```
# from google.colab import drive
# drive.mount('/content/drive')
```

```
# import libraries
from pyspark.sql import SparkSession
from pyspark.sql.functions import col, sum, unix_timestamp, to_date, lit
from pyspark.sql.types import DoubleType
```

```
# start a spark session
spark = SparkSession.builder.appName('Bank_fraud').getOrCreate()
```

```
fraud_train= spark.read.csv('/content/drive/Othercomputers/My Laptop - Personal/Files/Data_Files/Portfolio Projects/Bank_Fraud_Detection/fraudTrain.csv',
                header=True, inferSchema=True)
fraud_test = spark.read.csv('/content/drive/Othercomputers/My Laptop - Personal/Files/Data_Files/Portfolio Projects/Bank_Fraud_Detection/fraudTest.csv',
                header=True, inferSchema=True)
# load train data
fraud_train.show(5)
```

_c0	trans_date_trans_time	cc_num	merchant	category	amt	first	last	gender	street	city	state	zip	lat	long	city_pop	job	dob	trans_num
0	2019-01-01 00:00:18	2703186189652095	fraud_Rippin, Kub...	misc_net	4.97	Jennifer	Banks	F	561 Perry Cove	Moravian Falls	NC	28654	36.0788	-81.1781	3495	Psychologist, cou...	1988-03-09	0b242abb623afc578...
1	2019-01-01 00:00:44	630423337322	fraud_Heller, Gut...	grocery_pos	107.23	Stephanie	Gill	F	43039 Riley Green...	Orient	WA	99160	48.8878	-118.2105	149	Special education...	1978-06-21	1f76529f857473494...
2	2019-01-01 00:00:51	38859492057661	fraud_Lind-Buckridge	entertainment	220.11	Edward	Sanchez	M	594 White Dale Su...	Malad City	ID	83252	42.1808	-112.262	4154	Nature conservati...	1962-01-19	a1a22d70485983eac...
3	2019-01-01 00:01:16	3534093764340240	fraud_Kutch, Herm...	gas_transport	45.0	Jeremy	White	M	9443 Cynthia Cour...	Boulder	MT	59632	46.2306	-112.1138	1939	Patent attorney	1967-01-12	6b849c168bdad6f86...
4	2019-01-01 00:03:06	375534208663984	fraud_Keeling-Crist	misc_pos	41.96	Tyler	Garcia	M	408 Bradley Rest	Doe Hill	VA	24433	38.4207	-79.4629	99	Dance movement ps...	1986-03-28	a41d7549acf907893...

only showing top 5 rows

```
# load test data
fraud_test.show(5)
```

_c0	trans_date_trans_time	cc_num	merchant	category	amt	first	last	gender	street	city	state	zip	lat	long	city_pop	job	dob	trans_n
0	2020-06-21 12:14:25	2291163933867244	fraud_Kirlin and ...	personal_care	2.86	Jeff	Elliott	M	351 Darlene Green	Columbia	SC	29209	33.9659	-80.9355	333497	Mechanical engineer	1968-03-19	2da90c7d74bd46a0c.
1	2020-06-21 12:14:33	3573030041201292	fraud_Sporer-Keebler	personal_care	29.84	Joanne	Williams	F	3638 Marsh Union	Altonah	UT	84002	40.3207	-110.436	302	Sales professiona...	1990-01-17	324cc204407e99f51.
2	2020-06-21 12:14:53	3598215285024754	fraud_Swaniawski,...	health_fitness	41.28	Ashley	Lopez	F	9333 Valentine Point	Bellmore	NY	11710	40.6729	-73.5365	34496	Librarian, public	1970-10-21	c81755dbbbea9d5c7.
3	2020-06-21 12:15:15	3591919803438423	fraud_Haley Group	misc_pos	60.05	Brian	Williams	M	32941 Krystal Mil...	Titusville	FL	32780	28.5697	-80.8191	54767	Set designer	1987-07-25	2159175b9efe66dc3.
4	2020-06-21 12:15:17	3526826139003047	fraud_Johnston-Ca...	travel	3.19	Nathan	Massey	M	5783 Evan Roads A...	Falmouth	MI	49632	44.2529	-85.01700000000001	1126	Furniture designer	1955-07-06	57ff021bd3f328f87.

only showing top 5 rows

Data Understanding

```
# count the number of rows
print(f"There are {fraud_train.count()} rows in the fraud_train dataset")
print(f"There are {fraud_test.count()} rows in the fraud_test dataset")
```

```
There are 1296675 rows in the fraud_train dataset
There are 555719 rows in the fraud_test dataset
```

```
# check duplicates - fraud_train
fraud_train.exceptAll(fraud_train.dropDuplicates()).show()
```

_c0	trans_date_trans_time	cc_num	merchant	category	amt	first	last	gender	street	city	state	zip	lat	long	city_pop	job	dob	trans_num	unix_time	merch_lat	merch_long	is_fraud

```
# check duplicates - fraud_test
fraud_test.exceptAll(fraud_test.dropDuplicates()).show()
```

_c0	trans_date_trans_time	cc_num	merchant	category	amt	first	last	gender	street	city	state	zip	lat	long	city_pop	job	dob	trans_num	unix_time	merch_lat	merch_long	is_fraud

```
# check schema
print('Schema of fraud_train:')
fraud_train.printSchema()
```

```
Schema of fraud_train:
root
 |-- _c0: integer (nullable = true)
 |-- trans_date_trans_time: timestamp (nullable = true)
 |-- cc_num: long (nullable = true)
 |-- merchant: string (nullable = true)
 |-- category: string (nullable = true)
 |-- amt: double (nullable = true)
 |-- first: string (nullable = true)
 |-- last: string (nullable = true)
 |-- gender: string (nullable = true)
 |-- street: string (nullable = true)
 |-- city: string (nullable = true)
 |-- state: string (nullable = true)
 |-- zip: integer (nullable = true)
 |-- lat: double (nullable = true)
 |-- long: double (nullable = true)
 |-- city_pop: integer (nullable = true)
 |-- job: string (nullable = true)
 |-- dob: date (nullable = true)
 |-- trans_num: string (nullable = true)
 |-- unix_time: integer (nullable = true)
 |-- merch_lat: double (nullable = true)
 |-- merch_long: double (nullable = true)
 |-- is_fraud: integer (nullable = true)
```

```
# check schema
print('Schema of fraud_test:')
fraud_test.printSchema()
```

```
Schema of fraud_test:
root
 |-- _c0: integer (nullable = true)
 |-- trans_date_trans_time: timestamp (nullable = true)
 |-- cc_num: long (nullable = true)
 |-- merchant: string (nullable = true)
 |-- category: string (nullable = true)
 |-- amt: double (nullable = true)
 |-- first: string (nullable = true)
 |-- last: string (nullable = true)
 |-- gender: string (nullable = true)
 |-- street: string (nullable = true)
 |-- city: string (nullable = true)
 |-- state: string (nullable = true)
 |-- zip: integer (nullable = true)
 |-- lat: double (nullable = true)
 |-- long: double (nullable = true)
 |-- city_pop: integer (nullable = true)
 |-- job: string (nullable = true)
 |-- dob: date (nullable = true)
 |-- trans_num: string (nullable = true)
 |-- unix_time: integer (nullable = true)
 |-- merch_lat: double (nullable = true)
 |-- merch_long: double (nullable = true)
 |-- is_fraud: integer (nullable = true)
```

```
# Check for null values in each column
# cast nulls to int: 1 for null, 0 for non-null. Summarize the count of these values.
print('Null values in fraud_train:')
fraud_train.select([sum(col(c).isNull().cast('int')).alias(c) for c in fraud_train.columns]).show()
```

