ANDREW ROSE 12.29.2020

Fraud Detection

How to Detect Fraudulent Transactions:
"Save money and provide better service to customers"

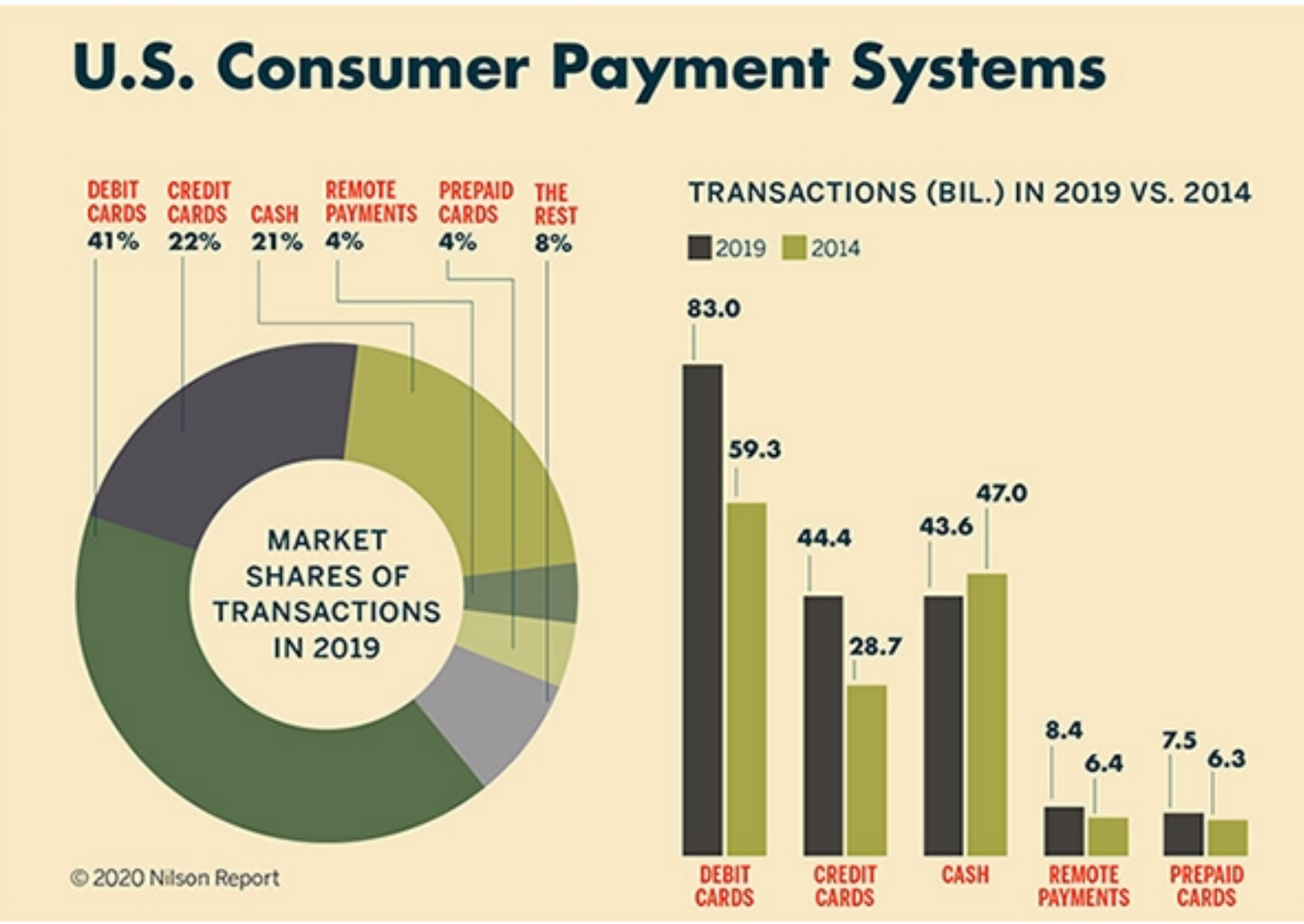
DETECTING FRAUD :

Content

Introduction
Data Set EDA
Cleaning Pre-processing
Initial Modeling
Tuning/ Optimization
Conclusion & Recommendations

GLOBAL FRAUD IS INCREASING RAPIDLY!

“THE WORLDWIDE FRAUD LOSS BY 2023 IS EXPECTED TO REACH $33 BILLION AND IN 2027 IT IS PREDICTED TO REACH $38.5 BILLION” (NILSON REPORT, ISSUE-1146, PG.8)



Card Payments Up

Cash Payments Down

Fraud steadily increasing

Introduction

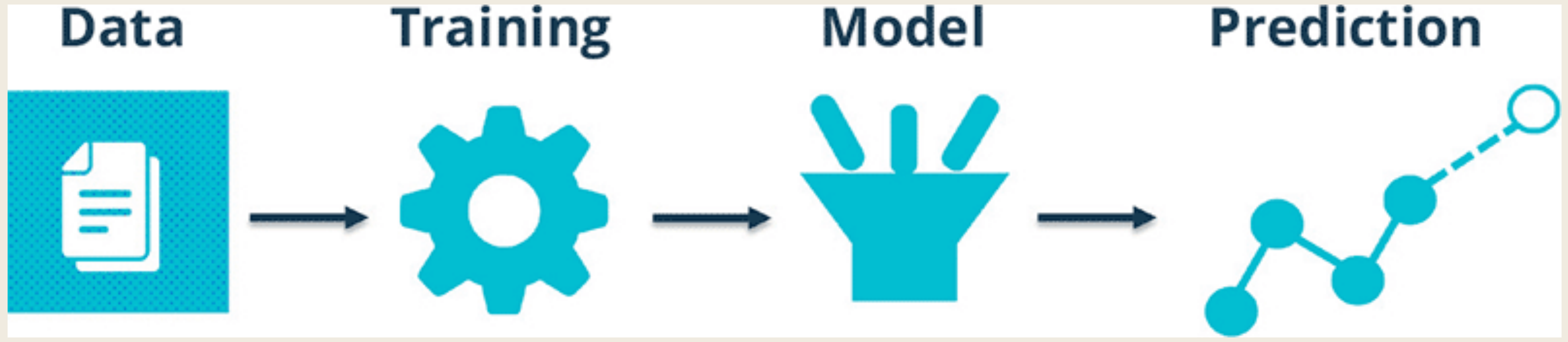
PROBLEM:

FRAUD TRANSACTIONS INCREASING IN SHEER NUMBER AND SOPHISTICATION.

CREDIT CARD FRAUD HAS BEEN A GROWING PROBLEM IN THE DIGITAL AGE. IN 2020, COVID-19 ACCELERATED THIS TREND AND WE ARE SEEING RECORD NUMBERS OF CARD TRANSACTIONS. ALONG WITH THESE TRENDS, FRAUDULENT TRANSACTIONS HAVE BEEN INCREASING AS WELL. THE COST OF FRAUD INCLUDES THE GROSS AMOUNT BUT ALSO THE ASSOCIATED SERVICING WHICH TAKES TIME AND RESOURCES. HAVING AN EFFICIENT MODEL TO HELP DETERMINE FRAUD CAN GREATLY IMPROVE EFFICIENCY AND CUSTOMER SATISFACTION.

SOLUTION:

DEPLOY CONTEMPORARY MACHINE LEARNING MODEL USING PREVIOUS TRANSACTION DATA TO HELP US PREDICT FRAUD.



Data Set EDA

fraud cases : 12417
valid cases : 773946
fraud case % : 0.016043754990658264

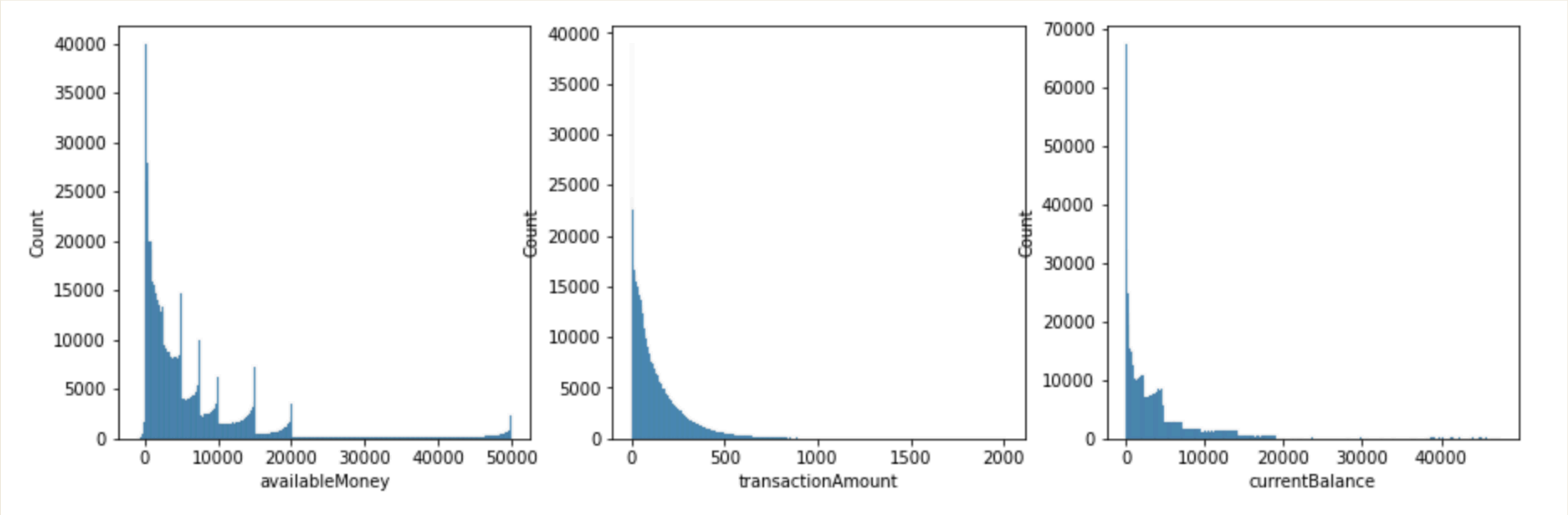
We have a very imbalanced dataset. Only a small amount of fraud cases we can use to train on.
This is a common scenario in the world of fraud detection, where most transactions are legitimate.

