Credit Card Fraud Detection

Zaki Alawami

October 21, 2021

Kaplan/SDAIA Data Science Bootcamp

AGENDA

- Problem Statement
- Exploratory Data Analysis (EDA)
- Performance Metrics
- Model Selection & Training
- Summary of Findings
- Q&A

Problem Statement

- Credit card providers, Banks, and Merchants can save billions of dollars that is lost annually to credit card fraud.
- More importantly credit card holders (customers) will have a higher level of satisfaction, trust, and loyalty to their credit card providers if the latter can effectively detect and prevent fraudulent transactions, without annoying their customers with blocked legitimate payments.

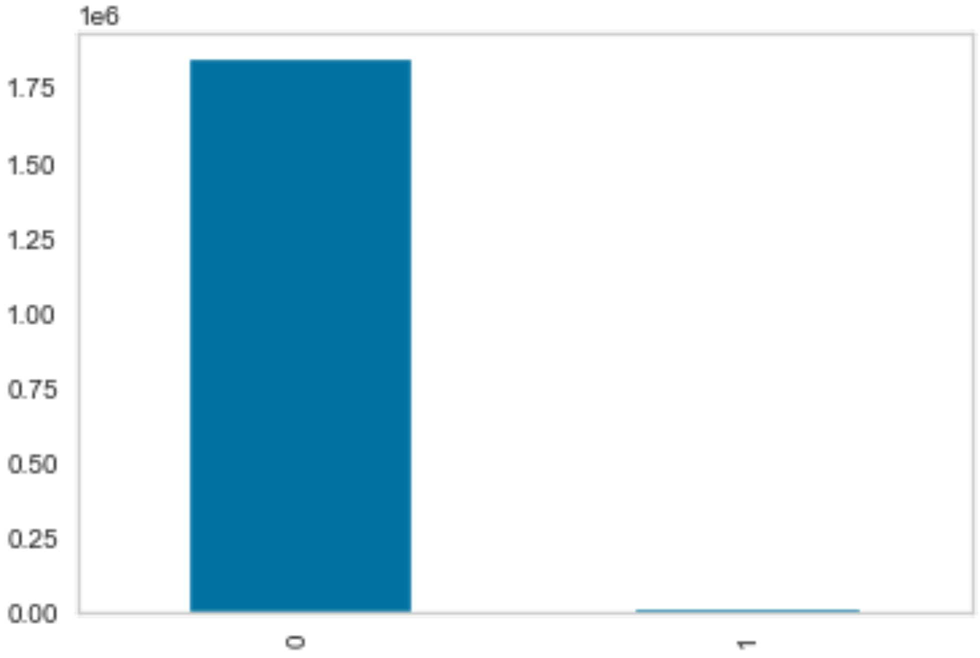
- How to prevent and manage credit card fraud?
  - Analyze a credit card transactions dataset
  - Discover insights
  - Create a classification model to predict fraud transactions.

Dataset Description

- [Kaggle Dataset]()

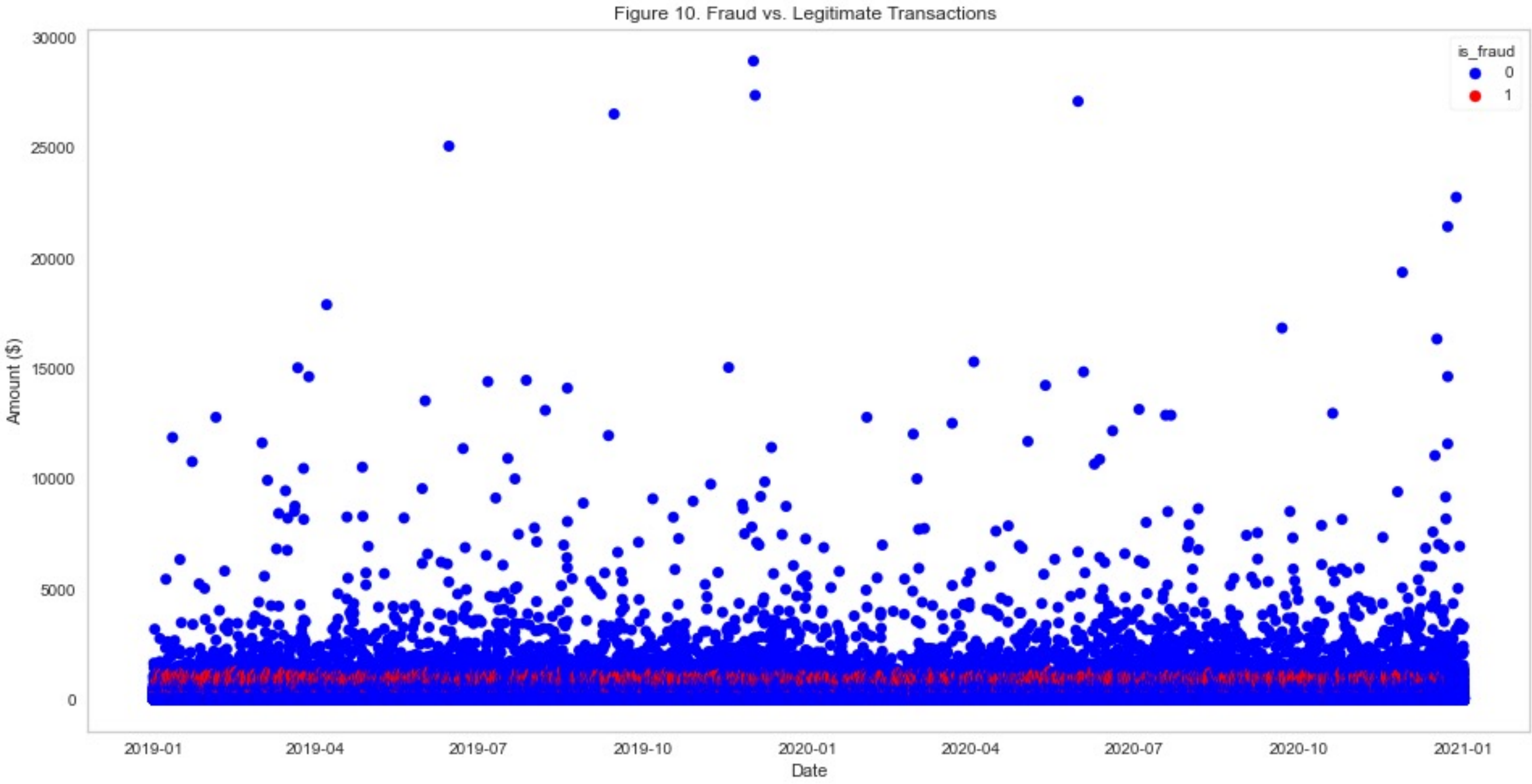
- Imbalanced Dataset:
    - 21 columns/features
    - one target (valid/fraud)
    - ~1.84 Million records
    - Fraud class is 0.52%