Null values in fraud_train:

_c0	trans_date_trans_time	cc_num	merchant	category	amt	first	last	gender	street	city	state	zip	lat	long	city_pop	job	dob	trans_num	unix_time	merch_lat	merch_long	is_fraud
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

```
print('Null values in fraud_test:')
fraud_test.select([sum(col(c).isNull().cast("int")).alias(c) for c in fraud_test.columns]).show()
```

Null values in fraud_test:

_c0	trans_date_trans_time	cc_num	merchant	category	amt	first	last	gender	street	city	state	zip	lat	long	city_pop	job	dob	trans_num	unix_time	merch_lat	merch_long	is_fraud
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Resources

You are not subscribed. Learn more

You currently have zero compute units available. Resources offered free of charge are not guaranteed. Purchase more units here.

Manage sessions

Not connected to runtime.

Change runtime type

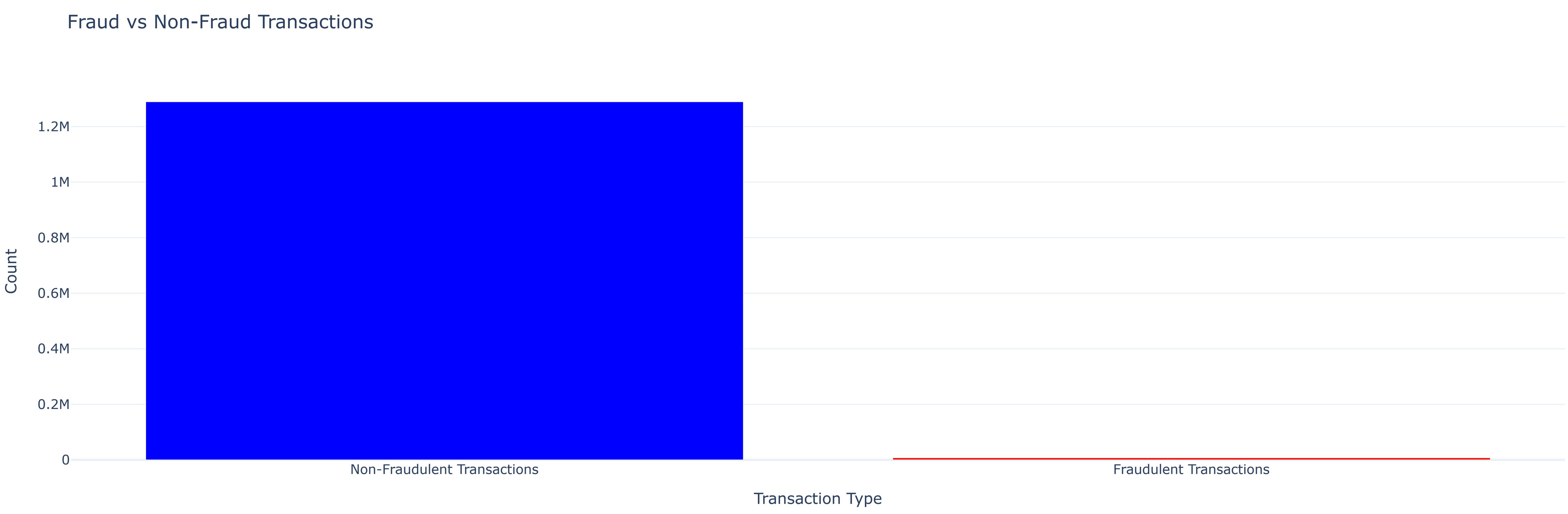
```
# check number of unique customers (cc_num) and unique transactions (trans_num) from fraud_train
# Count unique customers (cc_num)
unique_customers = fraud_train.select("cc_num").distinct().count()
print(f"Number of unique customers: {unique_customers}")

# Count unique transactions (trans_num)
unique_transactions = fraud_train.select("trans_num").distinct().count()
print(f"Number of unique transactions: {unique_transactions}")
```

```
Number of unique customers: 983
Number of unique transactions: 1296675
```

```
# filter rows where is_fraud is 0 - Nn-fraudulent transaction
non_fraud = fraud_train.filter(fraud_train["is_fraud"] == 0)
print(f"Number of non-fraudulent transactions: {non_fraud.count()}")

# filter rows where is_fraud is 1 - Fraudulent transaction
fraud = fraud_train.filter(fraud_train["is_fraud"] == 1)
print(f"Number of fraudulent transactions: {fraud.count()}")
```

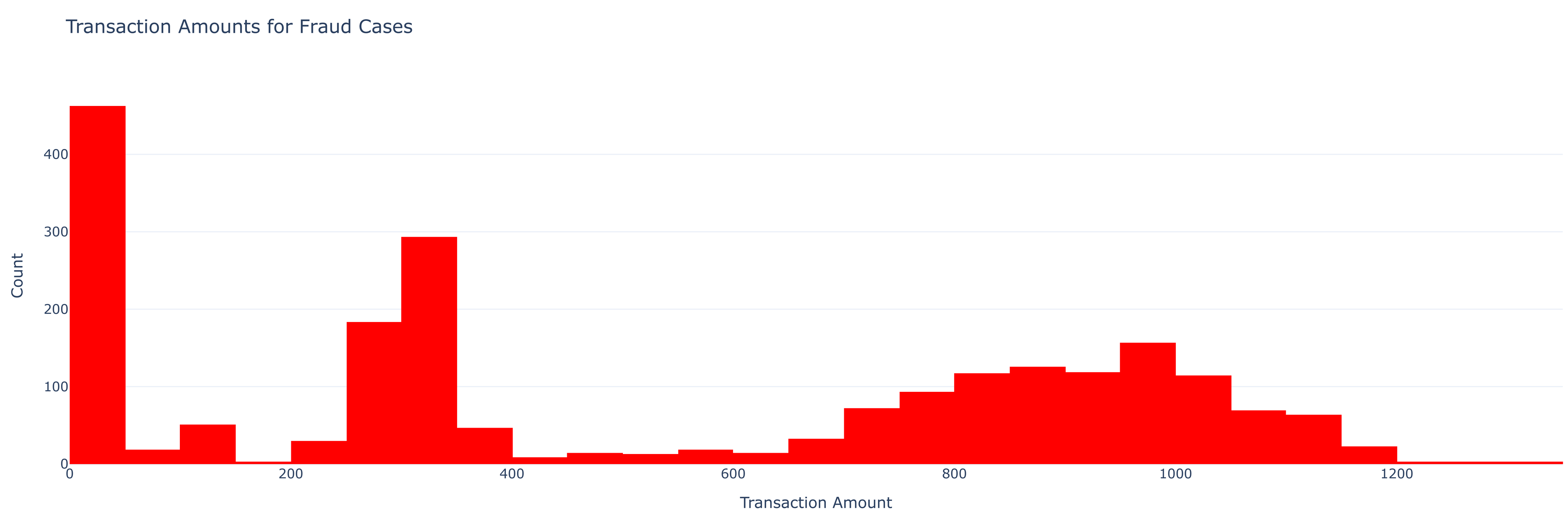
```
Number of non-fraudulent transactions: 1289169
Number of fraudulent transactions: 7506
```

```
import plotly.graph_objects as go

# Count transactions
non_fraud_count = non_fraud.count()
fraud_count = fraud.count()

# Create the bar graph
fig = go.Figure()

# Add bars for non-fraud and fraud counts
fig.add_trace(go.Bar(
    x=['Non-Fraudulent Transactions', 'Fraudulent Transactions'],
    y=[non_fraud_count, fraud_count],
    marker_color=['blue', 'red']  # Different colors for clarity
))

# Customize layout
fig.update_layout(
    title='Fraud vs Non-Fraud Transactions',
    xaxis_title='Transaction Type',
    yaxis_title='Count',
    template='plotly_white',
)

# Show the plot
fig.show()
```



