```
# IMPORTANT: RUN THIS CELL IN ORDER TO IMPORT YOUR KAGGLE DATA SOURCES,
# THEN FEEL FREE TO DELETE THIS CELL.
# NOTE: THIS NOTEBOOK ENVIRONMENT DIFFERS FROM KAGGLE'S PYTHON
# ENVIRONMENT SO THERE MAY BE MISSING LIBRARIES USED BY YOUR
# NOTEBOOK.
import kagglehub
jinquan_cc_sample_data_path = kagglehub.dataset_download('jinquan/cc-sample-data')

print('Data source import complete.')
print(jinquan_cc_sample_data_path)
```

```
Data source import complete.
/root/.cache/kagglehub/datasets/jinquan/cc-sample-data/versions/1
```

```
import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)

data_path = '/root/.cache/kagglehub/datasets/jinquan/cc-sample-data/versions/1/cc_sample_transaction.json'
import os
for dirname, _, filenames in os.walk('/root/.cache/kagglehub/datasets/jinquan/cc-sample-data/versions/'):
    for filename in filenames:
        print(os.path.join(dirname, filename))
```

```
/root/.cache/kagglehub/datasets/jinquan/cc-sample-data/versions/1/cc_sample_transaction.json
```

```
!pip install pyspark
```

```
Requirement already satisfied: pyspark in /usr/local/lib/python3.11/dist-packages (3.5.4)
Requirement already satisfied: py4j==0.10.9.7 in /usr/local/lib/python3.11/dist-packages (from pyspark) (0.10.9.7)
```

```
!pip install ydata_profiling
```

```
Requirement already satisfied: ydata_profiling in /usr/local/lib/python3.11/dist-packages (4.12.2)
Requirement already satisfied: scipy<1.16,>=1.4.1 in /usr/local/lib/python3.11/dist-packages (from ydata_profiling) (1.1
Requirement already satisfied: pandas!=1.4.0,<3,>1.1 in /usr/local/lib/python3.11/dist-packages (from ydata_profiling) (
Requirement already satisfied: matplotlib>=3.5 in /usr/local/lib/python3.11/dist-packages (from ydata_profiling) (3.10.0
Requirement already satisfied: pydantic>=2 in /usr/local/lib/python3.11/dist-packages (from ydata_profiling) (2.10.6)
Requirement already satisfied: PyYAML<6.1,>=5.0.0 in /usr/local/lib/python3.11/dist-packages (from ydata_profiling) (6.0
Requirement already satisfied: jinja2<3.2,>=2.11.1 in /usr/local/lib/python3.11/dist-packages (from ydata_profiling) (3.
Requirement already satisfied: visions<0.8.0,>=0.7.5 in /usr/local/lib/python3.11/dist-packages (from visions[type_image
Requirement already satisfied: numpy<2.2,>=1.16.0 in /usr/local/lib/python3.11/dist-packages (from ydata_profiling) (1.2
Requirement already satisfied: htmlmin==0.1.12 in /usr/local/lib/python3.11/dist-packages (from ydata_profiling) (0.1.12
Requirement already satisfied: phik<0.13,>=0.11.1 in /usr/local/lib/python3.11/dist-packages (from ydata_profiling) (0.1
Requirement already satisfied: requests<3,>=2.24.0 in /usr/local/lib/python3.11/dist-packages (from ydata_profiling) (2.
Requirement already satisfied: tqdm<5,>=4.48.2 in /usr/local/lib/python3.11/dist-packages (from ydata_profiling) (4.67.1
Requirement already satisfied: seaborn<0.14,>=0.10.1 in /usr/local/lib/python3.11/dist-packages (from ydata_profiling) (
Requirement already satisfied: multimethod<2,>=1.4 in /usr/local/lib/python3.11/dist-packages (from ydata_profiling) (1.
Requirement already satisfied: statsmodels<1,>=0.13.2 in /usr/local/lib/python3.11/dist-packages (from ydata_profiling)
