Data Preparation for Fraud Detection

This notebook is dedicated to preparing the dataset for a fraud detection model. The focus is on cleaning, transforming, and inital exploring the data to ensure it's well-suited for effective model training.

In [ ]:
```
# import required libraries
import polars as pl

# adjust display setting polars
pl.Config.set_tbl_cols(-1)
```

Out[ ]:
```
polars.config.Config
```

Initial Exploration

We start with an initial exploratory to understand the dataset's structure, identify any immediate issues, and plan for necessary data cleaning steps.

In [ ]:
```
# load the historical transactions
transactions_df = pl.read_csv("../raw-datasets/credit_card_transactions-ibm_v2.csv")

# display it
transactions_df
```

Out[ ]: shape: (24_386_900, 15)

User	Card	Year	Month	Day	Time	Amount	Use Chip	Merchant Name	Merchant City	Merchant State	Zip	MCC	Errors?	Is Fraud?
i64	i64	i64	i64	i64	str	str	str	i64	str	str	f64	i64	str	str
0	0	2002	9	1	"06:21"	"$134.09"	"Swipe Transact...	3527213246127876953	"La Verne"	"CA"	91750.0	5300	null	"No"
0	0	2002	9	1	"06:42"	"$38.48"	"Swipe Transact...	-727612092139916043	"Monterey Park"	"CA"	91754.0	5411	null	"No"
0	0	2002	9	2	"06:22"	"$120.34"	"Swipe Transact...	-727612092139916043	"Monterey Park"	"CA"	91754.0	5411	null	"No"
0	0	2002	9	2	"17:45"	"$128.95"	"Swipe Transact...	3414527459579106770	"Monterey Park"	"CA"	91754.0	5651	null	"No"
0	0	2002	9	3	"06:23"	"$104.71"	"Swipe Transact...	5817218446178736267	"La Verne"	"CA"	91750.0	5912	null	"No"
0	0	2002	9	3	"13:53"	"$86.19"	"Swipe Transact...	-7146670748125200898	"Monterey Park"	"CA"	91755.0	5970	null	"No"
0	0	2002	9	4	"05:51"	"$93.84"	"Swipe Transact...	-727612092139916043	"Monterey Park"	"CA"	91754.0	5411	null	"No"
0	0	2002	9	4	"06:09"	"$123.50"	"Swipe Transact...	-727612092139916043	"Monterey Park"	"CA"	91754.0	5411	null	"No"
0	0	2002	9	5	"06:14"	"$61.72"	"Swipe Transact...	-727612092139916043	"Monterey Park"	"CA"	91754.0	5411	null	"No"
0	0	2002	9	5	"09:35"	"$57.10"	"Swipe Transact...	4055257078481058705	"La Verne"	"CA"	91750.0	7538	null	"No"
0	0	2002	9	5	"20:18"	"$76.07"	"Swipe Transact...	-4500542936415012428	"La Verne"	"CA"	91750.0	5814	null	"No"
0	0	2002	9	5	"20:41"	"$53.91"	"Online Transac...	-9092677072201095172	"ONLINE"	null	null	4900	null	"No"
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
1999	1	2020	2	26	"20:18"	"$44.54"	"Chip Transacti...	2500998799892805156	"Merrimack"	"NH"	3054.0	4121	null	"No"
1999	1	2020	2	27	"07:47"	"$47.18"	"Online Transac...	-5841929396161652653	"ONLINE"	null	null	4121	null	"No"
1999	1	2020	2	27	"09:31"	"$120.00"	"Chip Transacti...	-4282466774399734331	"Berlin"	"NH"	3570.0	4829	null	"No"
1999	1	2020	2	27	"11:36"	"$12.91"	"Chip Transacti...	3414527459579106770	"Nashua"	"NH"	3064.0	5651	null	"No"
1999	1	2020	2	27	"20:18"	"$15.52"	"Chip Transacti...	97032797689821735	"Merrimack"	"NH"	3054.0	5411	null	"No"
1999	1	2020	2	27	"20:29"	"$56.67"	"Chip Transacti...	2500998799892805156	"Merrimack"	"NH"	3054.0	4121	null	"No"
1999	1	2020	2	27	"22:18"	"$63.43"	"Chip Transacti...	-5162038175624867091	"Merrimack"	"NH"	3054.0	5541	null	"No"
1999	1	2020	2	27	"22:23"	"$-54.00"	"Chip Transacti...	-5162038175624867091	"Merrimack"	"NH"	3054.0	5541	null	"No"
1999	1	2020	2	27	"22:24"	"$54.00"	"Chip Transacti...	-5162038175624867091	"Merrimack"	"NH"	3054.0	5541	null	"No"
1999	1	2020	2	28	"07:43"	"$59.15"	"Chip Transacti...	2500998799892805156	"Merrimack"	"NH"	3054.0	4121	null	"No"
1999	1	2020	2	28	"20:10"	"$43.12"	"Chip Transacti...	2500998799892805156	"Merrimack"	"NH"	3054.0	4121	null	"No"
1999	1	2020	2	28	"23:10"	"$45.13"	"Chip Transacti...	4751695835751691036	"Merrimack"	"NH"	3054.0	5814	null	"No"