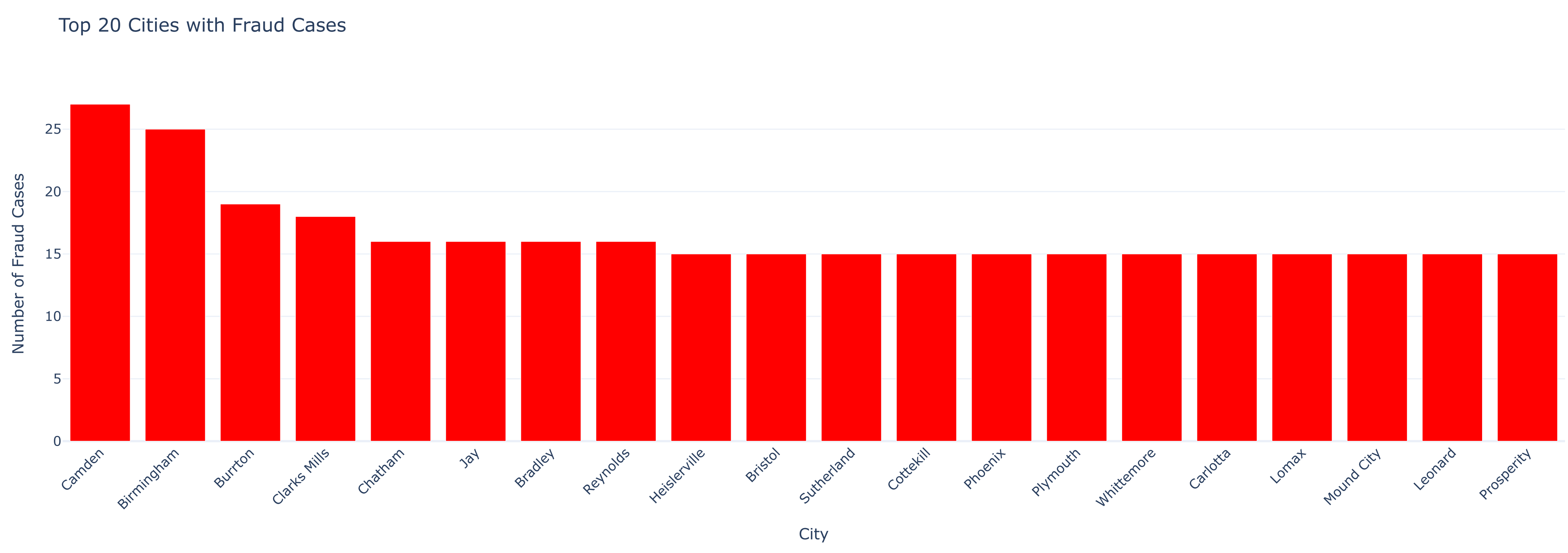
- Fraud cases are 7506 in number, making that 0.5% iof the total transaction. The data is extremely imbalnced

```
#  Transaction Amount Distribution
# Filter for fraud cases only
fraud_cases = fraud_test.filter(col("is_fraud") == 1)

# Convert the fraud cases to a Pandas DataFrame
fraud_cases_pd = fraud_cases.select("amt").toPandas()

# Create the plot for fraud cases' transaction amounts
fig_fraud_amt = go.Figure(
    go.Histogram(
        x=fraud_cases_pd["amt"],
        nbinsx=30,  # You can adjust the number of bins
        marker_color="red",
        name="Fraud Transaction Amount"
    )
)

# Update layout for the plot
fig_fraud_amt.update_layout(
    title_text="Transaction Amounts for Fraud Cases",
    xaxis_title="Transaction Amount",
    yaxis_title="Count",
    template="plotly_white"
)

# Show the plot
fig_fraud_amt.show()
```