Requirement already satisfied: typeguard<5,>=3 in /usr/local/lib/python3.11/dist-packages (from ydata_profiling) (4.4.2)
Requirement already satisfied: imagehash==4.3.1 in /usr/local/lib/python3.11/dist-packages (from ydata_profiling) (4.3.1
Requirement already satisfied: wordcloud>=1.9.3 in /usr/local/lib/python3.11/dist-packages (from ydata_profiling) (1.9.4
Requirement already satisfied: dacite>=1.8 in /usr/local/lib/python3.11/dist-packages (from ydata_profiling) (1.9.2)
Requirement already satisfied: PyWavelets in /usr/local/lib/python3.11/dist-packages (from imagehash==4.3.1->ydata_profi
Requirement already satisfied: pillow in /usr/local/lib/python3.11/dist-packages (from imagehash==4.3.1->ydata_profiling
Requirement already satisfied: MarkupSafe>=2.0 in /usr/local/lib/python3.11/dist-packages (from jinja2<3.2,>=2.11.1->yda
Requirement already satisfied: contourpy>=1.0.1 in /usr/local/lib/python3.11/dist-packages (from matplotlib>=3.5->ydata_
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.11/dist-packages (from matplotlib>=3.5->ydata_prof
Requirement already satisfied: fonttools>=4.22.0 in /usr/local/lib/python3.11/dist-packages (from matplotlib>=3.5->ydata
Requirement already satisfied: kiwisolver>=1.3.1 in /usr/local/lib/python3.11/dist-packages (from matplotlib>=3.5->ydata
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.11/dist-packages (from matplotlib>=3.5->ydata_p
Requirement already satisfied: pyparsing>=2.3.1 in /usr/local/lib/python3.11/dist-packages (from matplotlib>=3.5->ydata_
Requirement already satisfied: python-dateutil>=2.7 in /usr/local/lib/python3.11/dist-packages (from matplotlib>=3.5->yd
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.11/dist-packages (from pandas!=1.4.0,<3,>1.1->ydat
Requirement already satisfied: tzdata>=2022.7 in /usr/local/lib/python3.11/dist-packages (from pandas!=1.4.0,<3,>1.1->yd
Requirement already satisfied: joblib>=0.14.1 in /usr/local/lib/python3.11/dist-packages (from phik<0.13,>=0.11.1->ydata
Requirement already satisfied: annotated-types>=0.6.0 in /usr/local/lib/python3.11/dist-packages (from pydantic>=2->ydat
Requirement already satisfied: pydantic-core==2.27.2 in /usr/local/lib/python3.11/dist-packages (from pydantic>=2->ydata
Requirement already satisfied: typing-extensions>=4.12.2 in /usr/local/lib/python3.11/dist-packages (from pydantic>=2->y
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.11/dist-packages (from requests<3,>=2.
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.11/dist-packages (from requests<3,>=2.24.0->ydata_
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.11/dist-packages (from requests<3,>=2.24.0->
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.11/dist-packages (from requests<3,>=2.24.0->
Requirement already satisfied: patsy>=0.5.6 in /usr/local/lib/python3.11/dist-packages (from statsmodels<1,>=0.13.2->yda
Requirement already satisfied: attrs>=19.3.0 in /usr/local/lib/python3.11/dist-packages (from visions<0.8.0,>=0.7.5->vis
Requirement already satisfied: networkx>=2.4 in /usr/local/lib/python3.11/dist-packages (from visions<0.8.0,>=0.7.5->vis
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.11/dist-packages (from python-dateutil>=2.7->matplotli
```