Overview: With approximately 24.4 million transactions, this dataset offers a comprehensive set of data points. The upcoming steps involve thorough data cleaning and transformation to shape the data for optimal use in machine learning model training.

We'll conduct an initial exploration to get a sense of the dataset's quality, identify any missing values, and understand the overall data distribution.

In [ ]:
```
# calculate of missing values in each column
missing_values = transactions_df.select([pl.col(column).is_null().sum().alias(column) for column in transactions_df.columns])
```

```
print("Percentage of Missing Values:")
missing_values / len(transactions_df) * 100
```

```
Percentage of Missing Values:
```

Out[ ]: shape: (1, 15)

User	Card	Year	Month	Day	Time	Amount	Use Chip	Merchant Name	Merchant City	Merchant State	Zip	MCC	Errors?	Is Fraud?
f64	f64	f64	f64	f64	f64	f64	f64	f64	f64	f64	f64	f64	f64	f64
0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	11.156896	11.801972	0.0	98.407215	0.0

We observed a significant number of missing values in the `Errors?` column. It seems likely that these null values indicate transactions without any errors, implying successful transactions.

In [ ]:
```
print("`Is Fraud?` proportion: \n",
      transactions_df.group_by("Is Fraud?")
      .agg(pl.count().alias("count"))
      .with_columns([
          (pl.col("count") / len(transactions_df) * 100).alias("percentage")
          ])
      .sort("count", descending=True))
```

```
`Is Fraud?` proportion:
 shape: (2, 3)
```

Is Fraud? --- str	count --- u32	percentage --- f64
No	24357143	99.87798
Yes	29757	0.12202

Our target column, `Is Fraud?`, shows a distribution of 99.87% (24,357,143 transactions) as non-fraudulent and only 0.13% (29,757 transactions) as fraudulent. This distribution aligns well with real-world scenarios where fraudulent transactions are relatively rare.

In [ ]:
```
print("`Use Chip` proportion: \n",
      transactions_df.group_by("Use Chip")
      .agg(pl.count().alias("count"))
      .with_columns([
          (pl.col("count") / len(transactions_df) * 100).alias("percentage")
          ])
      .sort("count", descending=True))
```

```
`Use Chip` proportion:
 shape: (3, 3)
```

Use Chip --- str	count --- u32	percentage --- f64
Swipe Transaction	15386082	63.091586
Chip Transaction	6287598	25.782687
Online Transaction	2713220	11.125727

The `Use Chip` column represents the transaction method used by the customer. Our analysis revealed that Swipe Transactions account for 63.09%, Chip Transactions for 25.79%, and Online Transactions for 11.12%.