- Features:

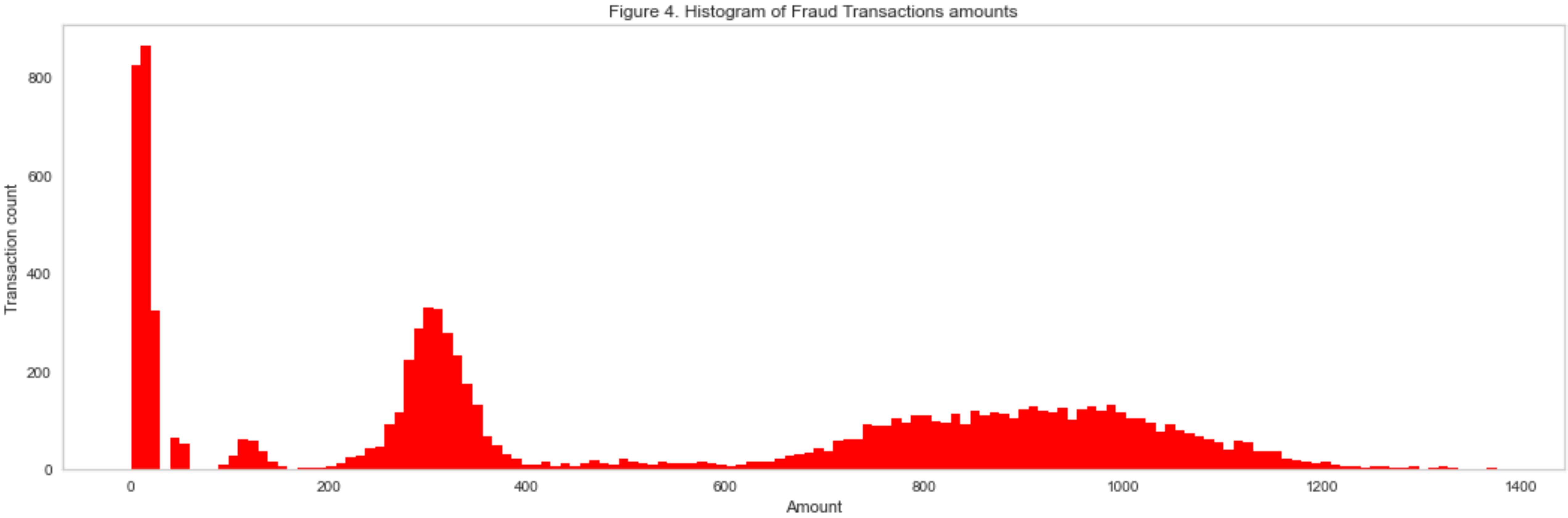


trans_date_trans_time	cc_num	merchant	category	amt	first	last	gender	street	city	state	zip	lat	long	city_pop	job	dob	trans_num	unix_time	merch_lat	merch_long	is_fraud