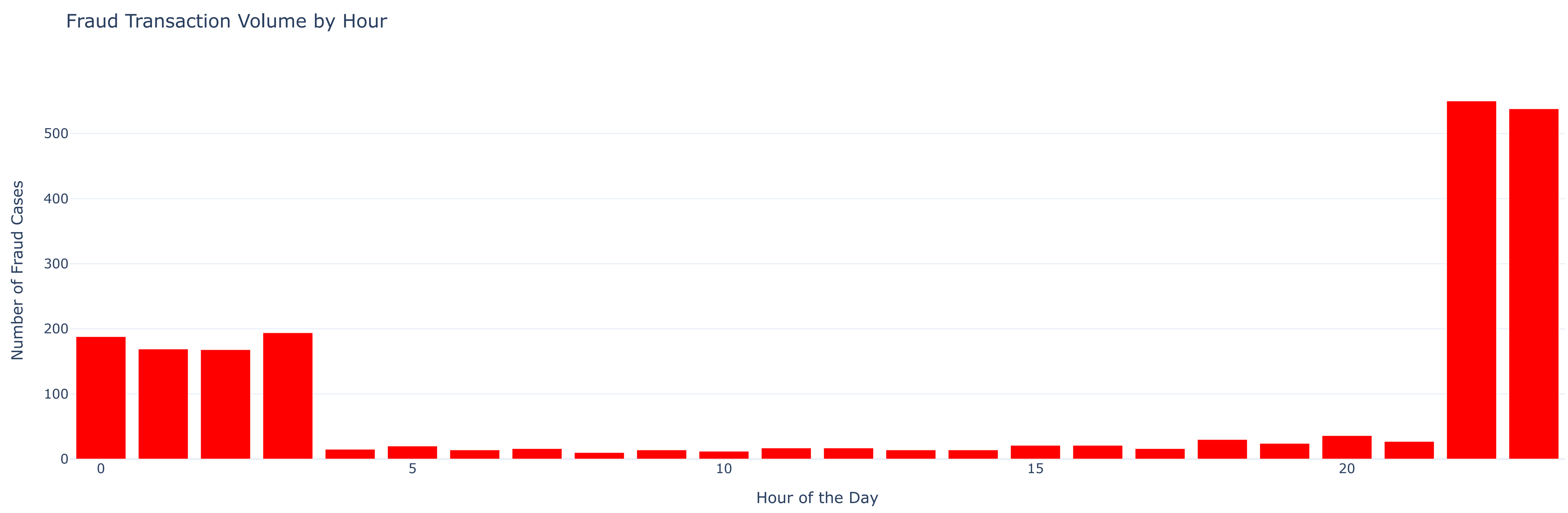



```
# Top 20 Cities with Fraud Cases
# Group by city and count the number of fraud cases in each city
fraud_by_city = fraud_cases.groupBy("city").count().toPandas()

# Sort by the count in descending order and select the top 20 cities
top_20_cities = fraud_by_city.sort_values(by="count", ascending=False).head(20)

# Create the bar plot for fraud cases by city (top 20 cities)
fig_fraud_city_top_20 = go.Figure(
    go.Bar(
        x=top_20_cities["city"],
        y=top_20_cities["count"],
        marker_color="red",
        name="Fraud Cases by City"
    )
)

# Update layout for the plot
fig_fraud_city_top_20.update_layout(
```

```
    title_text="Top 20 Cities with Fraud Cases",
    xaxis_title="City",
    yaxis_title="Number of Fraud Cases",
    template="plotly_white",
    xaxis_tickangle=-45  # Rotates x-axis labels if necessary
)

# Show the plot
fig_fraud_city_top_20.show()
```




```
from pyspark.sql.functions import hour
# Fraud Transaction Volume by Hour
# Extract the hour from the transaction date and time
fraud_cases_with_hour = fraud_cases.withColumn("hour", hour(fraud_cases["trans_date_trans_time"]))

# Group by hour and count the number of fraud cases for each hour
fraud_by_hour = fraud_cases_with_hour.groupBy("hour").count().toPandas()

# Create the bar plot for fraud transaction volume by hour
fig_fraud_time_bar = go.Figure(
    go.Bar(
        x=fraud_by_hour["hour"],  # Time (hour)
        y=fraud_by_hour["count"],  # Number of fraud cases
        marker=dict(color="red"),
        name="Fraud Cases by Hour"
    )
)

# Update layout for the plot
fig_fraud_time_bar.update_layout(
    title_text="Fraud Transaction Volume by Hour",
    xaxis_title="Hour of the Day",
    yaxis_title="Number of Fraud Cases",
    template="plotly_white"
)

# Show the plot
fig_fraud_time_bar.show()
```



Data Cleaning

- Rename the columns for better understanding
    - `first` to `first_name`
    - `last` to `last_name`
    - `city_pop` to `city_population`
    - `dob` to `date_of_birth`

```
# rename the columns for both train and test datasets
# train dataset
fraud_train = fraud_train.withColumnRenamed('first', 'first_name') \
                        .withColumnRenamed('last', 'last_name') \
                        .withColumnRenamed('city_pop', 'city_population') \
                        .withColumnRenamed('dob', 'date_of_birth')

# test dataset
fraud_test = fraud_test.withColumnRenamed('first', 'first_name') \
                      .withColumnRenamed('last', 'last_name') \
                      .withColumnRenamed('city_pop', 'city_population') \
                      .withColumnRenamed('dob', 'date_of_birth')

# Verify the columns were dropped
fraud_train.printSchema()
fraud_test.printSchema()
```

```
root
 |-- _c0: integer (nullable = true)
 |-- trans_date_trans_time: timestamp (nullable = true)
 |-- cc_num: long (nullable = true)
 |-- merchant: string (nullable = true)
 |-- category: string (nullable = true)
 |-- amt: double (nullable = true)
 |-- first_name: string (nullable = true)
 |-- last_name: string (nullable = true)
 |-- gender: string (nullable = true)
 |-- street: string (nullable = true)
 |-- city: string (nullable = true)
 |-- state: string (nullable = true)
 |-- zip: integer (nullable = true)
 |-- lat: double (nullable = true)
 |-- long: double (nullable = true)
 |-- city_population: integer (nullable = true)
 |-- job: string (nullable = true)
 |-- date_of_birth: date (nullable = true)
 |-- trans_num: string (nullable = true)
 |-- unix_time: integer (nullable = true)
 |-- merch_lat: double (nullable = true)
 |-- merch_long: double (nullable = true)
 |-- is_fraud: integer (nullable = true)

root
 |-- _c0: integer (nullable = true)
 |-- trans_date_trans_time: timestamp (nullable = true)
 |-- cc_num: long (nullable = true)
 |-- merchant: string (nullable = true)
 |-- category: string (nullable = true)
 |-- amt: double (nullable = true)
 |-- first_name: string (nullable = true)
 |-- last_name: string (nullable = true)
 |-- gender: string (nullable = true)
 |-- street: string (nullable = true)
 |-- city: string (nullable = true)
 |-- state: string (nullable = true)
```

```
 |-- zip: integer (nullable = true)
 |-- lat: double (nullable = true)
 |-- long: double (nullable = true)
 |-- city_population: integer (nullable = true)
 |-- job: string (nullable = true)
 |-- date_of_birth: date (nullable = true)
 |-- trans_num: string (nullable = true)
 |-- unix_time: integer (nullable = true)
 |-- merch_lat: double (nullable = true)
 |-- merch_long: double (nullable = true)
 |-- is_fraud: integer (nullable = true)
```