We can see the total amount of Fraud transactions is about $2.8 million on 12,417 cases.
The average legitimate transaction amounts are $135.57.
While the average fraud transaction amounts are $225.21.

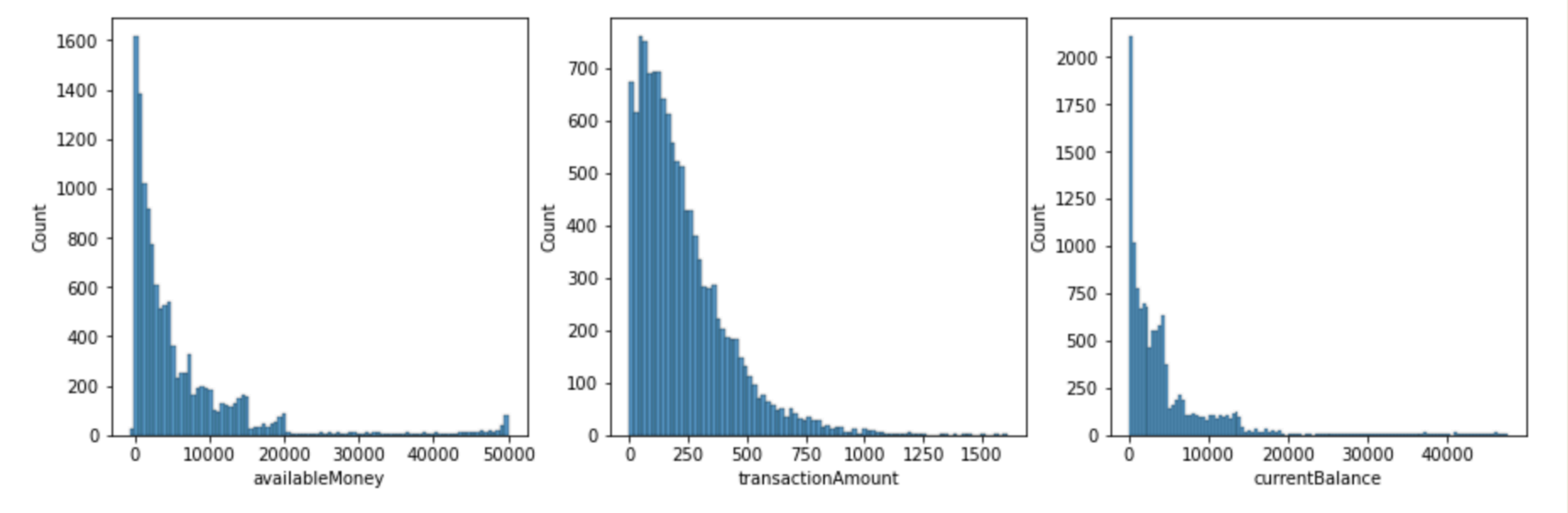
Explore the Numeric and Categorical variables.
We can see that we have 5000 unique customerIds and they have 10 different creditlimit categories. The most common merchant is Uber.

	customerId	merchantName	acqCountry	merchantCountryCode	posEntryMode	cardPresent	expirationDateKeyInMatch	isFraud	creditLimit	posConditionCode	transactionType
count	786363	786363	786363	786363	786363	786363	786363	786363	786363	786363	786363
unique	5000	2490	5	5	6	2	2	2	10	4	4
top	380680241	Uber	US	US	05	False	False	False	5000	01	PURCHASE
freq	32850	25613	774709	778511	315035	433495	785320	773946	201863	628787	745193

We will begin exploring the numeric features of the dataset.
We don't have too many columns to work with but lets take a took.



We can see extreme skew in the data visually indicating the imbalanced nature of the transactions. Fraud vs Non Fraud transactions show a similar level of skewness.



We can see the mean availableMoney and currentBalance columns for Fraud vs no-Fraud are very similar. The largest difference is the transaction amount.
135 no-Fraud vs 225 Fraud

The mean/std transaction size is larger for fraud vs no fraud…

All Transactions:

	availableMoney	transactionAmount	currentBalance
count	12417.000000	12417.000000	12417.000000
mean	6142.894186	225.215905	4902.064338
std	8703.131117	189.551393	7074.701649
min	-614.390000	0.000000	0.000000
25%	1078.020000	86.000000	822.210000
50%	3120.950000	176.980000	2747.390000
75%	7502.820000	311.460000	5644.350000
max	50000.000000	1608.350000	47473.940000

Fraud transactions:

	availableMoney	transactionAmount	currentBalance
count	773946.000000	773946.000000	773946.000000
mean	6252.455386	135.570249	4502.428675
std	8883.600096	146.525305	6446.866656
min	-1005.630000	0.000000	0.000000
25%	1077.420000	33.190000	688.032500
50%	3186.145000	86.760000	2446.940000
75%	7500.000000	189.390000	5286.100000
max	50000.000000	2011.540000	47498.810000

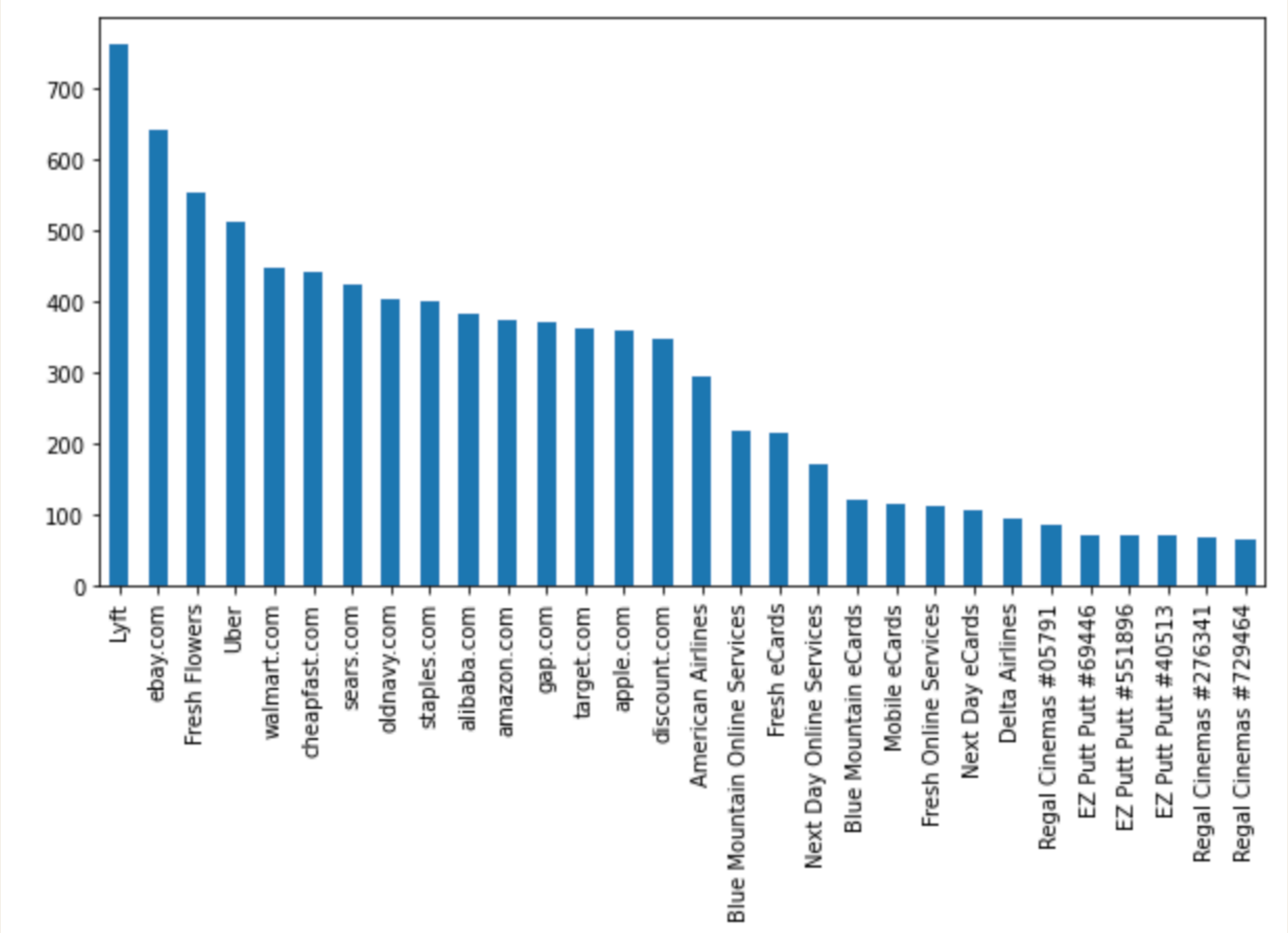
What are the spending habits / patterns of the transactions?