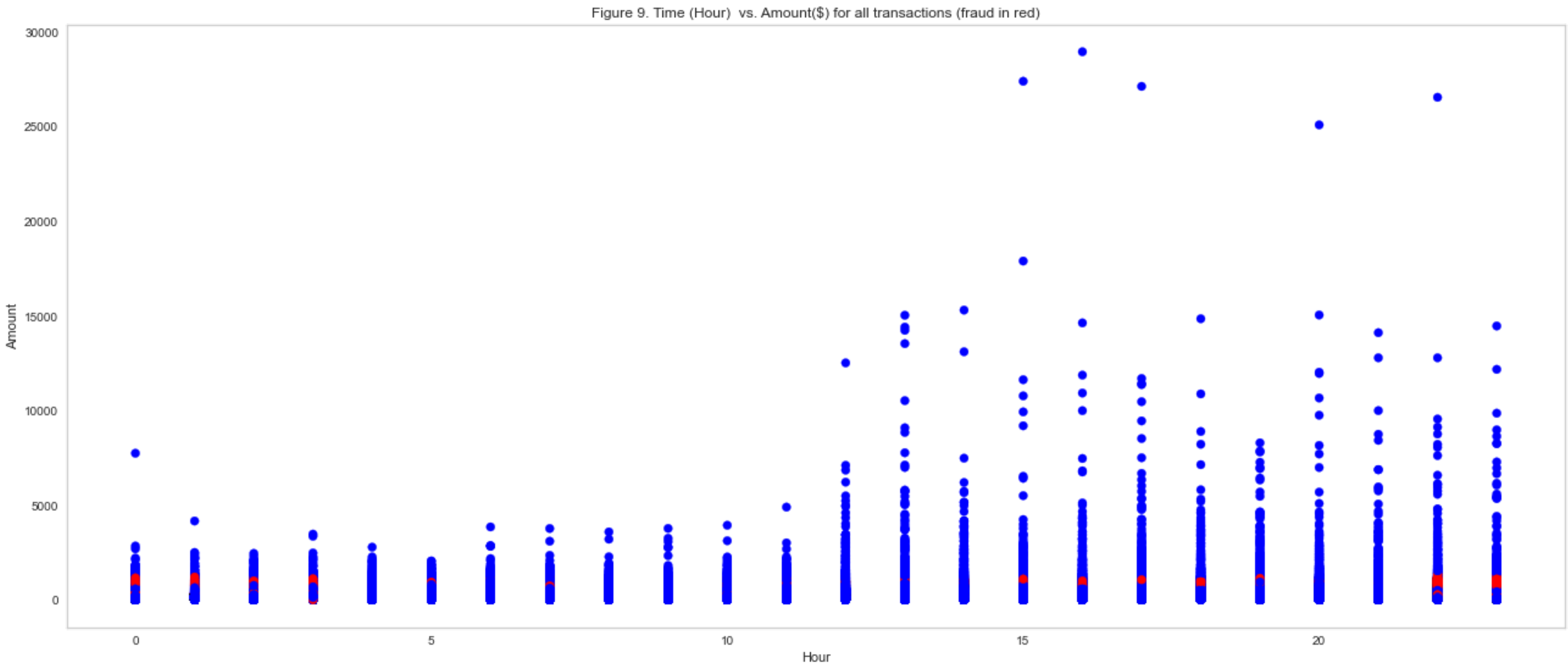
Exploratory Data Analysis



Exploratory Data Analysis

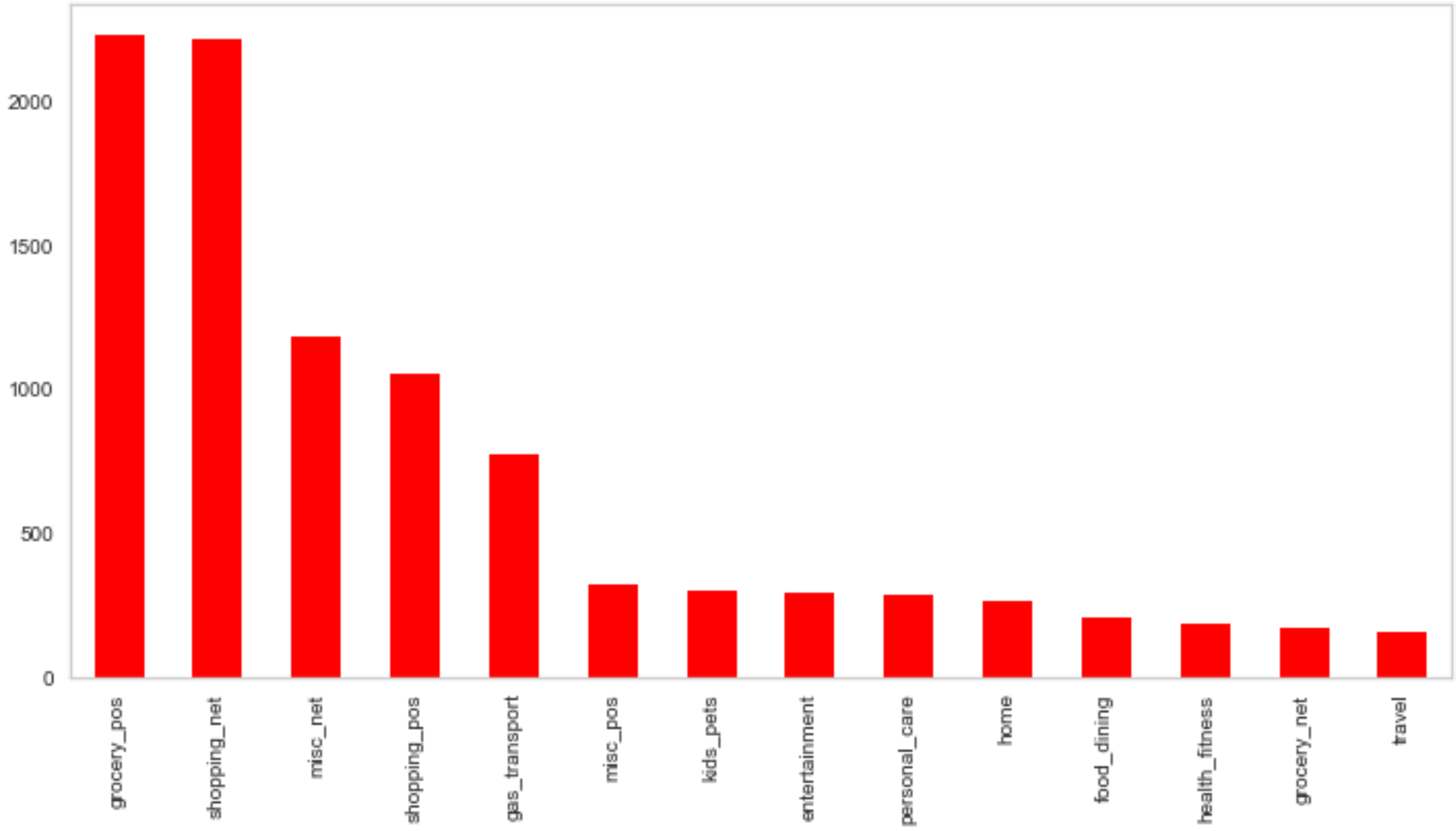


Exploratory Data Analysis

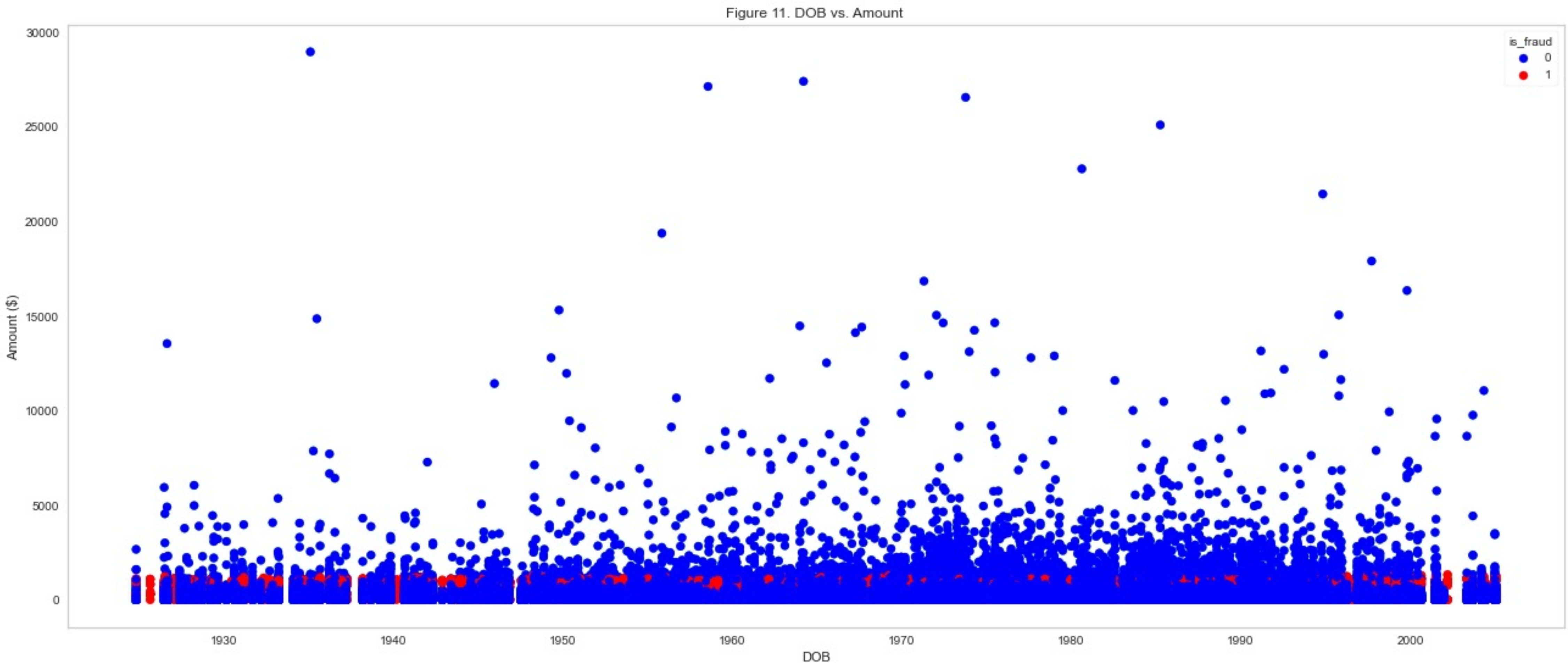


Exploratory Data Analysis

Figure 7. Histogram of fraud transaction categories

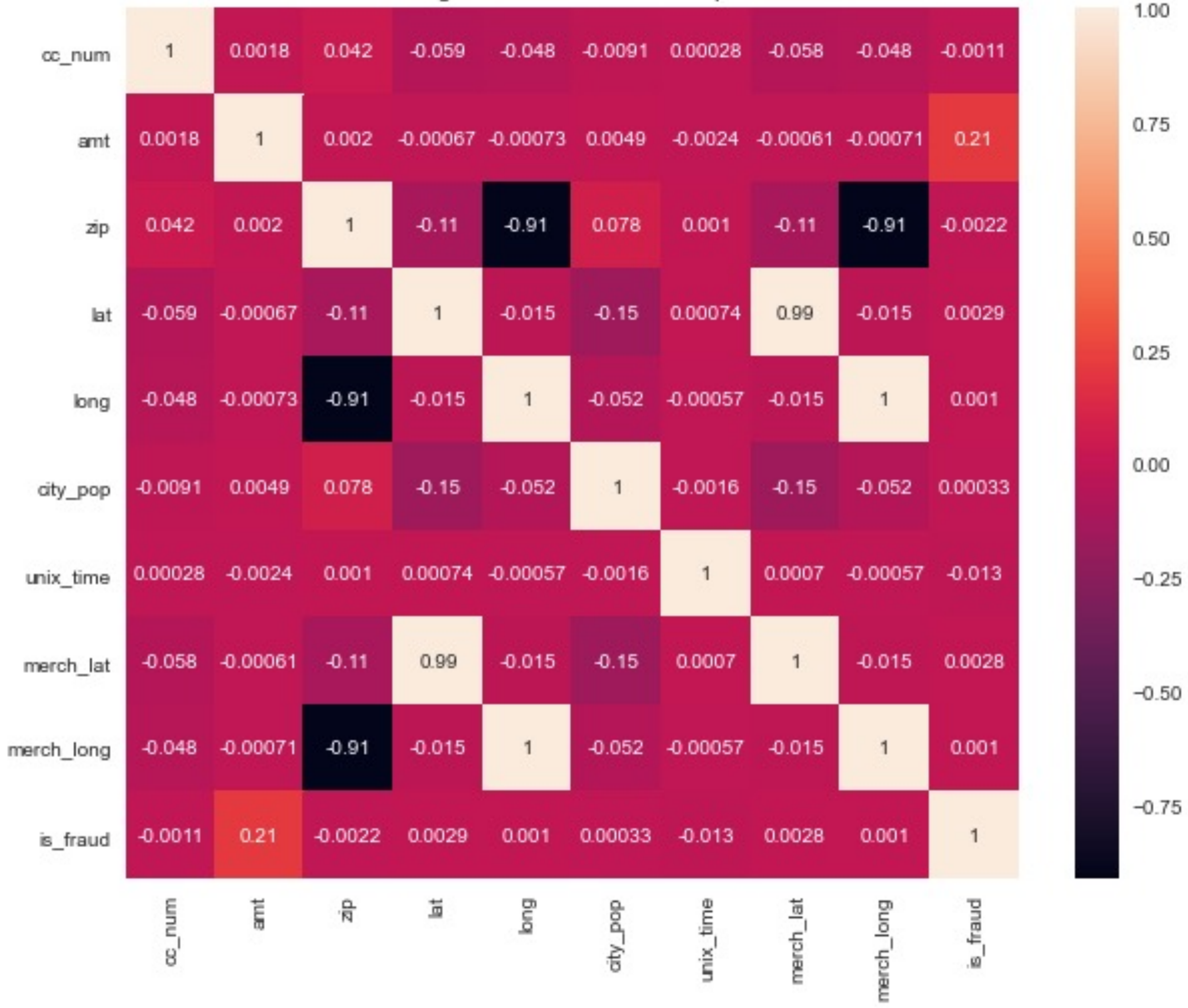


Exploratory Data Analysis

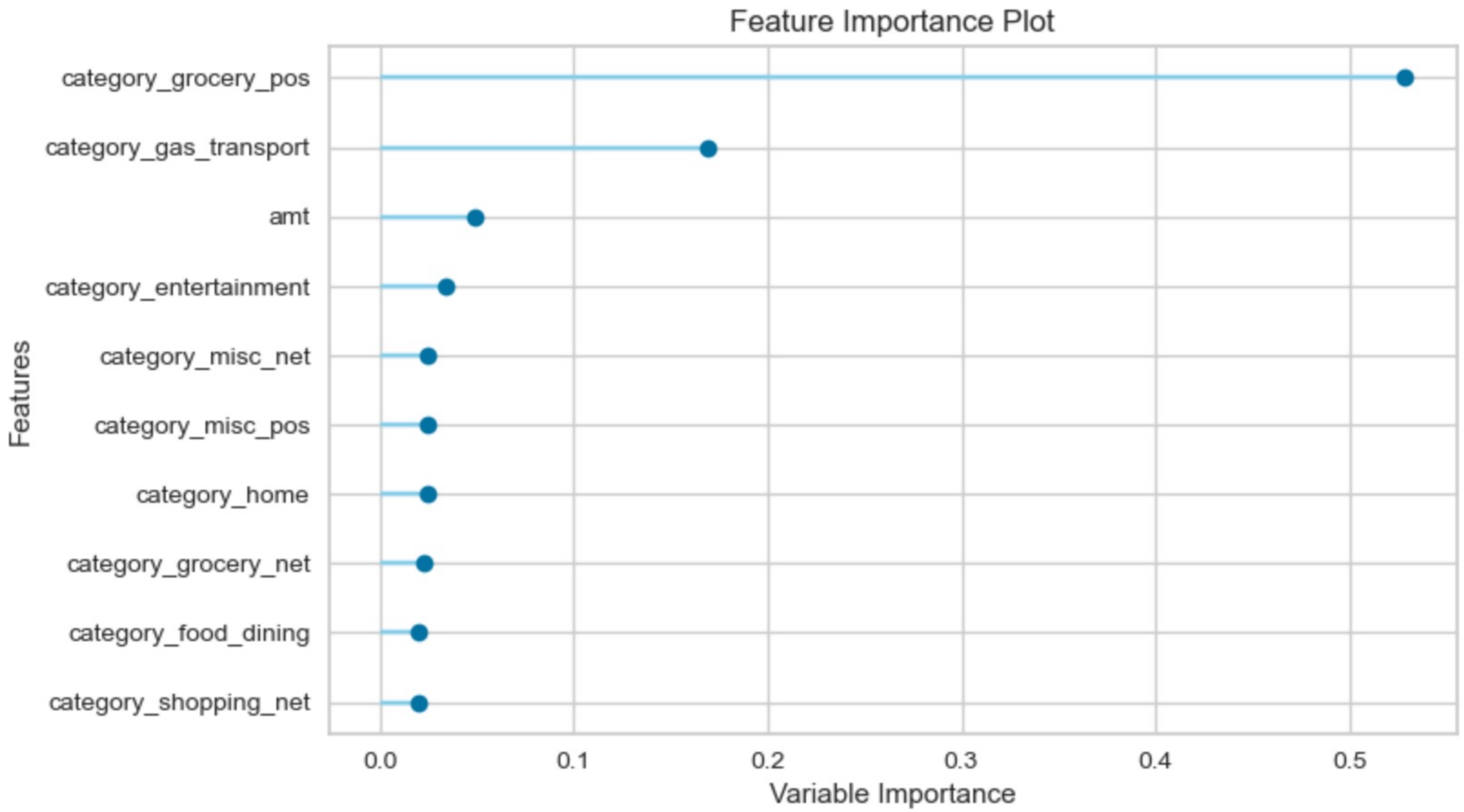


Exploratory Data Analysis

Figure 12. Seaborn Heatmap



