```
!pip install snowflake-connector-python pandas scikit-learn xgboost
```

Show hidden output

```
import snowflake.connector
import pandas as pd
import numpy as np
```

```
conn = snowflake.connector.connect (
    user = "ARINDHAMFLAKE",
    password = "Dataanalyst2025",
    account = "ZIEPNFM-WH94884",
    warehouse = "GPSFLEET_WH",
    database = "FLEET_DB",
    schema = "ANALYTICS"
)
```

```
df_trip = pd.read_sql("SELECT * FROM trip_summary;", conn)
df_driver = pd.read_sql("SELECT * FROM driver_daily;", conn)
print(df_trip.shape, df_driver.shape)
```

```
/tmp/ipython-input-4157858477.py:1: UserWarning: pandas only supports SQLAlchemy connectable (engine/connection) or database
  df_trip = pd.read_sql("SELECT * FROM trip_summary;", conn)
/tmp/ipython-input-4157858477.py:2: UserWarning: pandas only supports SQLAlchemy connectable (engine/connection) or database
  df_driver = pd.read_sql("SELECT * FROM driver_daily;", conn)
(120000, 17) (10, 11)
```

```
df_trip.head()
```

	TRIP_ID	DRIVER_ID	VEHICLE_ID	DAY	TRIP_START_AT	TRIP_DURATION_MINUTES	TRIP_DISTANCE_KM	AVG_SPEED	MAX_SPEED	AVG_A
0	114387	101	1001	2023-01-02	2023-01-02 07:46:26	66.626622	4.558716	55.215226	55.215226	
1	114394	101	3003	2023-01-02	2023-01-02 07:46:33	36.829610	30.989605	51.353421	51.353421	
2	114401	101	4004	2023-01-02	2023-01-02 07:46:40	35.398257	25.775307	93.328352	93.328352	
3	114408	101	1001	2023-01-02	2023-01-02 07:46:47	53.200739	91.098530	7.532813	7.532813	
4	114415	103	3003	2023-01-02	2023-01-02 07:46:54	10.778823	112.873466	2.887324	2.887324	

```
df_driver.head()
```

	DRIVER_ID	DAY	TRIP_COUNT_DAILY	AVG_TRIP_DISTANCE_DAILY	AVG_SPEED_DAILY	TOTAL_BRAKE_EVENTS_DAILY	ANOMALIES_COUNT_D
0	102	2023-01-02	6740	50.300488	29.966187	33643	
1	105	2023-01-02	3274	49.032908	29.668213	16241	
2	104	2023-01-02	3468	49.872775	28.997656	17308	
3	101	2023-01-01	43094	49.866401	29.951755	215095	
4	103	2023-01-01	8621	49.175268	29.836446	43178	

```
df_trip.head()
```

	TRIP_ID	DRIVER_ID	VEHICLE_ID	DAY	TRIP_START_AT	TRIP_DURATION_MINUTES	TRIP_DISTANCE_KM	AVG_SPEED	MAX_SPEED	AVG_A
0	114387	101	1001	2023-01-02	2023-01-02 07:46:26	66.626622	4.558716	55.215226	55.215226	
1	114394	101	3003	2023-01-02	2023-01-02 07:46:33	36.829610	30.989605	51.353421	51.353421	
2	114401	101	4004	2023-01-02	2023-01-02 07:46:40	35.398257	25.775307	93.328352	93.328352	
3	114408	101	1001	2023-01-02	2023-01-02 07:46:47	53.200739	91.098530	7.532813	7.532813	
4	114415	103	3003	2023-01-02	2023-01-02 07:46:54	10.778823	112.873466	2.887324	2.887324	

```
1 df_trip.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 120000 entries, 0 to 119999
Data columns (total 17 columns):
 #   Column                        Non-Null Count   Dtype
---  ------                        --------------   -----
 0   TRIP_ID                       120000 non-null  object
 1   DRIVER_ID                     120000 non-null  object
 2   VEHICLE_ID                    120000 non-null  object
 3   DAY                           120000 non-null  datetime64[ns]
 4   TRIP_START_AT                 120000 non-null  datetime64[ns]
 5   TRIP_DURATION_MINUTES         120000 non-null  float64
 6   TRIP_DISTANCE_KM              120000 non-null  float64
 7   AVG_SPEED                     120000 non-null  float64
 8   MAX_SPEED                     120000 non-null  float64
 9   AVG_ACCELARATION              120000 non-null  float64
 10  TOTAL_BRAKE_EVENTS            120000 non-null  int64
 11  MAX_ROUTE_DEVIATION           120000 non-null  float64
 12  ANY_ANOMALOUS_EVENT           120000 non-null  bool
 13  ANY_GEOFENCING_VIOLATION      120000 non-null  bool
 14  ANY_ROUTE_ANOMALY             120000 non-null  bool
 15  AVG_ROUTE_DEVIATION_SCORE     120000 non-null  float64
 16  BEHAVIORAL_CONSISTENCY_INDEX  120000 non-null  float64
dtypes: bool(3), datetime64[ns](2), float64(8), int64(1), object(3)
memory usage: 13.2+ MB
```

```
1 display(df_trip.describe().T)
```