Basics Importing

```
from pyspark.sql import SparkSession
from pyspark.sql.functions import *
from pyspark.sql.types import *


spark = SparkSession.builder \
    .appName("CreditCardTransactions") \
    .getOrCreate()

df = spark.read.json(data_path)
```

Basics Profiling

```
from ydata_profiling import ProfileReport
print("===== DataFrame Schema =====")
df.printSchema()

print("\n===== Sample Data =====")
df.show(5, truncate=False,vertical=True)

print("\n===== Summary Statistics =====")
df.describe().show(vertical=True)

df_sample = df.limit(10000).toPandas()

# print("\n===== Generating Auto-Profiling Report =====")
# profile = ProfileReport(df_sample, title="Credit Card Transactions Profiling Report", explorative=True)
# profile.to_file("/kaggle/working/data_profiling_report.html")

# print("Auto-profiling report saved as 'data_profiling_report.html'.")
```

```
===== Sample Data =====
-RECORD 0-----------------------------------------------------------------------------------------------------------
 Unnamed: 0             | 0
 amt                    | 4.97
 category               | misc_net
 cc_bic                 | CITIUS33CHI
 cc_num                 | 2703186189652095
 is_fraud               | 0
 merch_eff_time         | 1325376018798532
 merch_last_update_time | 1325376018666
 merch_lat              | 36.011293
 merch_long             | -82.048315
 merch_zipcode          | 28705
 merchant               | fraud_Rippin, Kub and Mann
 personal_detail        | {"person_name":"Jennifer,Banks,eeeee","gender":"F","address":"{\"street\":\"561 Perry Cove\",\
 trans_date_trans_time  | 2019-01-01 00:00:18
 trans_num              | 0b242abb623afc578575680df30655b9
-RECORD 1-----------------------------------------------------------------------------------------------------------
 Unnamed: 0             | 1
 amt                    | 107.23
 category               | grocery_pos
 cc_bic                 | ADMDUS41
 cc_num                 | 630423337322
 is_fraud               | 0
 merch_eff_time         | 1325376044867960
 merch_last_update_time | 132537604479
 merch_lat              | 49.159046999999994
 merch_long             | -118.186462
 merch_zipcode          | NULL
 merchant               | fraud_Heller, Gutmann and Zieme
 personal_detail        | {"person_name":"Stephanie,Gill,eeeee","gender":"F","address":"{\"street\":\"43039 Riley Greens
 trans_date_trans_time  | 2019-01-01 00:00:44
 trans_num              | 1f76529f8574734946361c461b024d99
-RECORD 2-----------------------------------------------------------------------------------------------------------
```

```
    cc_num                 | 3534093764340240
    is_fraud               | 0
    merch_eff_time         | 1325376076794698
    merch_last_update_time | 1325376076365
```

Column Parsing and Formatting

Since `personal_detail` is a json string, I have to parse it using `StructType` Finally, I flatten all parsed fields into a huge dataframe object with the destructured columns.

```
personal_schema = StructType([
    StructField("person_name", StringType()),
    StructField("gender", StringType()),
    StructField("address", StringType()),
    StructField("lat", StringType()),
    StructField("long", StringType()),
    StructField("city_pop", StringType()),
    StructField("job", StringType()),
    StructField("dob", StringType())
])

address_schema = StructType([
    StructField("street", StringType()),
    StructField("city", StringType()),
    StructField("state", StringType()),
    StructField("zip", StringType())
])

df = df.withColumn("personal_detail", from_json(col("personal_detail"), personal_schema)) \
       .withColumn("address", from_json(col("personal_detail.address"), address_schema))

df = df.select(
    col("Unnamed: 0").alias("id"),
    col("trans_date_trans_time"),
    col("cc_num"),
    col("merchant"),
    col("category"),
    col("amt"),
    col("personal_detail.person_name").alias("person_name"),
    col("personal_detail.gender").alias("gender"),
    col("address.street").alias("street"),
    col("address.city").alias("city"),
    col("address.state").alias("state"),
    col("address.zip").alias("zip"),
    col("personal_detail.lat").alias("lat"),
    col("personal_detail.long").alias("long"),
    col("personal_detail.city_pop").alias("city_pop"),
    col("personal_detail.job").alias("job"),
    col("personal_detail.dob").alias("dob"),
    col("trans_num"),
    col("merch_lat"),
    col("merch_long"),
    col("is_fraud"),
    col("merch_zipcode"),
    col("merch_last_update_time"),
    col("merch_eff_time"),
    col("cc_bic")
)
```

For handling PII Data

For PII Data, there are several ways to handle them.

1. I can either mask them with mask string '**** 1243'
2. Hash the field.

I choose to hash the column because it's reversible and I can get decode the original data.

```
# Handle PII Data (Hashing Sensitive Columns)
pii_columns = ["cc_num", "street", "dob", "job"]

for column in pii_columns:
    df = df.withColumn(column, sha2(col(column), 256))
```

```
df = df.withColumn("person_name_cleaned", regexp_replace(col("person_name"), "[^a-zA-Z]", ",")) \
```

```
       .withColumn("tmp_split", split(col("person_name_cleaned"), ",")) \
       .withColumn("first", trim(col("tmp_split")[0])) \
       .withColumn("last", trim(col("tmp_split")[1])) \
       .drop("tmp_split", "person_name", "person_name_cleaned")

df = df.withColumn("first", sha2(col("first"), 256)) \
       .withColumn("last", sha2(col("last"), 256))
```

Handling Timestamps

I cast everything into timestamp then convert to TC+8

```
# Convert Timestamps to UTC+8
df = df.withColumn("trans_date_trans_time",
                   from_utc_timestamp(to_timestamp(col("trans_date_trans_time"), "yyyy-MM-dd HH:mm:ss"), "UTC+8"))

df = df.withColumn("merch_last_update_time",
                   from_utc_timestamp(from_unixtime(col("merch_last_update_time").cast("double")/1000), "UTC+8")) \
       .withColumn("merch_eff_time",
                   from_utc_timestamp(from_unixtime(col("merch_eff_time").cast("double")/1e6), "UTC+8"))
```