Feature Importance



Performance Metrics

Precision	• The fraction of correct positive predictions
Recall	• The fraction of positive cases predicted correctly. • High recall means you are confident you didn't miss any positive cases.
F1 score	Balance of precision vs. recall
LTPFR	Legitimate Transactions Predicted as Fraud Rate: (FP/ FP + FN)

Models Selection & Performance

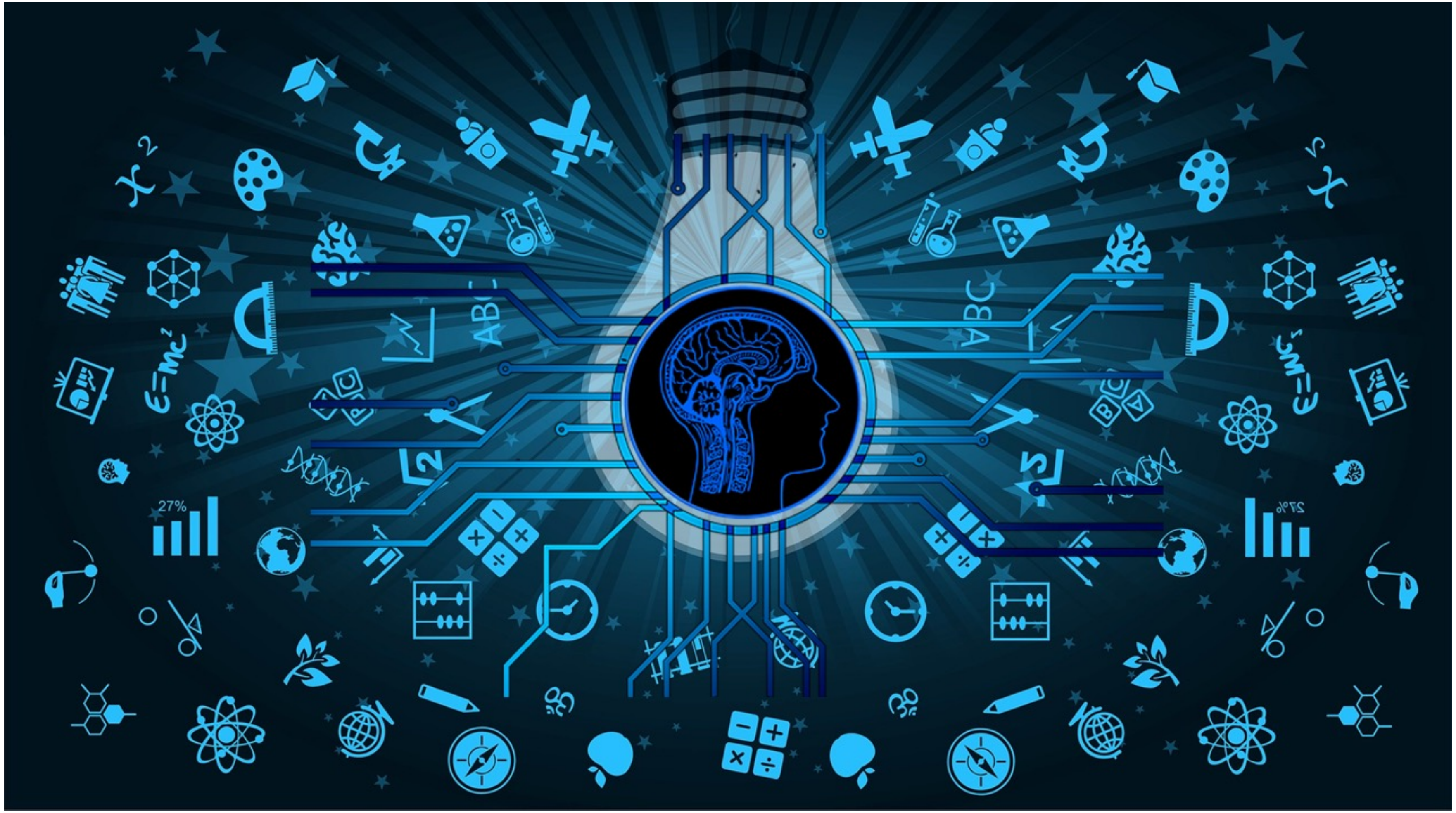
Model	Features	Accuracy	Precision	Recall	F1	LTPFR
