral_index_day
FROM fleet_db.analytics.trip_summary
GROUP BY driver_id, day;
```

Why: Driver-day level smoothing reduces noise and gives a good level for interventions (rather than per-second telemetry). It supports trend detection and driver coaching.

Analysis questions supported: Who are the top-risk drivers this week? Who improved vs last month? Which drivers have frequent high braking events?

STAGE F — ANALYSIS QUESTIONS (exact list to answer by SQL / ML)

I'll give prioritized questions (must-do) and deep-dive questions (secondary). Each has the recommended location (SQL in Snowflake vs Colab Python) and why.

MUST-DO (supports business objectives)

1. **Top risky drivers:** For last 30 days, rank drivers by average daily anomaly probability (SQL using `driver_daily`). — *Snowflake / Power BI*
2. **Top risky trips:** Top 100 trips with highest `max_route_deviation` and `total_brake_events`. — *Snowflake / Power BI*
3. **Anomaly trend:** Time series of daily anomalies count (7-day moving average). — *SQL + Power BI*

4. **Map flagged trips:** Map lat/lon for trips with `any_anomalous_event = TRUE`. — *Power BI*

ML-focused (Colab)

5. **Trip-level anomaly classification**: Predict `any_anomalous_event` using trip_summary features (XGBoost). Evaluate AUC, Precision@K. — *Colab*

6. **Driver risk regression (optional):** Build a risk score per driver-day (0–100) using aggregated features. — *Colab*

7. **Unsupervised route anomalies:** Use IsolationForest on route_deviation_score, heading, speed variance to find unlabeled anomalies. — *Colab*

Secondary deep-dive (add-ons)

8. Hotspot analysis: geospatial clustering of route deviations by area/time (DBSCAN). — *Colab → Power BI map*

9. Behavioral consistency trend: compute week-over-week delta in `avg_behavioral_index`. — *SQL + Power BI*

10. Clustering drivers into segments (defensive, risky, normal). — *Colab*

Why this ordering: Start with SQL answers and dashboards to show immediate business value. Then do ML modeling to improve precision and add predictive alerts.

STAGE G — PULL DATA TO COLAB (exact steps & code)

Options to pull:

- **Recommended:** Snowflake connector in Colab (direct, reproducible).
- **Alternate:** Download CSV from Snowflake UI and upload to Colab (manual).

Colab setup & connect (copy-paste):

in Colab cell 1

!pip install snowflake-connector-python pandas scikit-learn xgboost joblib folium

```
# in Colab cell 2 - connect
import snowflake.connector, pandas as pd
conn = snowflake.connector.connect(
    user='YOUR_USER',
    password='YOUR_PASSWORD',
    account='YOUR_ACCOUNT',   # e.g. xy12345.ap-south-1
    warehouse='GPSFLEET_WH',
    database='FLEET_DB',
    schema='ANALYTICS'
)
sql = "SELECT * FROM trip_summary WHERE day >= '2023-01-01';"
df_trip = pd.read_sql(sql, conn)
print(df_trip.shape)
df_trip.head()
```

Why connector: You get up-to-date tables, and your notebook is reproducible. Use Colab secrets for credentials (recommended).

STAGE H — FEATURE ENGINEERING & MODEL (Colab code skeleton)

Feature ideas (copy to Colab):

- `speed_stats`: avg_speed, max_speed, sd_speed.
- `brake_rate`: total_brake_events / trip_duration or per km.

- `route_dev_norm`: route_deviation_score normalized by trip_distance.
- `time_of_day_bin`: morning/afternoon/night.
- `weather_flag`: map weather_conditions → severity score.

Colab snippet (feature creation):

```
import numpy as np

df = df_trip.copy()

df['brake_per_km'] = df['total_brake_events'] / (df['trip_distance'].replace(0,np.nan))

df['route_dev_per_km'] = df['max_route_deviation'] / (df['trip_distance'].replace(0,np.nan))

df['hour'] = pd.to_datetime(df['trip_start_ts']).dt.hour

df['time_of_day'] = pd.cut(df['hour'], bins=[-1,6,12,18,24],
labels=['night','morning','afternoon','evening'])

df = df.fillna(0)

feature_cols =
['avg_speed','max_speed','brake_per_km','route_dev_per_km','avg_acceleration']  # expand
as needed

X = df[feature_cols]

y = df['any_anomalous_event'].astype(int)
```

Train/test + model (XGBoost):

```
from sklearn.model_selection import train_test_split

from xgboost import XGBClassifier

from sklearn.metrics import roc_auc_score, precision_recall_fscore_support


X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42,
stratify=y)

model = XGBClassifier(n_estimators=200, max_depth=6, learning_rate=0.08,
use_label_encoder=False, eval_metric='logloss')
```

```
model.fit(X_train, y_train, early_stopping_rounds=10, eval_set=[(X_test,y_test)],
verbose=False)

y_pred_proba = model.predict_proba(X_test)[:,1]

print("AUC:", roc_auc_score(y_test, y_pred_proba))
```

Why XGBoost: Strong performance on tabular data, handles missing values, commonly used in industry.

STAGE I — PREDICTIONS → WRITE BACK (two options)

Option A (programmatic, recommended): write_pandas