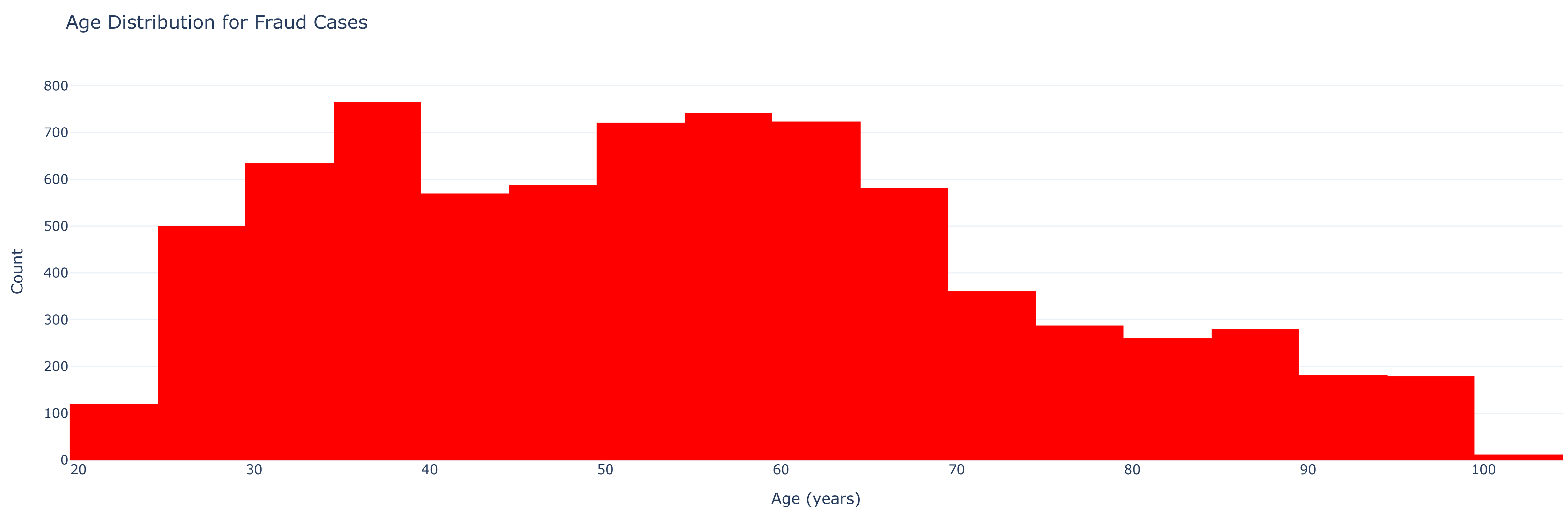
Feature Engineering

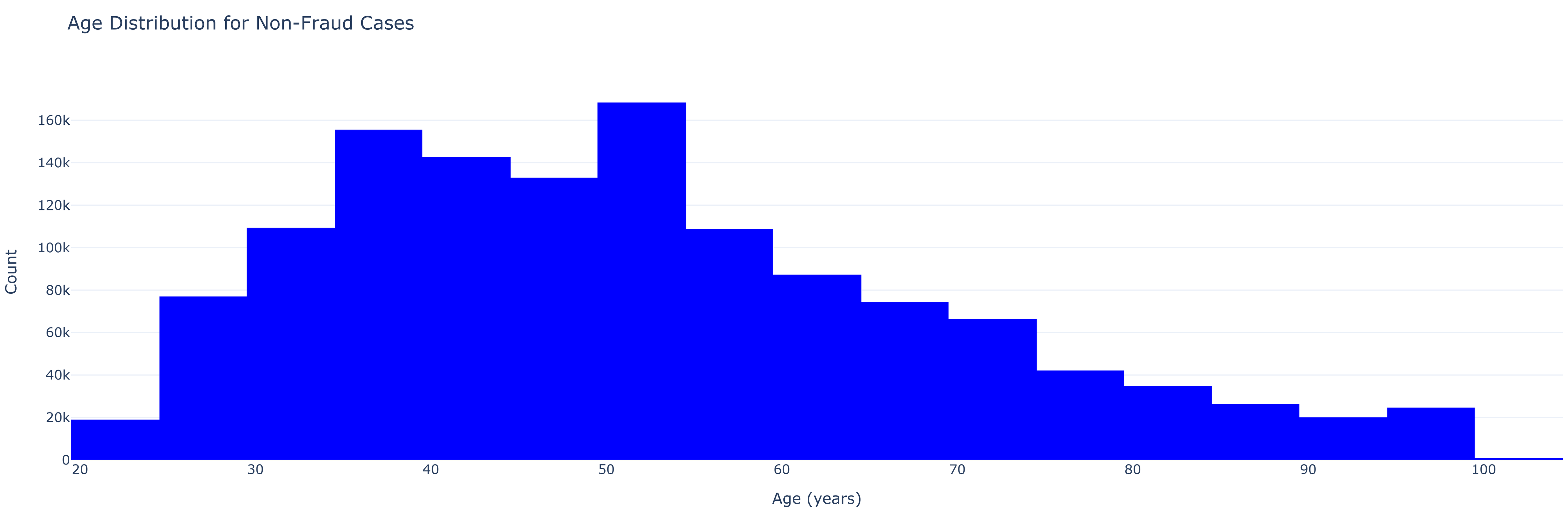
- Add the `age` column derived from date_of_birth
- Create `transaction_hour` and `transaction_day`, `transaction_month`
- Add a distance in kilometres column `distance_cust_to_merch(km)`
- Drop columns that are not needed in the modelling
  - `_c0`
  - `cc_num`
  - `trans_num`
  - `unix_time`
  - `first_name`
  - `last_name`
  - `street`

```python
from pyspark.sql.functions import col, current_date, year, month, dayofweek, hour, sqrt, pow

# Create 'age' column from 'date_of_birth'
fraud_train = fraud_train.withColumn('age', year(current_date()) - year(col('date_of_birth')))
fraud_test = fraud_test.withColumn('age', year(current_date()) - year(col('date_of_birth')))

# Create 'transaction_hour' and 'transaction_day', 'transaction_month'
# train dataset
fraud_train = fraud_train.withColumn('transaction_month', month(col('trans_date_trans_time')))
fraud_train = fraud_train.withColumn('transaction_hour', hour(col('trans_date_trans_time')))
fraud_train = fraud_train.withColumn('transaction_day', dayofweek(col('trans_date_trans_time')))

# test dataset
fraud_test = fraud_test.withColumn('transaction_month', month(col('trans_date_trans_time')))
fraud_test = fraud_test.withColumn('transaction_hour', hour(col('trans_date_trans_time')))
fraud_test = fraud_test.withColumn('transaction_day', dayofweek(col('trans_date_trans_time')))

# Calculate distance between customer and merchant in km. 111 is usied to get the estimate distance
# train dataset
fraud_train = fraud_train.withColumn(
    'distance_cust_to_merch(km)',
    sqrt(pow(col('lat') - col('merch_lat'), 2) + pow(col('long') - col('merch_long'), 2)) * 111)
# test dataset
fraud_test = fraud_test.withColumn(
    'distance_cust_to_merch(km)',
    sqrt(pow(col('lat') - col('merch_lat'), 2) + pow(col('long') - col('merch_long'), 2)) * 111)

# drop unnecessary columns
fraud_train = fraud_train.drop('_c0', 'cc_num', 'trans_num', 'unix_time', 'first_name', 'last_name', 'street')
fraud_test = fraud_test.drop('_c0', 'cc_num', 'trans_num', 'unix_time', 'first_name', 'last_name', 'street')

# Verify updated data
fraud_train.show(5)
fraud_test.show(5)
```

```
+-------------------+--------------------+-------------+------+------+--------------+-----+-----+-------+---------+---------------+--------------------+-------------+-----------------+-----------+--------+---+-----------------+---------
|trans_date_trans_time|            merchant|     category|   amt|gender|          city|state|  zip|    lat|     long|city_population|                 job|date_of_birth|        merch_lat| merch_long|is_fraud|age|transaction_month|transacti
+-------------------+--------------------+-------------+------+------+--------------+-----+-----+-------+---------+---------------+--------------------+-------------+-----------------+-----------+--------+---+-----------------+---------
|  2019-01-01 00:00:18|fraud_Rippin, Kub...|     misc_net|  4.97|     F|Moravian Falls|   NC|28654|36.0788| -81.1781|           3495|Psychologist, cou...|   1988-03-09|        36.011293| -82.048315|       0| 37|                1|
|  2019-01-01 00:00:44|fraud_Heller, Gut...|  grocery_pos|107.23|     F|        Orient|   WA|99160|48.8878|-118.2105|            149|Special education...|   1978-06-21|49.159046999999994|-118.186462|       0| 47|                1|
|  2019-01-01 00:00:51|fraud_Lind-Buckridge|entertainment|220.11|     M|    Malad City|   ID|83252|42.1808| -112.262|           4154|Nature conservati...|   1962-01-19|        43.150704|-112.154481|       0| 63|                1|
|  2019-01-01 00:01:16|fraud_Kutch, Herm...|gas_transport|  45.0|     M|       Boulder|   MT|59632|46.2306|-112.1138|           1939|     Patent attorney|   1967-01-12|        47.034331|-112.561071|       0| 58|                1|
|  2019-01-01 00:03:06| fraud_Keeling-Crist|     misc_pos| 41.96|     M|      Doe Hill|   VA|24433|38.4207| -79.4629|             99|Dance movement ps...|   1986-03-28|        38.674999| -78.632459|       0| 39|                1|
+-------------------+--------------------+-------------+------+------+--------------+-----+-----+-------+---------+---------------+--------------------+-------------+-----------------+-----------+--------+---+-----------------+---------
only showing top 5 rows

+-------------------+--------------------+--------------+-----+------+-----------+-----+-----+-------+------------------+---------------+--------------------+-------------+------------------+-----------+--------+---+-----------------+----
|trans_date_trans_time|            merchant|      category|  amt|gender|       city|state|  zip|    lat|              long|city_population|                 job|date_of_birth|         merch_lat| merch_long|is_fraud|age|transaction_month|tran