Linear Regression	All Numerical	0.994828	0.000000	0.000000	0.000000	0.000000%
Random Forest	All Numerical	0.996243	0.772349	0.387787	0.516331	0.059%
PyCaret/KNN	All Numerical	0.9983	0.8052	0.9006	0.8502	NA
KNN Problem	All Numerical	0.998545	0.825840	0.910752	0.866220	0.100%
KNN	All Numerical (without cc_num)	0.994780	0.477041	0.097599	0.162045	0.056%
PyCaret/Xgboost	All Numerical (without cc_num)	0.9977	0.8395	0.6890	0.7567	NA
Xgboost_v1	All Numerical + ‘category feature one-hot encoded	0.997833	0.849780	0.705637	0.771029	0.065%
Xgboost_v2	All numerical + ‘hour’ new feature	0.996980	0.836855	0.516701	0.638916	0.052%
Xgboost_v3	All numerical + ‘category’ (ohe) + ‘hour’ feature	0.998351	0.908068	0.757829	0.826174	0.040%
Xgboost_v4	Only(‘amt’, ‘category’, hour’)	0.997941	0.857852	0.721294	0.783669	0.062%
Xgboost_v6	Over sampling (1:2 ratio of pos/neg target)	0.991543	0.374717	0.949896	0.537428	0.824%