What features stand out between fraud and no-fraud transactions.

Compare Distributions on merchants between fraud/no-fraud.

Fraud Transactions

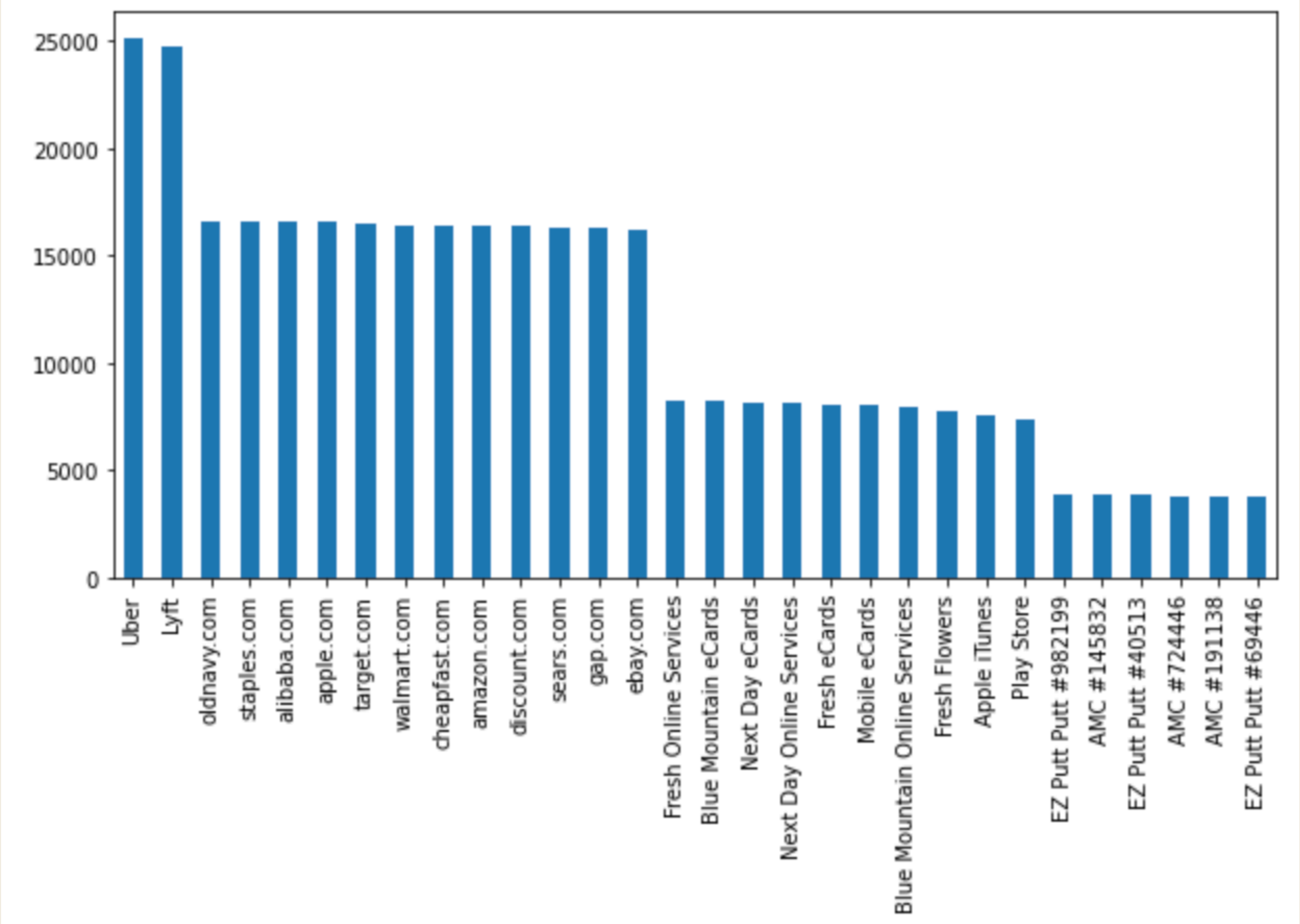
Lyft	760
ebay.com	639
Fresh Flowers	553
Uber	512
walmart.com	446