In [ ]:
```python
print("`Merchant State` proportion: \n",
      transactions_df.group_by("Merchant State")
      .agg(pl.count().alias("count"))
      .with_columns([
          (pl.col("count") / len(transactions_df) * 100).alias("percentage")
          ])
      .sort("count", descending=True))
```

```
`Merchant State` proportion:
 shape: (224, 3)
```

Merchant State --- str	count --- u32	percentage --- f64
null	2720821	11.156896
CA	2591830	10.62796
TX	1793298	7.35353
FL	1458699	5.981486
…	…	…
Tonga	2	0.000008
Botswana	1	0.000004
Kiribati	1	0.000004
Paraguay	1	0.000004

In the `Merchant State` column, we noticed that null values comprise the largest proportion (11.15%), followed by California (10.62%) and Texas (7.3%). With approximately 224 unique values, it's clear that transactions occur not only in the USA but also in other countries, as indicated by entries like Paraguay, Togo, etc.

In [ ]:
```python
print("`Merchant City` proportion: \n",
      transactions_df.group_by("Merchant City")
      .agg(pl.count().alias("count"))
      .with_columns([
          (pl.col("count") / len(transactions_df) * 100).alias("percentage")
          ])
      .sort("count", descending=True))
```

```
`Merchant City` proportion:
 shape: (13_429, 3)
```

Merchant City --- str	count --- u32	percentage --- f64
ONLINE	2720821	11.156896
Houston	246036	1.008886
Los Angeles	180496	0.740135
Miami	178653	0.732578
…	…	…
Boyne Falls	1	0.000004
Weyerhaeuser	1	0.000004
Poyen	1	0.000004
Long Bottom	1	0.000004

The `Merchant City` column also follows a similar pattern, with 'Online' being the most common category at 11.15%. Other frequent locations include U.S. cities such as Houston, Los Angeles, and Miami. The presence of 13,429 unique city names suggests that transactions span globally, not just within the USA.

In [ ]:
```
print("`Errors?` proportion: \n",
      transactions_df.group_by("Errors?")
      .agg(pl.count().alias("count"))
      .with_columns([
          (pl.col("count") / len(transactions_df) * 100).alias("percentage")
          ])
      .sort("count", descending=True))
```

```
`Errors?` proportion:
 shape: (24, 3)
┌───────────────────────────────────┬──────────┬────────────┐
│ Errors?                           ┆ count    ┆ percentage │
│ ---                               ┆ ---      ┆ ---        │
│ str                               ┆ u32      ┆ f64        │
╞═══════════════════════════════════╪══════════╪════════════╡
│ null                              ┆ 23998469 ┆ 98.407215  │
│ Insufficient Balance              ┆ 242783   ┆ 0.995547   │
│ Bad PIN                           ┆ 58918    ┆ 0.241597   │
│ Technical Glitch                  ┆ 48157    ┆ 0.197471   │
│ …                                 ┆ …        ┆ …          │
│ Bad Zipcode,Insufficient Balance  ┆ 13       ┆ 0.000053   │
│ Bad Zipcode,Technical Glitch      ┆ 7        ┆ 0.000029   │
│ Bad Card Number,Bad Expiration,I… ┆ 2        ┆ 0.000008   │
│ Bad Card Number,Bad Expiration,T… ┆ 1        ┆ 0.000004   │
└───────────────────────────────────┴──────────┴────────────┘
```

Consistent with earlier observations, the `Errors?` column indicates that 98.40% of transactions have no recorded errors, implying successful transactions. This column also includes other error categories like insufficient balance, bad pin, etc., totaling 24 distinct categories.

In [ ]:
```
print('Summary statistics: ')
transactions_df.describe()
```

```
Summary statistics:
```

Out[ ]: shape: (9, 16)