Summary of Findings - EDA

- From Figures (8, 9), we can see that fraud transaction occur mostly off-hours, between 3pm - 3am, but the majority of them occur between 10pm - 3am daily. This is when card owners are likely sleeping and therefore will not react quickly to the charges giving the criminals more time for making extra charges on the stolen cards.
- From Figures (2,3,4), we can see that fraud transactions occur in specific amount bands, such as: $$1-20, $$200-400, and $$700-1200, and the majority of farud transactions are in the $$1-20. Figure 3 shows that the majority of charge amount is actually between $$0-2, which can be interpreted as criminals testing the stolen cards first with small amount to make sure the credit card is valid and active. This is an interesting behaviour that the model should be able to learn from to predict fraudulent transactions. Also the all fraud transaction amount did not exceed about 1500, which is about the check amount for pensioners, so as not to run over their monthly income (Very considerate criminals :))
- From Figure 11, we can see a very interesting behavior especially for senior citizens that there are only fraud transactions against their credit cards!! This could be a scam where scammers through phishing can get credit card companies to issue cards for senior citizens without their knowledge (mail credit card scams, or phishing for their personal data and using it to request a reissue of their credit cards). This may also mean that there are fraud credit card charges for some people that have already passed away! (birth year was around 1925).
- Figure 5 shows that criminals do not discriminate against age of victims, they will hit anyone they can, but frequency of the age of the victims are mostly centered around middle age people, which makes sense as they are probably spending more and therefore using their credit cards more than the other age groups.
- Figure 7 shows that the majority of the categories that criminals bought using the stolen credit cards are either "grocery_pos" and "shopping_net'. So criminals are using these stolen credit cards to buy their groceries in with Point of Sale machines/cashiers (in-person shopping). I found this behavior surprising as I thought it will be mostly internet shopping (e-commerce, but this is the next most common category.
- From Figure (12,13) there does not seem to be a high correlation between features and the target.
- Before converting 'trans_date_trans_time' to datetime object, plotting using this feature took a lot of time and memory resources!
- I started plotting using Plotly which generates interactive charts. The charts generated using plotly once generated consumed heavy memory resources which eventually caused may laptop to freeze. I then switched to matplotlib/Seaborn.