<u>**Most popular merchants by # of transactions**</u>

All Transactions

Uber	25101
Lyft	24763
oldnavy.com	16591
staples.com	16581
alibaba.com	16576

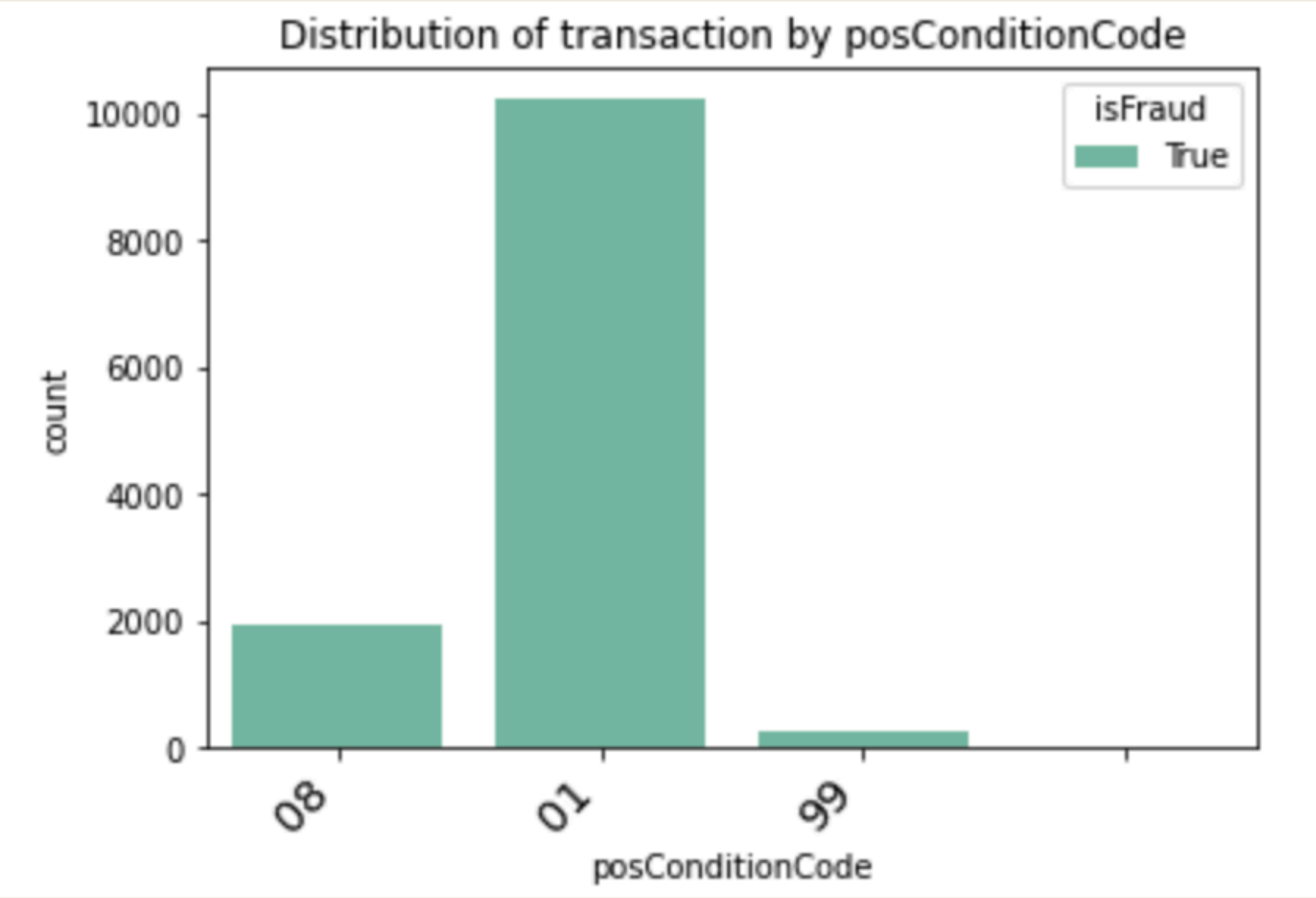


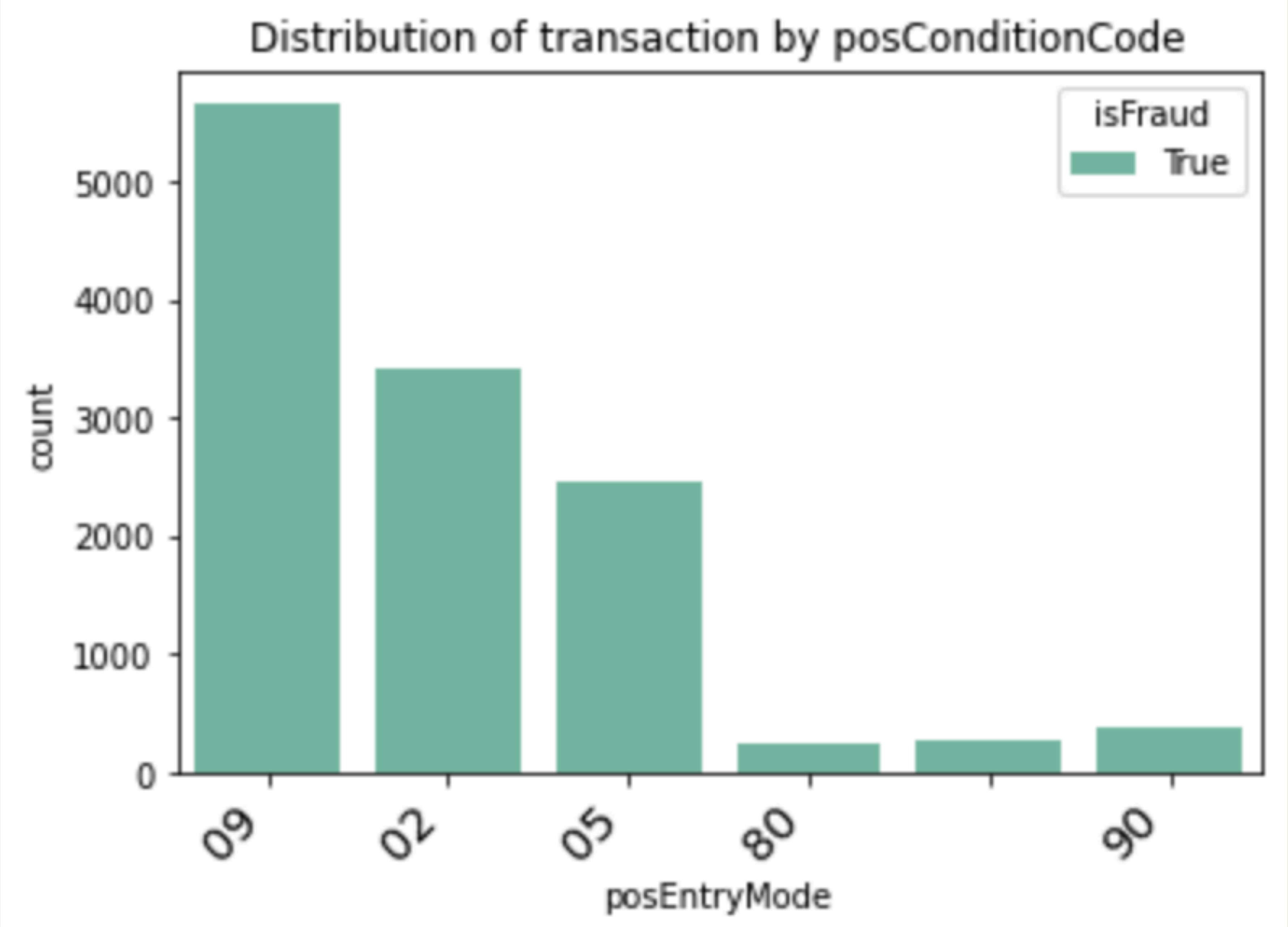
The Fraud vs no-Fraud vendor count is similar, however one vendor Fresh Flowers seems to stand out as it is #3 in fraud but #22 in overall transactions by count.

Fraud transactions
False 0.721752
True 0.278248
Name: cardPresent

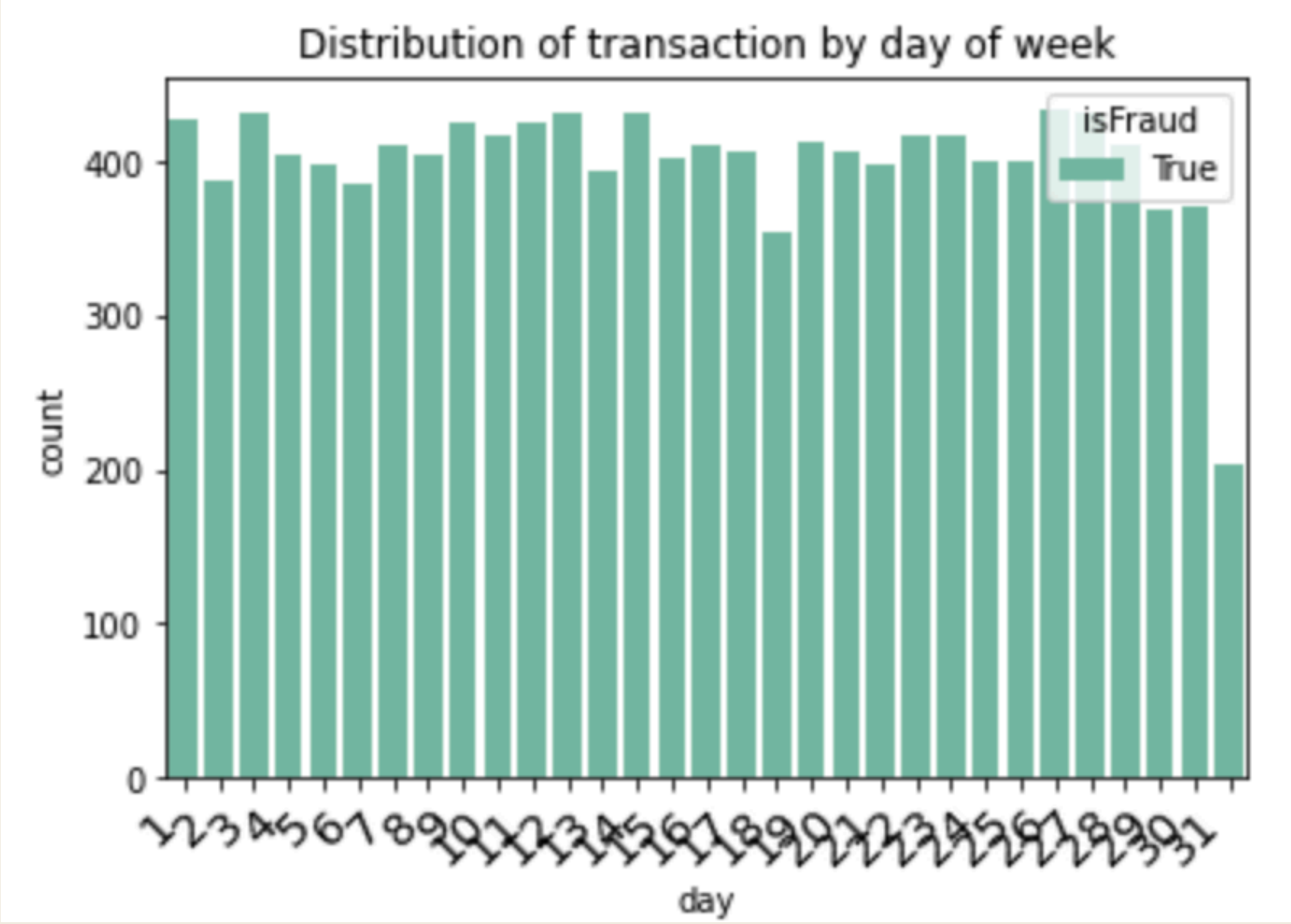
All Transactions
False 0.551266
True 0.448734
Name: cardPresent

(cardPresent = False) during Fraud transactions was 72% of time vs only 55% for all transactions.