	count	mean	min	25%	50%	75%	
DAY	120000	2023-01-01 06:43:12	2023-01-01 00:00:00	2023-01-01 00:00:00	2023-01-01 00:00:00	2023-01-02 00:00:00	2
TRIP_START_AT	120000	2023-01-01 16:39:59.500000	2023-01-01 00:00:00	2023-01-01 08:19:59.750000128	2023-01-01 16:39:59.500000	2023-01-02 00:59:59.249999872	2
TRIP_DURATION_MINUTES	120000.0	59.891751	0.00003	17.212549	41.402746	82.931203	7
TRIP_DISTANCE_KM	120000.0	49.921316	0.000521	14.3195	34.387936	69.138769	5
AVG_SPEED	120000.0	29.971987	0.000157	8.61525	20.633107	41.439832	3
MAX_SPEED	120000.0	29.971987	0.000157	8.61525	20.633107	41.439832	3
AVG_ACCELARATION	120000.0	1.003904	-0.99991	-0.41903	0.388639	1.777851	1
TOTAL_BRAKE_EVENTS	120000.0	4.995767	0.0	3.0	5.0	6.0	
MAX_ROUTE_DEVIATION	120000.0	0.285187	0.000305	0.160065	0.263214	0.389066	
AVG_ROUTE_DEVIATION_SCORE	120000.0	0.285187	0.000305	0.160065	0.263214	0.389066	

```
1 df_trip.isnull().sum()
```

Show hidden output

```
1 df_trip['ANY_ANOMALOUS_EVENT'].value_counts()
```

ANY_ANOMALOUS_EVENT	count
False	107980
True	12020

dtype: int64

```
print(df_trip['ANY_ROUTE_ANOMALY'].value_counts())
print(df_trip['ANY_GEOFENCING_VIOLATION'].value_counts())
```

```
ANY_ROUTE_ANOMALY
False    108035
True      11965
Name: count, dtype: int64
ANY_GEOFENCING_VIOLATION
False    113971
True       6029
Name: count, dtype: int64
```

```
df_trip.columns
```

```
Index(['TRIP_ID', 'DRIVER_ID', 'VEHICLE_ID', 'DAY', 'TRIP_START_AT',
       'TRIP_DURATION_MINUTES', 'TRIP_DISTANCE_KM', 'AVG_SPEED', 'MAX_SPEED',
       'AVG_ACCELARATION', 'TOTAL_BRAKE_EVENTS', 'MAX_ROUTE_DEVIATION',
       'ANY_ANOMALOUS_EVENT', 'ANY_GEOFENCING_VIOLATION', 'ANY_ROUTE_ANOMALY',
       'AVG_ROUTE_DEVIATION_SCORE', 'BEHAVIORAL_CONSISTENCY_INDEX',
       'BRAKE_PER_KM', 'ROUTE_DEV_PER_KM', 'HOUR', 'TIME_OF_DAY'],
      dtype='object')
```

```
# brake events per km
df_trip['BRAKE_PER_KM'] = (df_trip['TOTAL_BRAKE_EVENTS'] /df_trip['TRIP_DISTANCE_KM'].replace(0, np.nan))
```

```
# route deviation per km
df_trip['ROUTE_DEV_PER_KM'] = (df_trip['MAX_ROUTE_DEVIATION'] /df_trip['TRIP_DISTANCE_KM'].replace(0, np.nan))
```

```
# hour of day
df_trip['TRIP_START_HOUR'] = df_trip['TRIP_START_AT'].dt.hour
```

```
# bucket time of day
df_trip['TIME_OF_DAY'] = pd.cut(df_trip['TRIP_START_HOUR'],bins=[-1, 6, 12, 18, 24],labels=['night', 'morning', 'aftern
```

```
df_trip.head()
```