describe	User	Card	Year	Month	Day	Time	Amount	Use Chip	Merchant Name	Merchant City	Merchant State	Zip	MCC	Errors?	Is Fraud?
str	f64	f64	f64	f64	f64	str	str	str	f64	str	str	f64	f64	str	str
"count"	2.43869e7	2.43869e7	2.43869e7	2.43869e7	2.43869e7	"24386900"	"24386900"	"24386900"	2.43869e7	"24386900"	"21666079"	2.1508765e7	2.43869e7	"388431"	"24386900"
"null_count"	0.0	0.0	0.0	0.0	0.0	"0"	"0"	"0"	0.0	"0"	"2720821"	2.878135e6	0.0	"23998469"	"0"
"mean"	1001.019335	1.351366	2011.95517	6.525064	15.718123	null	null	null	-4.7692e17	null	null	50956.442115	5561.171253	null	null
"std"	569.461157	1.407154	5.105921	3.472355	8.794073	null	null	null	4.7589e18	null	null	29397.065949	879.315433	null	null
"min"	0.0	0.0	1991.0	1.0	1.0	"00:00"	"$-0.00"	"Chip Transacti…	-9.2229e18	"Aaronsburg"	"AA"	501.0	1711.0	"Bad CVV"	"No"
"25%"	510.0	0.0	2008.0	3.0	8.0	null	null	null	-4.5005e18	null	null	28374.0	5300.0	null	null
"50%"	1006.0	1.0	2013.0	7.0	16.0	null	null	null	-7.9468e17	null	null	46742.0	5499.0	null	null
"75%"	1477.0	2.0	2016.0	10.0	23.0	null	null	null	3.1895e18	null	null	77564.0	5812.0	null	null
"max"	1999.0	8.0	2020.0	12.0	31.0	"23:59"	"$999.97"	"Swipe Transact…	9.2233e18	"Zwolle"	"Zimbabwe"	99928.0	9402.0	"Technical Glit…	"Yes"

From the summary statistics presented above, it's challenging to discern clear insights due to the raw and unclean state of the data. Post-cleanup, we will conduct another round of summary statistics to better understand the dataset.

Data Cleaning and Features Transformation

In [ ]:
```
# clone the dataframe
transactions_prepared = transactions_df.clone()
```

We are set to perform initial data cleaning tasks. These include removing unnecessary characters, typecasting columns for consistency, and creating new features to enrich our dataset.

In [ ]:
```
# type casting, remove unnecesarry characters, create new datetime column
transactions_prepared = transactions_prepared.with_columns([
    pl.col("Amount").str.strip_chars("$").cast(pl.Float64).abs(), # remove dollar sign, cast to float, turn into absolute incase there's minus trx which is not make sense
    pl.col("Merchant Name").cast(pl.Utf8), # cast to string
    pl.col("MCC").cast(pl.Utf8).cast(pl.Categorical), # cast to categorical
    pl.col("Use Chip").cast(pl.Categorical),
    pl.col("Errors?").cast(pl.Categorical),

    # create 'timestamp_transaction' column
    (pl.col("Year").cast(pl.Utf8) + "-" +
     pl.col("Month").cast(pl.Utf8).str.zfill(2) + "-" +  # add leading zero
     pl.col("Day").cast(pl.Utf8).str.zfill(2) + " " +
     pl.col("Time")).str.strptime(pl.Datetime, "%Y-%m-%d %H:%M").alias("timestamp_transaction"),
]).drop(["Year", "Month", "Day", "Time"])  # drop unnecessary columns

# extract datetime information
# fill missing values
transactions_prepared = transactions_prepared.with_columns([
    pl.col("timestamp_transaction").dt.weekday().alias("weekday"),
    pl.col("timestamp_transaction").dt.hour().alias("hour"),
    pl.col("timestamp_transaction").dt.minute().alias("minute"),
    pl.col("Errors?").fill_null("No Error"), # fill missing values as No Error
    pl.col("Merchant City").fill_null("NA"), # fill null values as NA
    pl.col("Merchant State").fill_null("NA"),
    pl.col("Zip").fill_null("NA").str.strip_chars(".0"),
])
```

```
sys:1: CategoricalRemappingWarning: Local categoricals have different encodings, expensive re-encoding is done to perform this merge operation. Consider using a StringCache or an Enum type if the categories are known in advance
```

In [ ]:
```
# rename columns
transactions_prepared = transactions_prepared.rename({
    "User":"user_id", "Card":"card_index", "Amount":"trx_amount", "Use Chip":"trx_method",
    "Merchant Name":"merch_name", "Merchant State":"merch_state", "Merchant City":"merch_city", "Zip":"zip_code",
    "MCC":"merch_category_code", "Errors?":"error_status", "Is Fraud?":"is_fraud"
})

transactions_prepared.head()
```