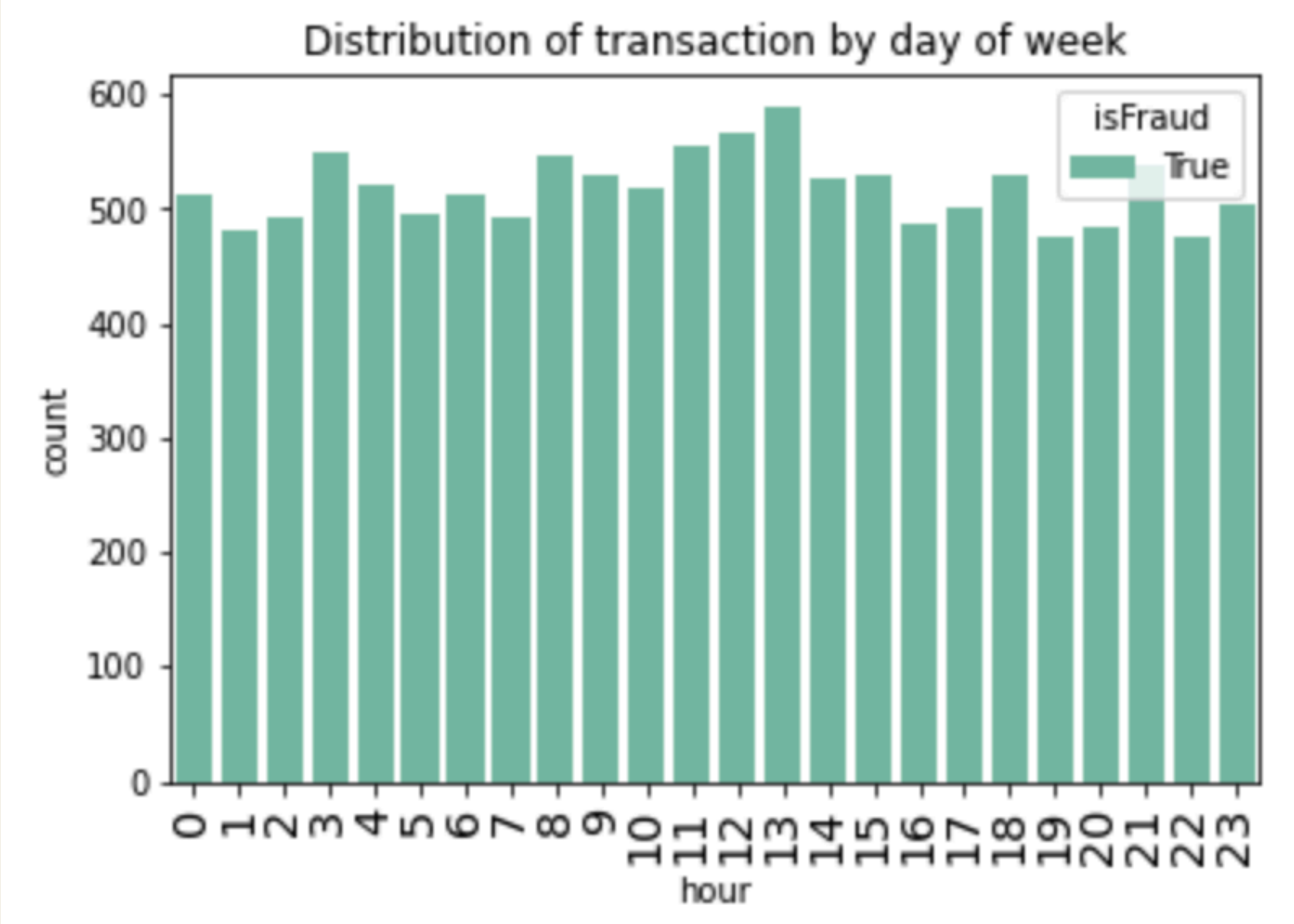




Vast majority of fraud transactions go through POS code 01 and POS entry Mode 09

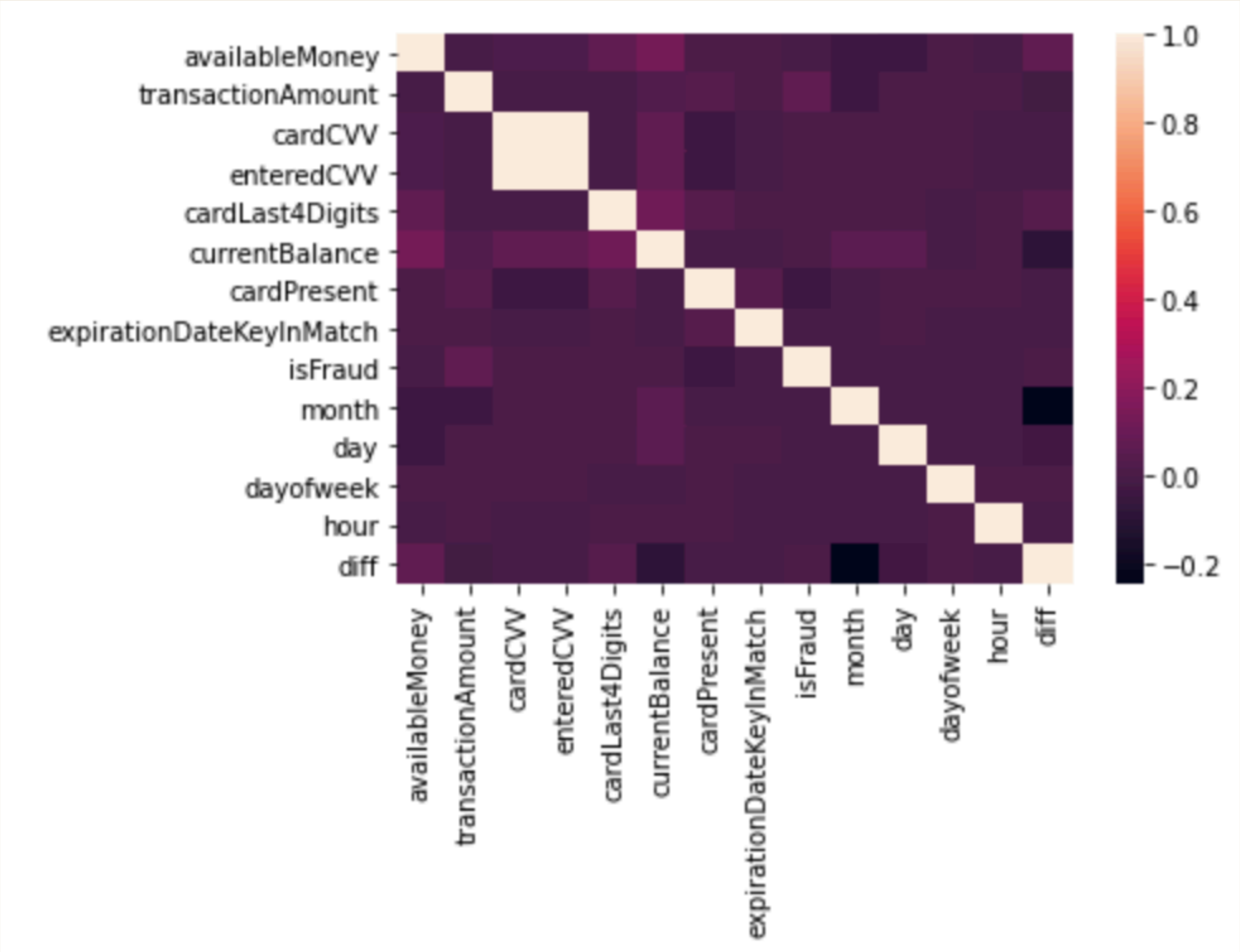


A very even amount of fraud transactions per day of the week save for the last of the month.



Count of Fraud transactions by hour of the day doesn't seem to show any big spikes. a slight increase around the middle of the day

Run a heatmap to see if any features are correlated



Not too much information here but we see slight correlations to fraud in the currentBalance and transactionAmount columns

Modeling - Comparisons

First we clean up the data (select relevant columns). We have little null data and everything else is relatively clean. We include a few manual datetime and ratio features.

Modern techniques include using models like XGBoost which have been shown to outperform almost every other model in regards to binary classification.

https://ieeexplore.ieee.org/abstract/document/9214206

https://www.e3s-conferences.org/articles/e3sconf/abs/2020/74/e3sconf_ebldm2020_02042/e3sconf_ebldm2020_02042.html

We do a comparison of all the available classification models in PyCaret.

```
0   creditLimit               786363 non-null  category
1   availableMoney            786363 non-null  float32
2   transactionDateTime       786363 non-null  datetime64[
3   transactionAmount         786363 non-null  float32
4   merchantName              786363 non-null  category
5   posEntryMode              786363 non-null  category
6   posConditionCode          786363 non-null  category
7   merchantCategoryCode      786363 non-null  category
8   currentExpDate            786363 non-null  category
9   accountOpenDate           786363 non-null  datetime64[
10  dateOfLastAddressChange   786363 non-null  datetime64[
11  transactionType           786363 non-null  category
12  currentBalance            786363 non-null  float32
13  cardPresent               786363 non-null  bool
14  expirationDateKeyInMatch  786363 non-null  bool
15  isFraud                   786363 non-null  bool
16  month                     786363 non-null  category
17  day                       786363 non-null  category
18  hour                      786363 non-null  category
19  CVVMatch                  786363 non-null  bool
20  accountDiff               786363 non-null  float32
21  acqCountry_merchant_match 786363 non-null  bool
22  creditRatio               786363 non-null  float32
23  transactionRatio          786363 non-null  float32
24  balanceRatio              786363 non-null  float32
25  balanceCreditRatio        786363 non-null  float32
dtypes: bool(5), category(10), datetime64[ns](3), float32(8)
memory usage: 54.9 MB
```

PyCaret setup parameters

```
clf = setup(data=df,
            target='isFraud',
            use_gpu=True,
            fold=5,
            remove_multicollinearity=True,
            multicollinearity_threshold=.95,
            combine_rare_levels=True,
            ignore_low_variance=True,
            feature_ratio=True,
            n_jobs=7)
```

0	session_id	6173	29	Normalize	False
1	Target	isFraud	30	Normalize Method	None
2	Target Type	Binary	31	Transformation	False
3	Label Encoded	False: 0, True: 1	32	Transformation Method	None
4	Original Data	(786363, 26)	33	PCA	False
5	Missing Values	False	34	PCA Method	None
6	Numeric Features	8	35	PCA Components	None
7	Categorical Features	14	36	Ignore Low Variance	True
8	Ordinal Features	False	37	Combine Rare Levels	True
9	High Cardinality Features	False	38	Rare Level Threshold	0.100000
10	High Cardinality Method	None	39	Numeric Binning	False
11	Transformed Train Set	(550454, 707)	40	Remove Outliers	False
12	Transformed Test Set	(235909, 707)	41	Outliers Threshold	None
13	Shuffle Train-Test	True	42	Remove Multicollinearity	True
14	Stratify Train-Test	False	43	Multicollinearity Threshold	0.950000
15	Fold Generator	StratifiedKFold	44	Clustering	False
16	Fold Number	5	45	Clustering Iteration	None
17	CPU Jobs	7	46	Polynomial Features	False
18	Use GPU	True	47	Polynomial Degree	None
19	Log Experiment	False	48	Trignometry Features	False
20	Experiment Name	clf-default-name	49	Polynomial Threshold	None
21	USI	5226	50	Group Features	False
22	Imputation Type	simple	51	Feature Selection	False
23	Iterative Imputation Iteration	None	52	Features Selection Threshold	None
24	Numeric Imputer	mean	53	Feature Interaction	False
25	Iterative Imputation Numeric Model	None	54	Feature Ratio	True
26	Categorical Imputer	constant	55	Interaction Threshold	0.010000
27	Iterative Imputation Categorical Model	None	56	Fix Imbalance	False
28	Unknown Categoricals Handling	least_frequent	57	Fix Imbalance Method	SMOTE