	TRIP_ID	DRIVER_ID	VEHICLE_ID	DAY	TRIP_START_AT	TRIP_DURATION_MINUTES	TRIP_DISTANCE_KM	AVG_SPEED	MAX_SPEED	AVG_A
0	114387	101	1001	2023-01-02	2023-01-02 07:46:26	66.626622	4.558716	55.215226	55.215226	
1	114394	101	3003	2023-01-02	2023-01-02 07:46:33	36.829610	30.989605	51.353421	51.353421	
2	114401	101	4004	2023-01-02	2023-01-02 07:46:40	35.398257	25.775307	93.328352	93.328352	
3	114408	101	1001	2023-01-02	2023-01-02 07:46:47	53.200739	91.098530	7.532813	7.532813	
4	114415	103	3003	2023-01-02	2023-01-02 07:46:54	10.778823	112.873466	2.887324	2.887324	

5 rows × 22 columns

```
df_trip.describe().T
```

	count	mean	min	25%	50%	75%
DAY	120000	2023-01-01 06:43:12	2023-01-01 00:00:00	2023-01-01 00:00:00	2023-01-01 00:00:00	2023-01-02 00:00:00
TRIP_START_AT	120000	2023-01-01 16:39:59.500000	2023-01-01 00:00:00	2023-01-01 08:19:59.750000128	2023-01-01 16:39:59.500000	2023-01-02 00:59:59.249999872
TRIP_DURATION_MINUTES	120000.0	59.891751	0.00003	17.212549	41.402746	82.931203
TRIP_DISTANCE_KM	120000.0	49.921316	0.000521	14.3195	34.387936	69.138769
AVG_SPEED	120000.0	29.971987	0.000157	8.61525	20.633107	41.439832
MAX_SPEED	120000.0	29.971987	0.000157	8.61525	20.633107	41.439832
AVG_ACCELARATION	120000.0	1.003904	-0.99991	-0.41903	0.388639	1.777851
TOTAL_BRAKE_EVENTS	120000.0	4.995767	0.0	3.0	5.0	6.0
MAX_ROUTE_DEVIATION	120000.0	0.285187	0.000305	0.160065	0.263214	0.389066
AVG_ROUTE_DEVIATION_SCORE	120000.0	0.285187	0.000305	0.160065	0.263214	0.389066
BEHAVIORAL_CONSISTENCY_INDEX	120000.0	0.285855	0.000295	0.160986	0.264427	0.389694
BRAKE_PER_KM	120000.0	1.452215	0.0	0.061677	0.134322	0.337682
ROUTE_DEV_PER_KM	120000.0	0.093352	0.000001	0.003194	0.007321	0.01897
HOUR	120000.0	9.45	0.0	4.0	8.0	15.0

```
print(df_trip.columns)
```

```
Index(['TRIP_ID', 'DRIVER_ID', 'VEHICLE_ID', 'DAY', 'TRIP_START_AT',
       'TRIP_DURATION_MINUTES', 'TRIP_DISTANCE_KM', 'AVG_SPEED', 'MAX_SPEED',
       'AVG_ACCELARATION', 'TOTAL_BRAKE_EVENTS', 'MAX_ROUTE_DEVIATION',
       'ANY_ANOMALOUS_EVENT', 'ANY_GEOFENCING_VIOLATION', 'ANY_ROUTE_ANOMALY',
       'AVG_ROUTE_DEVIATION_SCORE', 'BEHAVIORAL_CONSISTENCY_INDEX',
       'BRAKE_PER_KM', 'ROUTE_DEV_PER_KM', 'HOUR', 'TIME_OF_DAY',
       'TRIP_START_HOUR'],
      dtype='object')
```

```
feature_cols = [
    'TRIP_DURATION_MINUTES',
    'TRIP_DISTANCE_KM',
    'AVG_SPEED',
    'MAX_SPEED',
    'AVG_ACCELARATION',
    'TOTAL_BRAKE_EVENTS',
    'MAX_ROUTE_DEVIATION',
    'AVG_ROUTE_DEVIATION_SCORE',
    'BEHAVIORAL_CONSISTENCY_INDEX',
    'BRAKE_PER_KM',
    'ROUTE_DEV_PER_KM',
    'TRIP_START_HOUR'
]

X = df_trip[feature_cols]
```

```
y = df_trip['ANY_ANOMALOUS_EVENT'].astype(int)