+-------------------+--------------------+--------------+-----+------+-----------+-----+-----+-------+------------------+---------------+--------------------+-------------+------------------+-----------+--------+---+-----------------+----
|  2020-06-21 12:14:25|fraud_Kirlin and ...| personal_care| 2.86|     M|   Columbia|   SC|29209|33.9659|          -80.9355|         333497| Mechanical engineer|   1968-03-19|         33.986391| -81.200714|       0| 57|                6|
|  2020-06-21 12:14:33|fraud_Sporer-Keebler| personal_care|29.84|     F|    Altonah|   UT|84002|40.3207|          -110.436|            302|Sales professiona...|   1990-01-17|39.450497999999996|-109.960431|       0| 35|                6|
|  2020-06-21 12:14:53|fraud_Swaniawski,...|health_fitness|41.28|     F|   Bellmore|   NY|11710|40.6729|          -73.5365|          34496|   Librarian, public|   1970-10-21|          40.49581| -74.196111|       0| 55|                6|
|  2020-06-21 12:15:15|   fraud_Haley Group|      misc_pos|60.05|     M| Titusville|   FL|32780|28.5697|          -80.8191|          54767|        Set designer|   1987-07-25|28.812397999999998| -80.883061|       0| 38|                6|
|  2020-06-21 12:15:17|fraud_Johnston-Ca...|        travel| 3.19|     M|   Falmouth|   MI|49632|44.2529|-85.01700000000001|           1126|  Furniture designer|   1955-07-06|         44.959148| -85.884734|       0| 70|                6|
+-------------------+--------------------+--------------+-----+------+-----------+-----+-----+-------+------------------+---------------+--------------------+-------------+------------------+-----------+--------+---+-----------------+----
only showing top 5 rows
```

```python
# view age distribution for fraud cases
import plotly.graph_objects as go

# Filter for fraud and non-fraud cases from fraud_train
non_fraud_cases = fraud_train.filter(col('is_fraud') == 0)
fraud_cases = fraud_train.filter(col('is_fraud') == 1)

# Convert both fraud and non-fraud cases to Pandas DataFrames
non_fraud_cases_pd = non_fraud_cases.select('age').toPandas()
fraud_cases_pd = fraud_cases.select('age').toPandas()

# Create the first plot for fraud cases
fig_fraud = go.Figure(
    go.Histogram(
        x=fraud_cases_pd['age'],
        nbinsx=20,
        marker_color='red',
        name='Fraud Cases'
    )
)

fig_fraud.update_layout(
    title_text='Age Distribution for Fraud Cases',
    xaxis_title='Age (years)',
    yaxis_title='Count',
    template='plotly_white'
)
fig_fraud.show()

# Create the second plot for non-fraud cases
fig_non_fraud = go.Figure(
    go.Histogram(
        x=non_fraud_cases_pd['age'],
        nbinsx=20,
        marker_color='blue',
        name='Non-Fraud Cases'
    )
)

fig_non_fraud.update_layout(
    title_text='Age Distribution for Non-Fraud Cases',
    xaxis_title='Age (years)',
    yaxis_title='Count',
    template='plotly_white'
)
fig_non_fraud.show()
```






```
# check schema
fraud_train.printSchema()
fraud_test.printSchema()
```

```
root
 |-- trans_date_trans_time: timestamp (nullable = true)
 |-- merchant: string (nullable = true)
 |-- category: string (nullable = true)
 |-- amt: double (nullable = true)
 |-- gender: string (nullable = true)
 |-- city: string (nullable = true)
 |-- state: string (nullable = true)
 |-- zip: integer (nullable = true)
 |-- lat: double (nullable = true)
 |-- long: double (nullable = true)
 |-- city_population: integer (nullable = true)
 |-- job: string (nullable = true)
 |-- date_of_birth: date (nullable = true)
 |-- merch_lat: double (nullable = true)
 |-- merch_long: double (nullable = true)
 |-- is_fraud: integer (nullable = true)
 |-- age: integer (nullable = true)
 |-- transaction_month: integer (nullable = true)
 |-- transaction_hour: integer (nullable = true)
 |-- transaction_day: integer (nullable = true)
 |-- distance_cust_to_merch(km): double (nullable = true)

root
 |-- trans_date_trans_time: timestamp (nullable = true)
 |-- merchant: string (nullable = true)
 |-- category: string (nullable = true)
 |-- amt: double (nullable = true)
 |-- gender: string (nullable = true)
 |-- city: string (nullable = true)
 |-- state: string (nullable = true)
 |-- zip: integer (nullable = true)
 |-- lat: double (nullable = true)
 |-- long: double (nullable = true)
 |-- city_population: integer (nullable = true)
 |-- job: string (nullable = true)
 |-- date_of_birth: date (nullable = true)
 |-- merch_lat: double (nullable = true)
 |-- merch_long: double (nullable = true)
 |-- is_fraud: integer (nullable = true)
 |-- age: integer (nullable = true)
 |-- transaction_month: integer (nullable = true)
 |-- transaction_hour: integer (nullable = true)
 |-- transaction_day: integer (nullable = true)
 |-- distance_cust_to_merch(km): double (nullable = true)
```

```
# drop the 'trans_date_trans_time' column in both fraud_train and fraud_test datasets
fraud_train = fraud_train.drop('trans_date_trans_time')
fraud_test = fraud_test.drop('trans_date_trans_time')
```