Modeling Results

We can see that the gradient boosting models overall
have the best AUC along with some of the best compute times.
We will take a closer look at the top 3 models performance and features.

```
best = compare_models(sort = 'AUC', exclude=['gbc'])
```

executed in 1h 34m 24s, finished 06:38:58 2020-12-29

	Model	Accuracy	AUC	Recall	Prec.	F1	Kappa	MCC	TT (Sec)
lightgbm	Light Gradient Boosting Machine	0.9850	0.8232	0.0577	0.7599	0.1072	0.1052	0.2066	9.3820
xgboost	Extreme Gradient Boosting	0.9851	0.8185	0.0562	0.8890	0.1056	0.1040	0.2209	23.4380
catboost	CatBoost Classifier	0.9852	0.8094	0.0559	0.9904	0.1058	0.1043	0.2332	21.4400
et	Extra Trees Classifier	0.9853	0.7759	0.0651	0.9869	0.1221	0.1204	0.2514	198.3500
rf	Random Forest Classifier	0.9847	0.7747	0.0223	1.0000	0.0436	0.0430	0.1474	106.5780
ada	Ada Boost Classifier	0.9843	0.7624	0.0001	0.1000	0.0003	0.0003	0.0037	145.0700
lda	Linear Discriminant Analysis	0.9811	0.7548	0.0380	0.1363	0.0593	0.0528	0.0642	64.4220
nb	Naive Bayes	0.6816	0.7189	0.5913	0.0356	0.0629	0.0351	0.0811	7.7880
lr	Logistic Regression	0.9843	0.7001	0.0004	0.3500	0.0009	0.0008	0.0115	69.9860
dt	Decision Tree Classifier	0.9726	0.5531	0.1200	0.1215	0.1207	0.1068	0.1069	137.7320
knn	K Neighbors Classifier	0.9843	0.5163	0.0001	0.0400	0.0003	0.0001	0.0014	214.6640
qda	Quadratic Discriminant Analysis	0.9824	0.5000	0.0020	0.0084	0.0028	0.0001	-0.0001	41.8720
svm	SVM - Linear Kernel	0.9796	0.0000	0.0103	0.0519	0.0135	0.0076	0.0117	74.0800
ridge	Ridge Classifier	0.9843	0.0000	0.0000	0.0000	0.0000	-0.0000	-0.0001	7.3160

Top3

These are the scores of the top3 models on the unseen/holdout dataset.

	Model	Accuracy	AUC	Recall	Prec.	F1	Kappa	MCC
0	Light Gradient Boosting Machine	0.9847	0.8355	0.0648	0.7105	0.1187	0.1164	0.2116

	Model	Accuracy	AUC	Recall	Prec.	F1	Kappa	MCC
0	Extreme Gradient Boosting	0.9850	0.8367	0.0658	0.8697	0.1224	0.1204	0.2369

	Model	Accuracy	AUC	Recall	Prec.	F1	Kappa	MCC
0	CatBoost Classifier	0.9850	0.8242	0.0600	0.9825	0.1130	0.1114	0.2408

Exploring Feature Importances:

What's interesting is that these models top 10 features are mostly different yet arrive at a similar evaluation score. PosEntryMode_05 is the only feature in the top 10 between all 3 models.

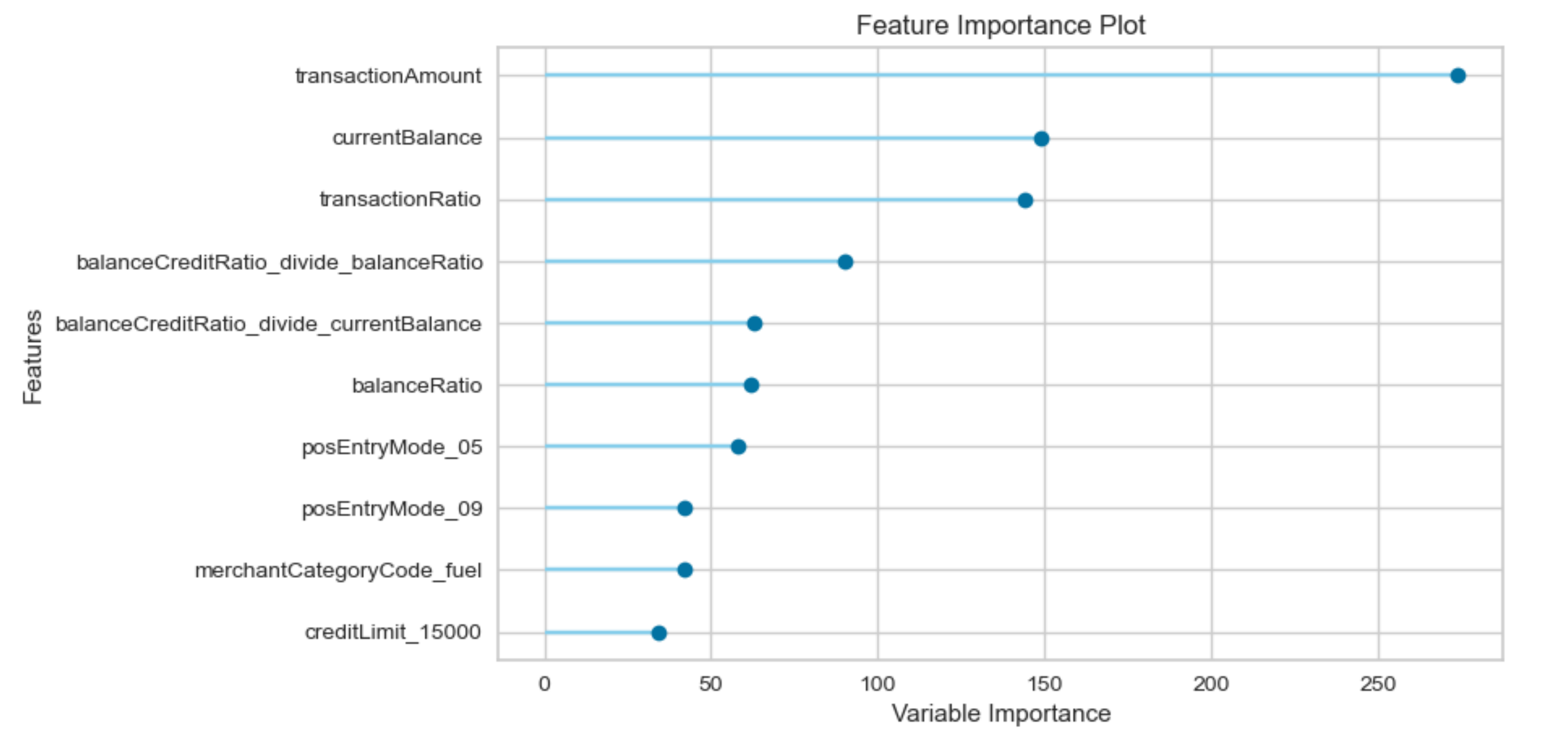
XGboost features indicate there are times of the day and certain merchants to look into more closely. We have Amazon, Apple, but also EZ-Putt which seems peculiar, would warrant further investigation. The feature merchantCategoryCode_entertainment also shows up in 2 models again warranting further scrutiny.

LGBM is also interesting in that it was the only model that utilized our manual engineered numeric ratios. We also ran feature_ratios=True in our PyCaret setup environment so we see our manual ratios put against other ratios.

Catboost and LGBM had transactionAmount as the top feature and it makes sense. During our initial EDA, we saw the average amount per fraud transaction is almost 33% above "normal" so you definitely want to watch out for the amounts. Catboost also has merchantCategoryCode_fuel as its 3rd most important feature, again signaling us to be more alert around these types of transactions.(gas cards are a well known fraud vehicle)