Summary of Findings – Predictive Modeling

- The Model accuracy is too high in the case of Linear Regression -> Too good to be true!
- Linear Regression accuracy is high because the test dataset is imbalanced, so even if we guess that all transactions are legitimate (not fraud), then we will still get high accuracy (99.5%)
- So I need other metrics of model performance, mainly we will use the Precision/Recall/F1 Score in this case due to the imbalanced dataset, and I also use my own metric LTPFR, which I have defined above.
- With that, I find out the Linear Regression Precision/Recall/F1 are all zero, mostly because the True Positive that has been predicted by LR is zero, which means that LR was not able to guess any of the fraud cases...this is a very Bad model.
- Next, we try Random Forests, and as we can see it has a relatively good Precision/Recall and F1 balance.
- I next set PyCaret loose on my data, and PyCaret predicts that KNN was the best model with best Precision/Recall. Something is wrong here and I will call this the KNN problem!
- to investigate the KNN problem, I use Scikit-learn to invetigate further
- It turns out that KNN when using cc_num, it overfits on this numerical feature, and can then generates very high performance metrics but they are bogus and when removing the cc_num features, KNN as expected under performs.
- I use PyCaret again without cc_num, and now xgboost becomes the best model.
- I now focus on xgboost with feature engineering.
- I one-hot code the 'catgory' feature as it has high feature importance
- I create a new feature, 'hour' as it was seen from EDA that it can be a good feature
- the best model is xgboost with most numerical features (except cc_num) plous 'category' plus 'hour'
- I tried other things like using a minimal set of features, oversampling (2:1), and using class_weights but this did not improve the performance, on the contrary it was worse.
- in the future, I can use additional feature engineering, such as creating new features that detect when fraudsters first charge minima amount to cc, so they make sure it is a valid one, and then later buy goods.
- I can also a Neural Network to see if it can deliver better performance.



Q&A

Pycaret Best Models

	Model	Accuracy	AUC	Recall	Prec.	F1	Kappa	MCC	TT (Sec)
knn	K Neighbors Classifier	0.9983	0.9970	0.9006	0.8052	0.8502	0.8493	0.8507	2.7320
et	Extra Trees Classifier	0.9981	0.9908	0.6736	0.9513	0.7885	0.7876	0.7996	62.2220
xgboost	Extreme Gradient Boosting	0.9962	0.9816	0.3779	0.7768	0.5081	0.5064	0.5401	24.8520
rf	Random Forest Classifier	0.9961	0.9534	0.3870	0.7299	0.5056	0.5038	0.5297	588.8780
gbc	Gradient Boosting Classifier	0.9952	0.9562	0.2615	0.5849	0.3613	0.3592	0.3890	1620.9980
ada	Ada Boost Classifier	0.9949	0.9612	0.3253	0.5160	0.3990	0.3966	0.4073	656.8030
lr	Logistic Regression	0.9948	0.5000	0.0000	0.0000	0.0000	0.0000	0.0000	0.8990
nb	Naive Bayes	0.9948	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.2440
svm	SVM - Linear Kernel	0.9948	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	20.5660
qda	Quadratic Discriminant Analysis	0.9948	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	4.1200
lightgbm	Light Gradient Boosting Machine	0.9945	0.9640	0.1064	0.4095	0.1670	0.1652	0.2050	3.3400
dt	Decision Tree Classifier	0.9934	0.7011	0.4057	0.3738	0.3890	0.3857	0.3861	1.6210
lda	Linear Discriminant Analysis	0.9907	0.8332	0.4759	0.2755	0.3489	0.3446	0.3578	0.8040

```
CPU times: user 10.3 s, sys: 12.4 s, total: 22.8 s
Wall time: 8h 18min 21s
```

Pycaret Best Models

	Model	Accuracy	AUC	Recall	Prec.	F1	Kappa	MCC	TT (Sec)
xgboost	Extreme Gradient Boosting	0.9977	0.9968	0.6890	0.8395	0.7567	0.7556	0.7594	199.0480
et	Extra Trees Classifier	0.9976	0.9636	0.6642	0.8354	0.7399	0.7387	0.7437	755.2370
rf	Random Forest Classifier	0.9974	0.9667	0.6552	0.8080	0.7235	0.7222	0.7263	216.4600
gbc	Gradient Boosting Classifier	0.9969	0.9249	0.5751	0.7816	0.6616	0.6601	0.6685	141.4840
dt	Decision Tree Classifier	0.9961	0.8187	0.6394	0.6215	0.6302	0.6283	0.6284	1.6780
lr	Logistic Regression	0.9948	0.5519	0.0000	0.0000	0.0000	0.0000	0.0000	1.9620
svm	SVM - Linear Kernel	0.9948	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	192.8580
knn	K Neighbors Classifier	0.9947	0.7106	0.0820	0.4593	0.1389	0.1376	0.1922	4.6210
ridge	Ridge Classifier	0.9947	0.0000	0.0000	0.0000	0.0000	-0.0002	-0.0007	0.6510
ada	Ada Boost Classifier	0.9947	0.9733	0.1920	0.4705	0.2681	0.2660	0.2953	10.8660
nb	Naive Bayes	0.9945	0.8019	0.0000	0.0000	0.0000	-0.0006	-0.0013	0.4210
lightgbm	Light Gradient Boosting Machine	0.9942	0.8132	0.4723	0.5067	0.4787	0.4760	0.4812	1.9470
lda	Linear Discriminant Analysis	0.9907	0.8198	0.4719	0.2734	0.3462	0.3418	0.3548	69.1520
qda	Quadratic Discriminant Analysis	0.6499	0.6514	0.6119	0.1195	0.1169	0.1093	0.1362	1.7700

```
CPU times: user 4min 52s, sys: 23.1 s, total: 5min 15s
Wall time: 4h 27min 7s
```