Data Quality Cleaning

Here, I choose to drop rows where is_fraud is null. This is simply for data quality checks, and for further visualization for meaningful insights. Otherwise, we can choose to not drop the rows too.

```
# Data Quality
df = df.withColumn("amt", col("amt").cast("double")) \
       .withColumn("city_pop", col("city_pop").cast("integer"))

df = df.withColumn("is_fraud", when(col("is_fraud").isin(["0", "1"]), col("is_fraud")).otherwise(None))

df = df.na.drop(subset=["cc_num", "amt", "is_fraud"])
```

For the visualization

Since some fields are hashed for PII purpose, I choose to focus on the fraud distribution statistics. From the visualization below we can see most of the transaction are non-fraudulent. Also, transaction amount of fraudulent transaction greatly exceeds non-fraudulent ones.

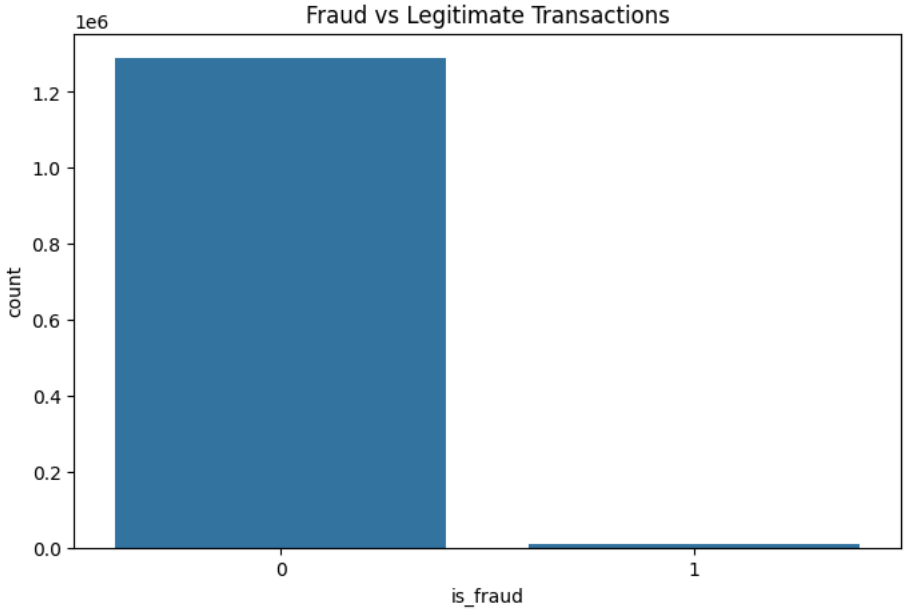
```
fraud_counts = df.groupBy("is_fraud").count().toPandas()
```