Data Encoding

```
from pyspark.ml.feature import StringIndexer, OneHotEncoder, VectorAssembler
from pyspark.ml import Pipeline
from pyspark.sql.functions import col

# define categorical columns
# categorical_columns = ['merchant', 'category', 'gender', 'state', 'job']
categorical_columns = ['category','gender', 'city', 'state', 'job']

# list to hold stages for the pipeline
indexers = []
encoders = []

# stringIndexer for each categorical column
for col_name in categorical_columns:
    indexer = StringIndexer(inputCol=col_name, outputCol=col_name + '_index', handleInvalid="skip")
    encoder = OneHotEncoder(inputCol=col_name + '_index', outputCol=col_name + '_onehot')
    indexers.append(indexer)
    encoders.append(encoder)

# list of numerical columns
numerical_columns = ['amt', 'city_population', 'age', 'transaction_month', 'transaction_hour',
                     'transaction_day', 'distance_cust_to_merch(km)']

# combine all encoded columns with numerical columns
assembler = VectorAssembler(
    inputCols=[col_name + '_onehot' for col_name in categorical_columns] + numerical_columns,
    outputCol="features"
)

# create the Pipeline
pipeline = Pipeline(stages=indexers + encoders + [assembler])

# Fit the pipeline on the training data
pipeline_model = pipeline.fit(fraud_train)

# Transform both the training and test datasets
fraud_train_transformed = pipeline_model.transform(fraud_train)
fraud_test_transformed = pipeline_model.transform(fraud_test)
```

```
from pyspark.ml.feature import StandardScaler

# assemble only numerical features for scaling
numerical_assembler = VectorAssembler(inputCols=numerical_columns, outputCol="numerical_features")

# apply StandardScaler to numerical features
scaler = StandardScaler(inputCol="numerical_features", outputCol="scaled_numerical_features",
                        withMean=True, withStd=True)
```

```
# assemble categorical and scaled numerical features into the final features vector
final_assembler = VectorAssembler(
    inputCols=[f"{col}_onehot" for col in categorical_columns] + ["scaled_numerical_features"],
    outputCol="features"
)

# create the pipeline
pipeline = Pipeline(stages=indexers + encoders + [numerical_assembler, scaler, final_assembler])

# fit the pipeline on the training data
pipeline_model = pipeline.fit(fraud_train)

# transform both the training and test datasets
fraud_train_transformed = pipeline_model.transform(fraud_train)
fraud_test_transformed = pipeline_model.transform(fraud_test)

# select only the final features and target column for modeling
fraud_train_final = fraud_train_transformed.select("features", "is_fraud")
fraud_test_final = fraud_test_transformed.select("features", "is_fraud")
```

```
from pyspark.ml.feature import VectorAssembler
from pyspark.sql import functions as F

# define the list of features you want to check correlation for
features = ['amt', 'city_population', 'age', 'transaction_month', 'transaction_hour',
            'transaction_day', 'distance_cust_to_merch(km)']

# drop the existing 'features' column if it exists
fraud_train_transformed = fraud_train_transformed.drop("features")

# assemble features into a new vector column
assembler = VectorAssembler(inputCols=features, outputCol="new_features")
fraud_train_transformed = assembler.transform(fraud_train_transformed)

# compute correlations between each feature and 'is_fraud'
for feature in features:
    correlation = fraud_train_transformed.stat.corr(feature, 'is_fraud')
    print(f"Correlation between {feature} and is_fraud: {correlation}")
```

```
Correlation between amt and is_fraud: 0.21940388895887128
Correlation between city_population and is_fraud: 0.0021359024181982463
Correlation between age and is_fraud: 0.012378101674716485
Correlation between transaction_month and is_fraud: -0.012409331585155019
Correlation between transaction_hour and is_fraud: 0.01379937052344759
Correlation between transaction_day and is_fraud: 0.009620213899482673
Correlation between distance_cust_to_merch(km) and is_fraud: 0.00043417047602957134
```

Oversampling the Minority Class (Fraudulent transactions)

- First convert the data to pandas dataframe, apply the SMOTE and then convert back to pyspark dataframe.

```
from imblearn.over_sampling import SMOTE
from pyspark.ml.linalg import Vectors
from pyspark.sql import Row
import pandas as pd

# Convert features and target column to Pandas
train_data = fraud_train_transformed.select('new_features', 'is_fraud').toPandas()

# Separate features (as dense arrays) and target variable
X = train_data['new_features'].apply(lambda x: x.toArray()).tolist()
y = train_data['is_fraud']

# Initialize and apply SMOTE
smote = SMOTE(random_state=42)
X_resampled, y_resampled = smote.fit_resample(X, y)

# Initialize Spark session if not already done
spark = SparkSession.builder.getOrCreate()

# Create Pandas DataFrame for the resampled data
resampled_data = pd.DataFrame(X_resampled, columns=[f"new_features_{i}" for i in range(len(X_resampled[0]))])
resampled_data['is_fraud'] = y_resampled

# Convert to PySpark DataFrame
resampled_spark = spark.createDataFrame(
    resampled_data.apply(lambda row: Row(features=Vectors.dense(row[:-1]),
                                         is_fraud=int(row[-1])), axis=1).tolist()
)
```

```
<ipython-input-52-2f9ecc95db63>:27: FutureWarning:

Series.__getitem__ treating keys as positions is deprecated. In a future version, integer keys will always be treated as labels (consistent with DataFrame behavior). To access a value by position, use `ser.iloc[pos]`
```

Modelling

```
# from pyspark.ml.classification import RandomForestClassifier
# from pyspark.ml.feature import VectorAssembler
# from pyspark.ml.evaluation import BinaryClassificationEvaluator
# from pyspark.ml.tuning import ParamGridBuilder, CrossValidator

# # select only 'amt' and 'is_fraud' columns (ensure to include relevant columns)
# fraud_train_two_features = fraud_train_transformed.select('amt', 'is_fraud')
# fraud_test_two_features = fraud_test_transformed.select('amt', 'is_fraud')

# # assemble 'amt' into a feature vector
# assembler = VectorAssembler(inputCols=['amt'], outputCol='features')
# fraud_train_two_features = assembler.transform(fraud_train_two_features)
# fraud_test_two_features = assembler.transform(fraud_test_two_features)

# # initialize the random forest classifier
# rf = RandomForestClassifier(featuresCol='features', labelCol='is_fraud', maxDepth=20, numTrees=100, maxBins=32)

# # Create a param grid for hyperparameter tuning
# paramGrid = ParamGridBuilder()\
#     .addGrid(rf.numTrees, [50, 100])\
#     .addGrid(rf.maxDepth, [5, 10])\
#     .addGrid(rf.maxBins, [16, 32])\
#     .build()

# # Initialize the evaluator (use ROC-AUC for evaluation)
# evaluator = BinaryClassificationEvaluator(labelCol='is_fraud')

# # Set up cross-validation
# cv = CrossValidator(estimator=rf, estimatorParamMaps=paramGrid, evaluator=evaluator, numFolds=3)

# # Train the model with cross-validation
# cv_model = cv.fit(fraud_train_two_features)

# # Make predictions
# predictions = cv_model.transform(fraud_test_two_features)

# # Evaluate the model
# roc_auc = evaluator.evaluate(predictions)

# print(f"ROC-AUC on test data: {roc_auc}")
```