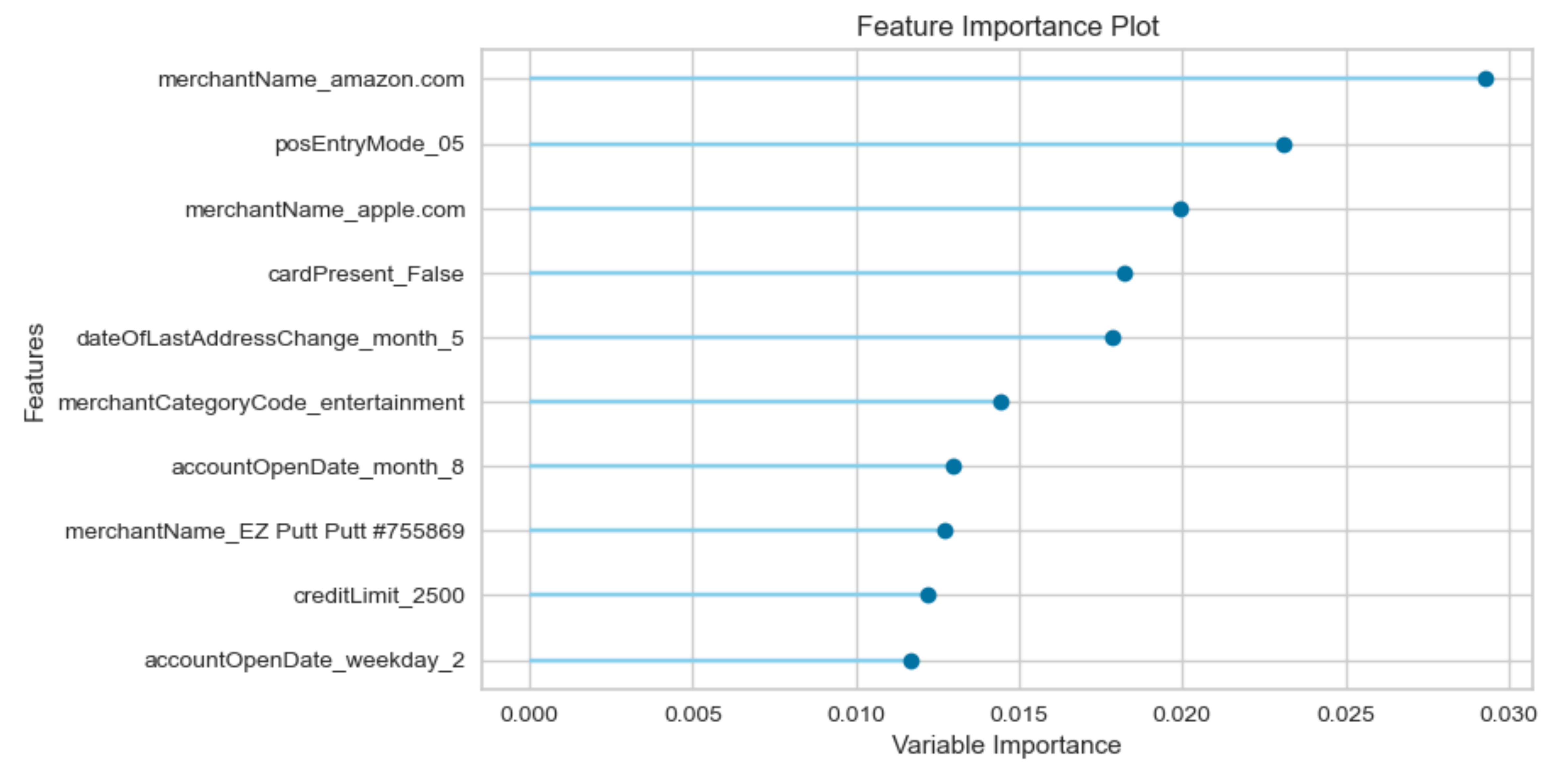
```
plot_model(lgbm, plot = 'feature')
```

executed in 2.67s, finished 11:09:19 2020-12-29



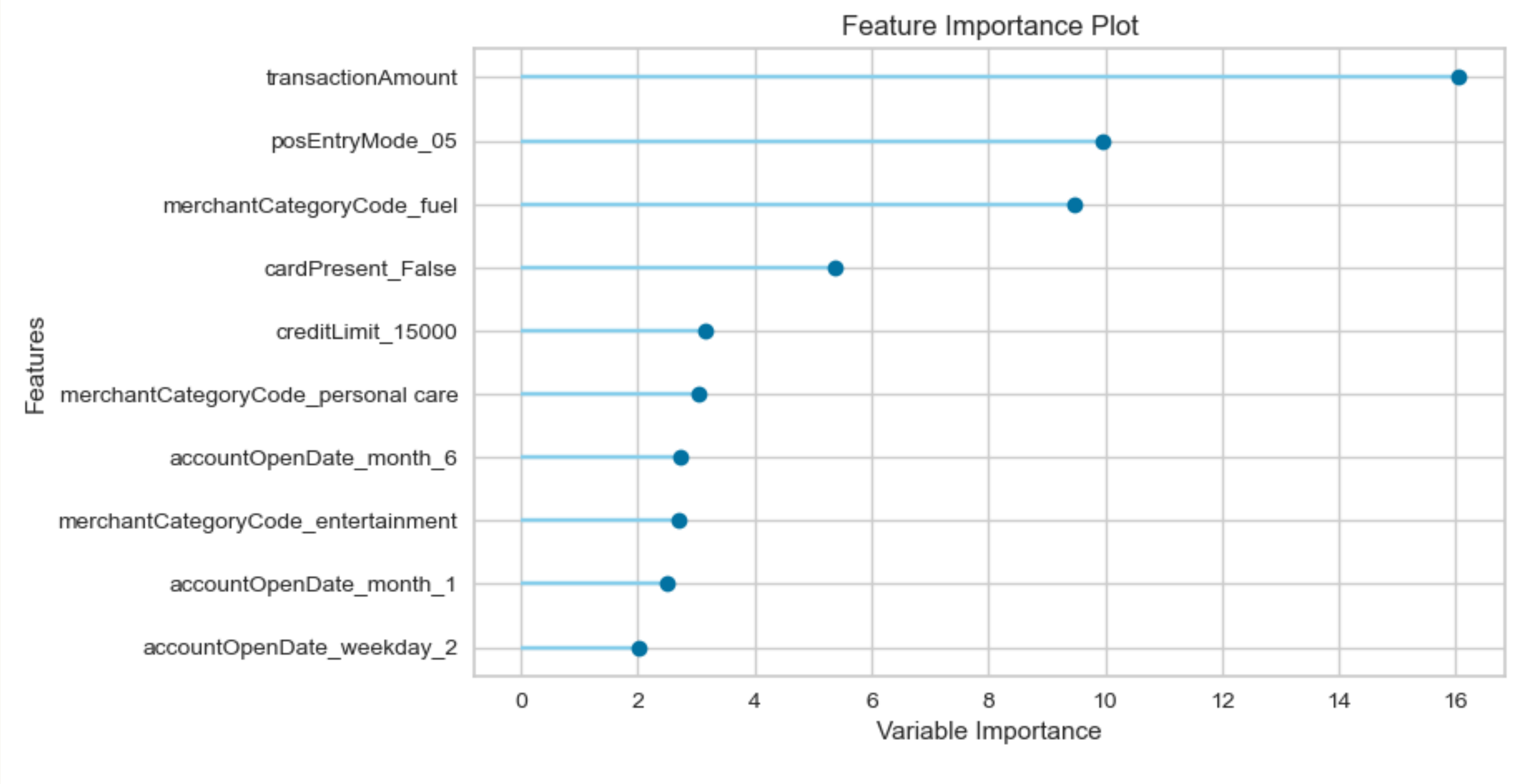

```
plot_model(xgboost, plot = 'feature')
```

executed in 2.85s, finished 11:09:15 2020-12-29




```
plot_model(catboost, plot = 'feature')
```

executed in 2.65s, finished 11:24:34 2020-12-29



Model Blending:

PyCaret has a feature that allows us to blend models to try and improve performance. So we take the top 4 models and blend them together. This actually gives us our best AUC (.84) so far.

However, the caveat with this model is we cannot view the feature importances so the results are a bit of a blackbox.

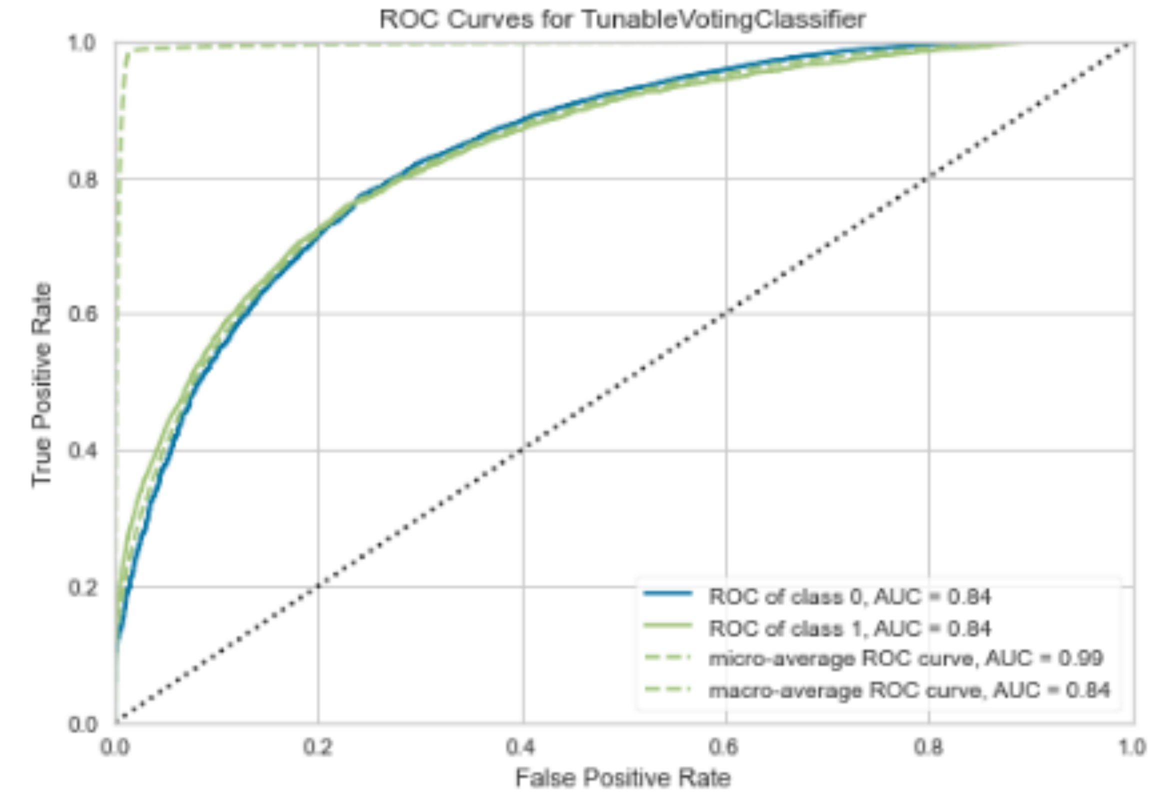
	Model	Accuracy	AUC	Recall	Prec.	F1	Kappa	MCC	TT (Sec)
catboost	CatBoost Classifier	0.9850	0.8105	0.0555	0.9838	0.1050	0.1035	0.2316	20.6560
xgboost	Extreme Gradient Boosting	0.9849	0.8231	0.0597	0.8658	0.1117	0.1099	0.2248	19.9840
lightgbm	Light Gradient Boosting Machine	0.9848	0.8254	0.0521	0.7998	0.0978	0.0960	0.2014	8.1560
rf	Random Forest Classifier	0.9848	0.7820	0.0397	0.9964	0.0764	0.0752	0.1971	96.4500