Out[ ]: shape: (5, 15)

user_id	card_index	trx_amount	trx_method	merch_name	merch_city	merch_state	zip_code	merch_category_code	error_status	is_fraud	timestamp_transaction	weekday	hour	minute
i64	i64	f64	cat	str	str	str	str	cat	cat	str	datetime[μs]	i8	i8	i8
0	0	134.09	"Swipe Transact...	"35272132461278...	"La Verne"	"CA"	"9175"	"5300"	"No Error"	"No"	2002-09-01 06:21:00	7	6	21
0	0	38.48	"Swipe Transact...	"-7276120921399...	"Monterey Park"	"CA"	"91754"	"5411"	"No Error"	"No"	2002-09-01 06:42:00	7	6	42
0	0	120.34	"Swipe Transact...	"-7276120921399...	"Monterey Park"	"CA"	"91754"	"5411"	"No Error"	"No"	2002-09-02 06:22:00	1	6	22
0	0	128.95	"Swipe Transact...	"34145274595791...	"Monterey Park"	"CA"	"91754"	"5651"	"No Error"	"No"	2002-09-02 17:45:00	1	17	45
0	0	104.71	"Swipe Transact...	"58172184461787...	"La Verne"	"CA"	"9175"	"5912"	"No Error"	"No"	2002-09-03 06:23:00	2	6	23

Columns are renamed for clarity and consistency, which aids in making the dataset more understandable and easier to work with during further analysis and modeling.

In [ ]:
```
print('Summary statistics: ')
transactions_prepared.describe()
```

```
Summary statistics:
```

Out[ ]: shape: (9, 16)

describe	user_id	card_index	trx_amount	trx_method	merch_name	merch_city	merch_state	zip_code	merch_category_code	error_status	is_fraud	timestamp_transaction	weekday	h
str	f64	f64	f64	str	str	str	str	str	str	str	str	str	f64	
"count"	2.43869e7	2.43869e7	2.43869e7	"24386900"	"24386900"	"24386900"	"24386900"	"24386900"	"24386900"	"24386900"	"24386900"	"24386900"	2.43869e7	2.438
"null_count"	0.0	0.0	0.0	"0"	"0"	"0"	"0"	"0"	"0"	"0"	"0"	"0"	0.0	
"mean"	1001.019335	1.351366	54.051265	null	null	null	null	null	null	null	null	null	4.003477	12.4
"std"	569.461157	1.407154	75.564936	null	null	null	null	null	null	null	null	null	1.999703	5.06
"min"	0.0	0.0	0.0	null	"-1000080909058...	"Aaronsburg"	"AA"	"10001"	null	null	"No"	"1991-01-02 07:...	1.0	
"25%"	510.0	0.0	12.22	null	null	null	null	null	null	null	null	null	2.0	
"50%"	1006.0	1.0	36.17	null	null	null	null	null	null	null	null	null	4.0	
"75%"	1477.0	2.0	72.0	null	null	null	null	null	null	null	null	null	6.0	
"max"	1999.0	8.0	12390.5	null	"99968297410928...	"Zwolle"	"Zimbabwe"	"NA"	null	null	"Yes"	"2020-02-28 23:...	7.0	

In the prepared dataframe, the summary statistics are now more insightful, particularly for the `trx_amount` and `timestamp_transaction` columns. Notably, the transaction amount varies significantly, with the smallest being $0andthelargestreachinganimpressive$12,390.50. The dataset spans a substantial period, from 1991 to 2020, indicating a wide timeframe for the transactions.

We will export this prepared dataframe as a Parquet file. Opting for Parquet format offers the benefits of space efficiency and faster I/O. The saved file will be utilized for more in-depth Exploratory Data Analysis (EDA) in a separate notebook, followed by the development of our machine learning model.

In [ ]:
```
transactions_prepared.write_parquet("../clean-datasets/transactions_prepared.parquet")
```