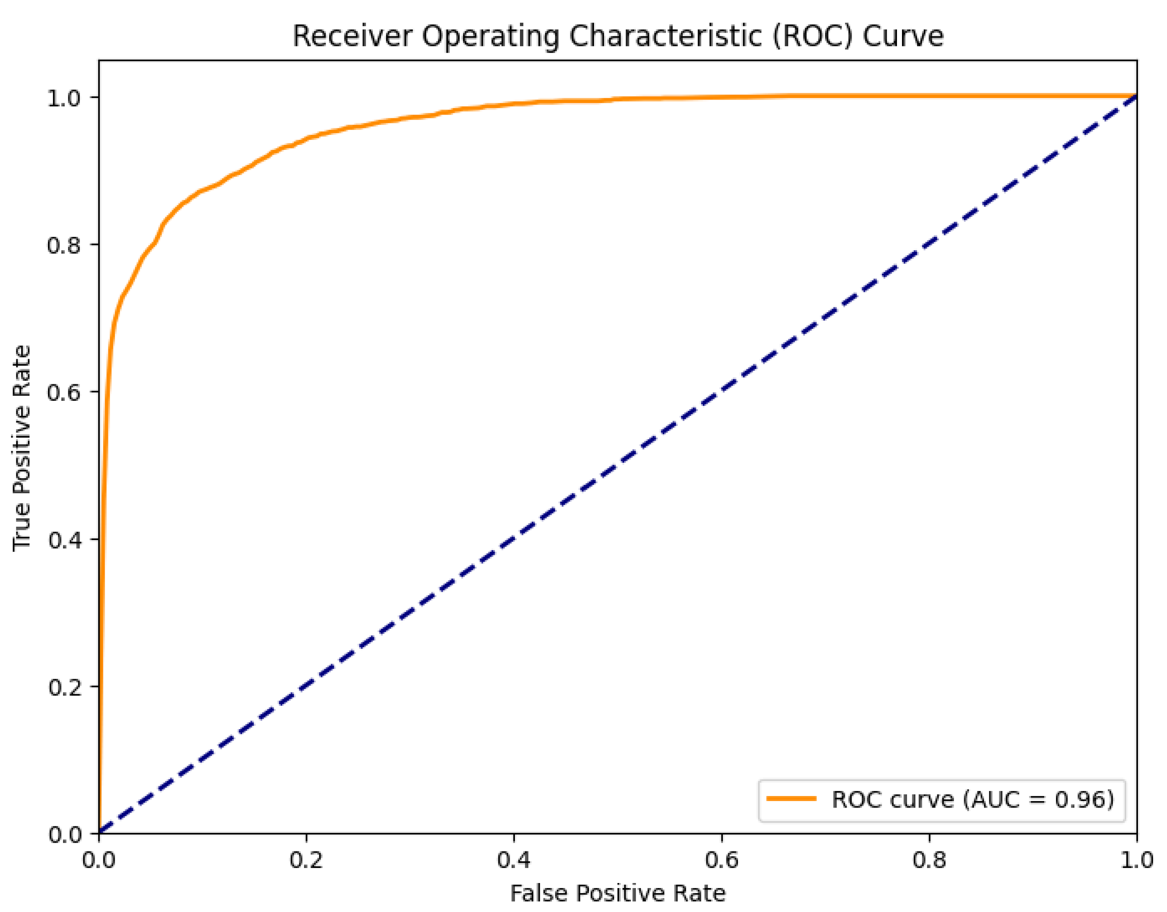
```
import xgboost as xgb
from pyspark.ml.feature import VectorAssembler
from pyspark.ml.evaluation import BinaryClassificationEvaluator
from pyspark.sql import functions as F

# select only 'amt' and 'is_fraud' columns (ensure to include relevant columns)
fraud_train_two_features = fraud_train_transformed.select('amt', 'is_fraud')
fraud_test_two_features = fraud_test_transformed.select('amt', 'is_fraud')

# assemble 'amt' into a feature vector (since 'age' is not selected, only 'amt' is included)
assembler = VectorAssembler(inputCols=['amt'], outputCol='features')
fraud_train_two_features = assembler.transform(fraud_train_two_features)
fraud_test_two_features = assembler.transform(fraud_test_two_features)

# Convert features column to RDD for XGBoost
train_data = fraud_train_two_features.select('features', 'is_fraud').rdd.map(lambda x: (x[0].toArray(), x[1]))
test_data = fraud_test_two_features.select('features', 'is_fraud').rdd.map(lambda x: (x[0].toArray(), x[1]))

# Prepare data for XGBoost
train_X = [x[0] for x in train_data.collect()]
train_y = [x[1] for x in train_data.collect()]
test_X = [x[0] for x in test_data.collect()]
test_y = [x[1] for x in test_data.collect()]

# Convert to DMatrix format (XGBoost's internal format)
train_dmatrix = xgb.DMatrix(train_X, label=train_y)
test_dmatrix = xgb.DMatrix(test_X, label=test_y)
```

```
# Set XGBoost hyperparameters
params = {
    'objective': 'binary:logistic',
    'eval_metric': 'auc',
    'max_depth': 6,
    'learning_rate': 0.1
}

# Train the model
xgb_model = xgb.train(params, train_dmatrix, num_boost_round=100)

# Make predictions
predictions = xgb_model.predict(test_dmatrix)

# Evaluate the model using AUC (since it's a binary classification task)
from sklearn.metrics import roc_auc_score

roc_auc = roc_auc_score(test_y, predictions)

print(f"ROC-AUC on test data: {roc_auc}")
```

```
/usr/local/lib/python3.11/dist-packages/xgboost/core.py:158: UserWarning:

[18:13:50] WARNING: /workspace/src/learner.cc:740:
Parameters: { "n_estimators" } are not used.


ROC-AUC on test data: 0.9589943243890668
```

```
import xgboost as xgb
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, confusion_matrix, roc_curve, auc
import matplotlib.pyplot as plt
import seaborn as sns

# Make predictions
predictions = xgb_model.predict(test_dmatrix)

# Evaluate the model using additional metrics
threshold = 0.5  # You can experiment with this threshold
predicted_classes = (predictions > threshold).astype(int)

# Calculate Accuracy, Precision, Recall, F1-Score
accuracy = accuracy_score(test_y, predicted_classes)
precision = precision_score(test_y, predicted_classes)
recall = recall_score(test_y, predicted_classes)
f1 = f1_score(test_y, predicted_classes)


# ROC Curve
fpr, tpr, _ = roc_curve(test_y, predictions)
roc_auc = auc(fpr, tpr)

# Plot the ROC Curve
plt.figure(figsize=(8, 6))
plt.plot(fpr, tpr, color='darkorange', lw=2, label=f'ROC curve (AUC = {roc_auc:.2f})')
plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic (ROC) Curve')
plt.legend(loc='lower right')
plt.show()

# Print evaluation metrics
print(f"Accuracy: {accuracy:.4f}")
print(f"Precision: {precision:.4f}")
print(f"Recall: {recall:.4f}")
print(f"F1-Score: {f1:.4f}")
```

```
/usr/local/lib/python3.11/dist-packages/sklearn/metrics/_classification.py:1565: UndefinedMetricWarning:

Precision is ill-defined and being set to 0.0 due to no predicted samples. Use `zero_division` parameter to control this behavior.
```




```
Accuracy: 0.9964
Precision: 0.0000
Recall: 0.0000
```