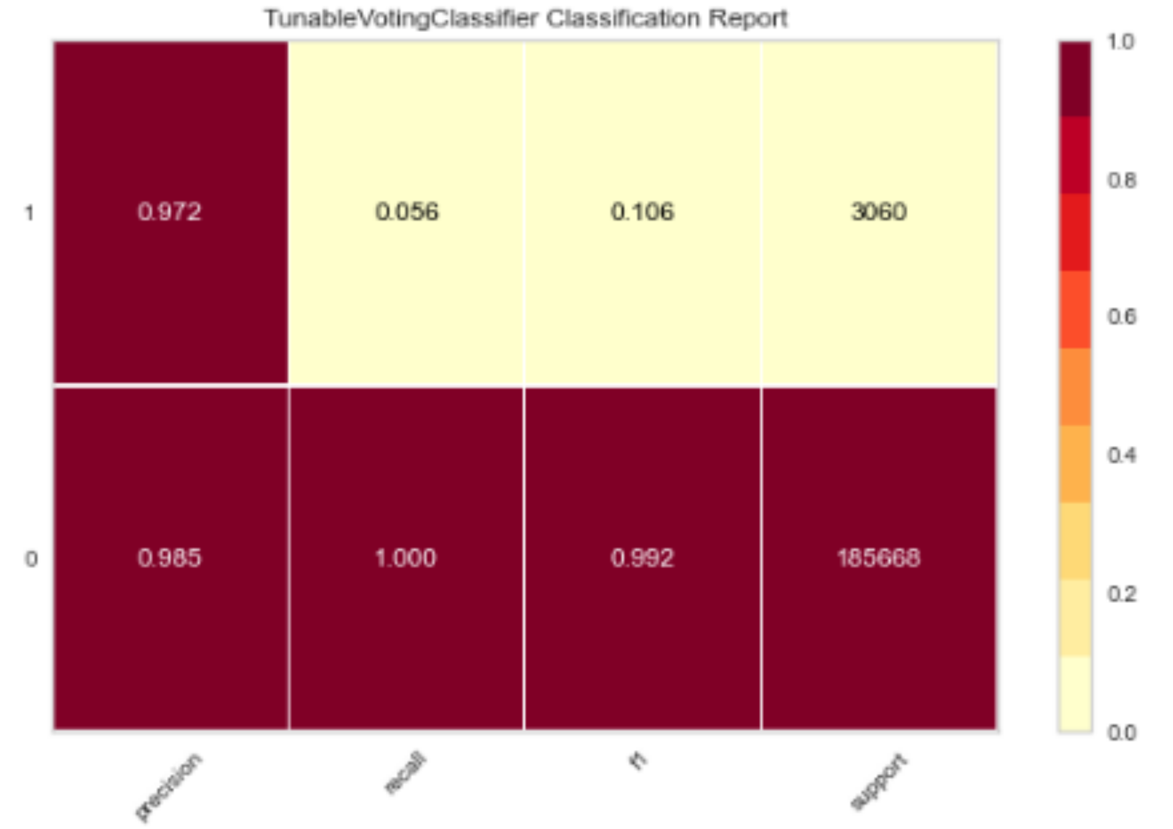
```
#blend top 4 base models
blender = blend_models(blends,choose_better = True, optimize='AUC')
```

executed in 27m 28s, finished 16:34:48 2020-12-19

	Accuracy	AUC	Recall	Prec.	F1	Kappa	MCC
0	0.9849	0.8419	0.0466	0.9683	0.0889	0.0876	0.2107
1	0.9848	0.8495	0.0390	1.0000	0.0750	0.0739	0.1959
2	0.9850	0.8377	0.0527	0.9718	0.1000	0.0985	0.2245
3	0.9850	0.8424	0.0550	0.9863	0.1042	0.1027	0.2311
4	0.9848	0.8320	0.0466	0.8841	0.0885	0.0871	0.2010
Mean	0.9849	0.8407	0.0480	0.9621	0.0913	0.0900	0.2126
SD	0.0001	0.0058	0.0056	0.0406	0.0102	0.0101	0.0134

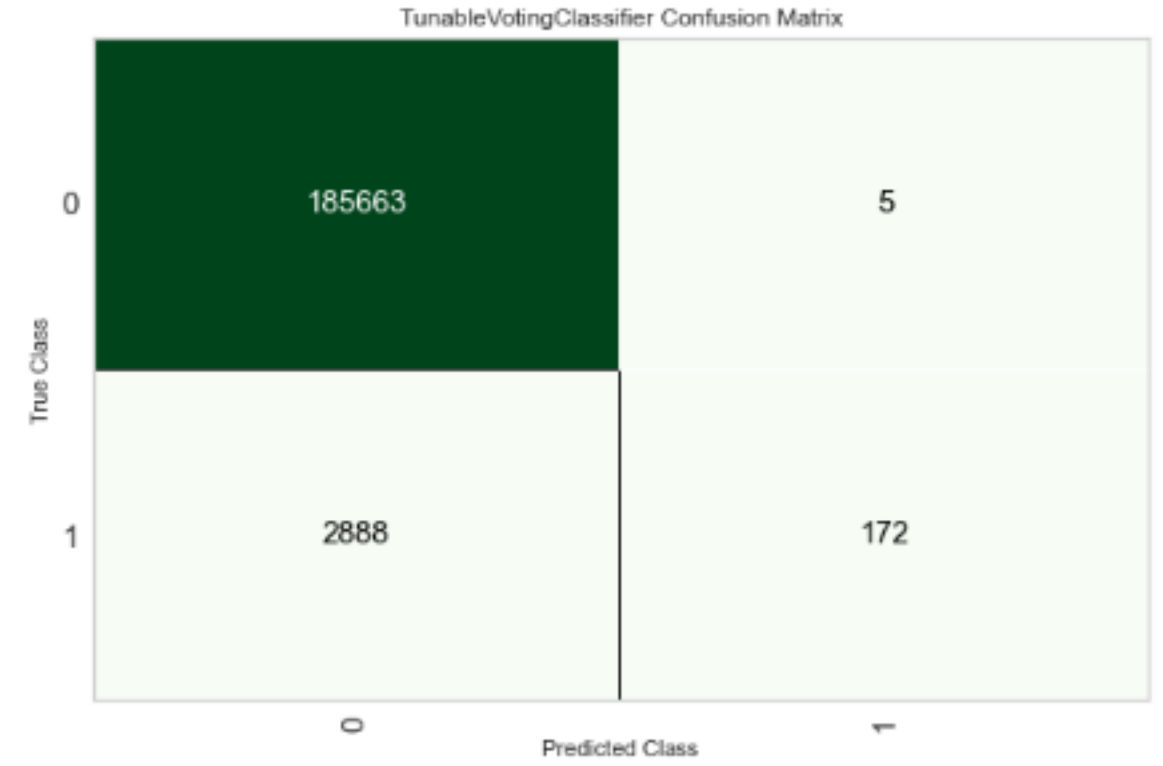
Blended Scores:






```
plot_model(blender, plot = 'confusion_matrix')
```

executed in 30.0s, finished 08:57:25 2020-12-29



In [25]:

```
pred_holdout_blender = predict_model(blender)
```

executed in 23.2s, finished 08:27:04 2020-12-29

	Model	Accuracy	AUC	Recall	Prec.	F1	Kappa	MCC
0	Voting Classifier	0.9847	0.8429	0.0562	0.9718	0.1063	0.1047	0.2318

Conclusion:

Overall the gradient boosting models have shown to be superior when it comes to binary classification of rare/infrequent events often with highly skewed distributions. We found that we can eke out a little bit of performance by blending these models together in PyCaret.

The most effective way to improve our prediction power is by analyzing the features utilized by the models to help inform our next direction of analysis or inquiry. This in turn will help us create better features and further refine the model.

It's true what they say, model optimization is more often than not a form of feature engineering. We have a model that is strong to start and ready to be deployed, while we also have a few new lines of inquiry to explore further.