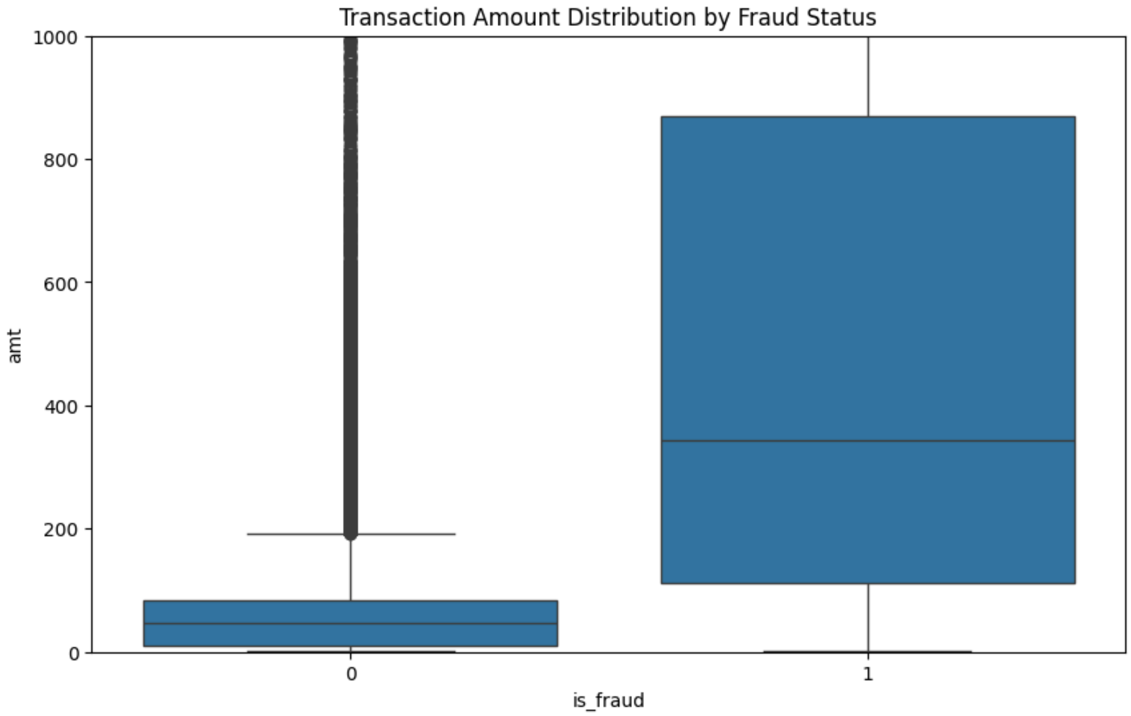
```
import matplotlib.pyplot as plt
import seaborn as sns

plt.figure(figsize=(8,5))
sns.barplot(x='is_fraud', y='count', data=fraud_counts)
plt.title("Fraud vs Legitimate Transactions")
plt.show()

amt_stats = df.groupBy("is_fraud").agg(mean("amt"), stddev("amt")).toPandas()

plt.figure(figsize=(10,6))
sns.boxplot(x='is_fraud', y='amt', data=df.sample(0.1).toPandas())
plt.ylim(0, 1000)
plt.title("Transaction Amount Distribution by Fraud Status")
plt.show()
```





More data quality and integrity checks

We can specify more constraints in the pipeline, and log the errors.

Due to time constraints, this is not fully integrated in the pipeline.

```
from pyspark.sql.functions import col, when, lit
from functools import reduce

validation_rules = {
    "cc_num": col("cc_num").rlike(r"^\d{16}$"),  # Must be 16 digits
    "amt": col("amt") > 0,  # Must be positive
    "is_fraud": col("is_fraud").isin(["0", "1"]),  # Must be 0 or 1
    "dob": col("dob").rlike(r"^\d{4}-\d{2}-\d{2}$"),  # Must be a valid date format
    "city_pop": col("city_pop") >= 0,  # Must be non-negative
    "lat": (col("lat").cast("double").between(-90, 90)),  # Must be valid latitude
    "long": (col("long").cast("double").between(-180, 180))  # Must be valid longitude
}


validation_condition = reduce(
    lambda a, b: a & b,
    [when(rule, lit(True)).otherwise(lit(False)) for rule in validation_rules.values()]
)
```

```
error_log = df.select("*").where(~validation_condition)


error_count = error_log.count()
print(f"Number of rows with errors: {error_count}")

error_log.show(5)
```

```
Number of rows with errors: 1296675
+---+--------------------+--------------------+--------------------+-------------+------+------+--------------------+--
| id|trans_date_trans_time|              cc_num|            merchant|     category|   amt|gender|              street|
+---+--------------------+--------------------+--------------------+-------------+------+------+--------------------+--
|  0|  2019-01-01 08:00:18|80923ef01336409c8...|fraud_Rippin, Kub...|     misc_net|  4.97|     F|41d1806600fe3193f...|Mo
|  1|  2019-01-01 08:00:44|f80a8e60a9f15ecf1...|fraud_Heller, Gut...|  grocery_pos|107.23|     F|aff1802dbeae07dab...|
|  2|  2019-01-01 08:00:51|756a303c0348d0ebb...|fraud_Lind-Buckridge|entertainment|220.11|     M|674a2376d747e43a0...|
|  3|  2019-01-01 08:01:16|374dcb008121abf2b...|fraud_Kutch, Herm...|gas_transport|  45.0|     M|bfac23de044b241ba...|
|  4|  2019-01-01 08:03:06|7f921c03617da9920...| fraud_Keeling-Crist|     misc_pos| 41.96|     M|eb6a57db860b9aec4...|
+---+--------------------+--------------------+--------------------+-------------+------+------+--------------------+--
only showing top 5 rows
```

Start coding or generate with AI.