```
from snowflake.connector.pandas_tools import write_pandas

df_preds = df.loc[X_test.index, ['trip_id','driver_id']].copy()

df_preds['pred_prob'] = model.predict_proba(X_test)[:,1]

df_preds['pred_label'] = (df_preds['pred_prob']>0.7).astype(int)

# write to analytic schema

success, nchunks, nrows, _ = write_pandas(conn, df_preds, 'predictions',
schema='ANALYTICS')

print(success, nrows)
```

Option B (manual): save CSV and upload to `my_stage`, then `COPY INTO fleet_db.analytics.predictions`.

SQL to create predictions table in Snowflake (if not exists):

```
CREATE OR REPLACE TABLE fleet_db.analytics.predictions (

  trip_id STRING,

  driver_id STRING,

  pred_prob DOUBLE,

  pred_label INT
```

);

Why writing back: Keeps all artifacts in one place for Power BI to consume and for reproducibility in interviews.

STAGE J — POWER BI DASHBOARD (step-by-step)

1. Open Power BI Desktop → Get Data → Snowflake.
2. Server: `<your_account>.snowflakecomputing.com` → Warehouse: `GPSFLEET_WH` → Database: `FLEET_DB` → Schema: `ANALYTICS`. Use Import for dev.
3. Load tables: `driver_daily`, `trip_summary`, `predictions`.
4. Build visuals:
    - Executive page: top 10 risky drivers (bar chart using driver_daily anomaly count or avg pred_prob), total anomalies (card), % drivers flagged.
    - Map page: scatter map of trip_summary where any_anomalous_event or pred_label=1; size by route_deviation_score, color by pred_prob.
    - Trend page: line chart of anomalies_count_day (7-day SMA).
    - Drill-down: click a driver → show his trips (table) and sample telemetry (if desired).
5. Save pbix and export screenshots.

Why Power BI: Visualizes the results for fleet managers; interactive filters enable operational actions.

STAGE K — BASIC PIPELINE (automation, simple)

We want a reproducible, minimal pipeline without heavy infra:

Manual MVP pipeline:

- Step A: Upload CSV to `my_stage` (every day or batch).
- Step B: Run `COPY INTO` to load RAW (or have a Snowflake Task run a SQL script).
- Step C: Run CLEAN SQL to rebuild `trips_clean` (or incremental logic).
- Step D: Trigger Colab notebook manually (or schedule via GitHub Actions) to fetch `trip_summary`, run model scoring, write predictions back.
- Step E: Power BI dataset refresh (manual or scheduled in Power BI Service with gateway).

Simple automation option (later):

- Use Snowflake **Tasks** + **Streams** to detect new files and run `COPY` / `CLEAN` steps. Keep Colab scoring as scheduled notebook (cron or GitHub Actions) until you do Snowpark.

Why this pipeline: Minimal infra, reproducible steps, easy to demo in interviews.

STAGE L — MODEL EVALUATION & BUSINESS KPIs (what to measure)

For classification (trip anomaly):

- **AUC-ROC** (global separability).
- **Precision@K / Top-k precision**: important because fleet managers act on top N flagged trips.
- **Recall for critical events**: how many real anomalies we caught.
- **False positive review cost**: estimate minutes per false positive × number of false positives.

For driver scoring:

- **Driver Risk Score distribution**: top 5% drivers vs rest.
- **Week-over-week reduction** after interventions (if you have time-series).

Why: Business cares about actionability (precision at top) and operational cost of false alarms.

STAGE M — GITHUB + RESUME + LINKEDIN

Repo structure (exact):

FleetSafe-DriverRisk/

├─ data/sample_snapshot.csv

├─ sql/00_setup.sql

├─ sql/01_load_raw.sql

├─ sql/02_create_clean.sql

├─ sql/03_analytics_rollups.sql

├─ notebooks/model_training.ipynb

├─ powerbi/FleetSafe_Report.pbix

├─ docs/architecture.md

├─ visuals/dashboard_screenshots.png

└─ README.md

README content to include: Problem statement, architecture diagram, how to run (step-by-step), sample screenshots, links to pbix & notebook.

LinkedIn: Post a 2-paragraph write-up + 2 images (dashboard + model metric chart) + GitHub link. In Projects, add bullet with technologies used.

Snowflake trial note: Don’t rely on trial for public demo. Export sample CSV, pbix & notebook to GitHub; optionally host a small Streamlit demo separately (later).

STAGE N — CHECKLIST / SPRINT PLAN (copy into your task manager)

Week 1 — Setup & ingestion:

- Create DB/schemas/stage, upload CSV, load RAW, build CLEAN.
  Week 2 — Analytics & EDA:

- Build trip_summary, driver_daily; exploratory visualizations in Colab & SQL.
  Week 3 — Modeling:

- Feature engineering, baseline model (LogReg), XGBoost, metrics.
  Week 4 — Predictions & Dashboard:

- Writeback predictions, build Power BI dashboard, iterate thresholds.
  Week 5 — Polishing & portfolio:

- Clean repo, README, screenshots, LinkedIn post, short demo video.

STAGE O — EXACT SQL & PYTHON BLOCKS (all in one place for copy-paste)

SQL: create tables + load (summary) — see full blocks above (`CREATE TABLE RAW`, `COPY INTO`, `CREATE CLEAN`, `CREATE analytics.trip_summary`, `CREATE analytics.driver_daily`, `CREATE TABLE analytics.predictions`).

Colab: connect + feature + model + writeback — see the code snippets in Sections G & H & I.