```

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y , test_size = 0.3, stratify = y , random_state = 42)
```

```
from xgboost import XGBClassifier
xg_model = XGBClassifier()
```

```
xg_model.fit(X_train , y_train)
```

```
XGBClassifier(base_score=None, booster=None, callbacks=None,
              colsample_bylevel=None, colsample_bynode=None,
              colsample_bytree=None, device=None, early_stopping_rounds=None,
              enable_categorical=False, eval_metric=None, feature_types=None,
              feature_weights=None, gamma=None, grow_policy=None,
              importance_type=None, interaction_constraints=None,
              learning_rate=None, max_bin=None, max_cat_threshold=None,
              max_cat_to_onehot=None, max_delta_step=None, max_depth=None,
              max_leaves=None, min_child_weight=None, missing=nan,
              monotone_constraints=None, multi_strategy=None, n_estimators=None,
              n_jobs=None, num_parallel_tree=None, ...)
```

```
y_pred = xg_model.predict(X_test)
```

```
from sklearn.metrics import *
acc_score = accuracy_score(y_test , y_pred)
auc_score = roc_auc_score(y_test, y_pred)
print("Accuracy score of XGBClassifier model is :", acc_score*100)
print("AUC Score of XGBClassifier model is :", auc_score*100)
```

```
Accuracy score of XGBClassifier model is : 89.98333333333333
AUC Score of XGBClassifier model is : 50.0
```

```
# probabilities (positive class)
y_proba = xg_model.predict_proba(X_test)[:,1]

from sklearn.metrics import roc_auc_score
auc_score = roc_auc_score(y_test, y_proba)
print("Correct AUC (using probabilities):", auc_score*100)
```

```
Correct AUC (using probabilities): 50.60528102904919
```

```
import numpy as np
k = int(0.05 * len(y_test))   # top 5% of test set
top_idx = np.argsort(y_proba)[-k:]   # indices of top-k highest probs
precision_at_k = y_test.iloc[top_idx].sum() / len(top_idx)
print("Precision@Top5%:", precision_at_k)

```

```
Precision@Top5%: 0.11055555555555556
```

```
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, eval_set=[(X_test,y_test)], verbose=False)
```

```
/usr/local/lib/python3.12/dist-packages/xgboost/training.py:199: UserWarning: [15:51:16] WARNING: /workspace/src/learner.cc:
Parameters: { "use_label_encoder" } are not used.

  bst.update(dtrain, iteration=i, fobj=obj)
```

```
XGBClassifier(base_score=None, booster=None, callbacks=None,
              colsample_bylevel=None, colsample_bynode=None,
              colsample_bytree=None, device=None, early_stopping_rounds=None,
              enable_categorical=False, eval_metric='logloss',
              feature_types=None, feature_weights=None, gamma=None,
              grow_policy=None, importance_type=None,
              interaction_constraints=None, learning_rate=0.08, max_bin=None,
              max_cat_threshold=None, max_cat_to_onehot=None,
              max_delta_step=None, max_depth=6, max_leaves=None,
              min_child_weight=None, missing=nan, monotone_constraints=None,
              multi_strategy=None, n_estimators=200, n_jobs=None,
```

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


```

```
from sklearn.metrics import roc_auc_score, precision_recall_curve
auc = roc_auc_score(y_test, y_proba)
print("Auc_score",auc*100)
# precision@k: take top K% (or top K trips)
k = int(0.05 * len(y_test))   # top 5% as example
top_idx = np.argsort(y_proba)[-k:]
precision_at_k = y_test.iloc[top_idx].sum() / len(top_idx)
print("Precision@Top5%:", precision_at_k*100)

```

```
Auc score 50.446081388845485
```

```
df_preds = pd.DataFrame({
    'TRIP_ID': df_trip.iloc[X_test.index]['TRIP_ID'].values,
    'DRIVER_ID': df_trip.iloc[X_test.index]['DRIVER_ID'].values,
    'ACTUAL': y_test.values,
    'PRED_PROB': y_proba

})
```

```
from snowflake.connector.pandas_tools import write_pandas

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

```

```
True 36000
```

```
df_preds
```

	TRIP_ID	DRIVER_ID	ACTUAL	PRED_PROB	SCORE_AT
0	94001	101	0	0.105088	2025-12-03 16:55:20.889377