Cognizant Big Bytes Challenge 2019

3 Heavenly Kings

Darren Choy
Lau Jun Rong
Ong De Lin

Main Sections

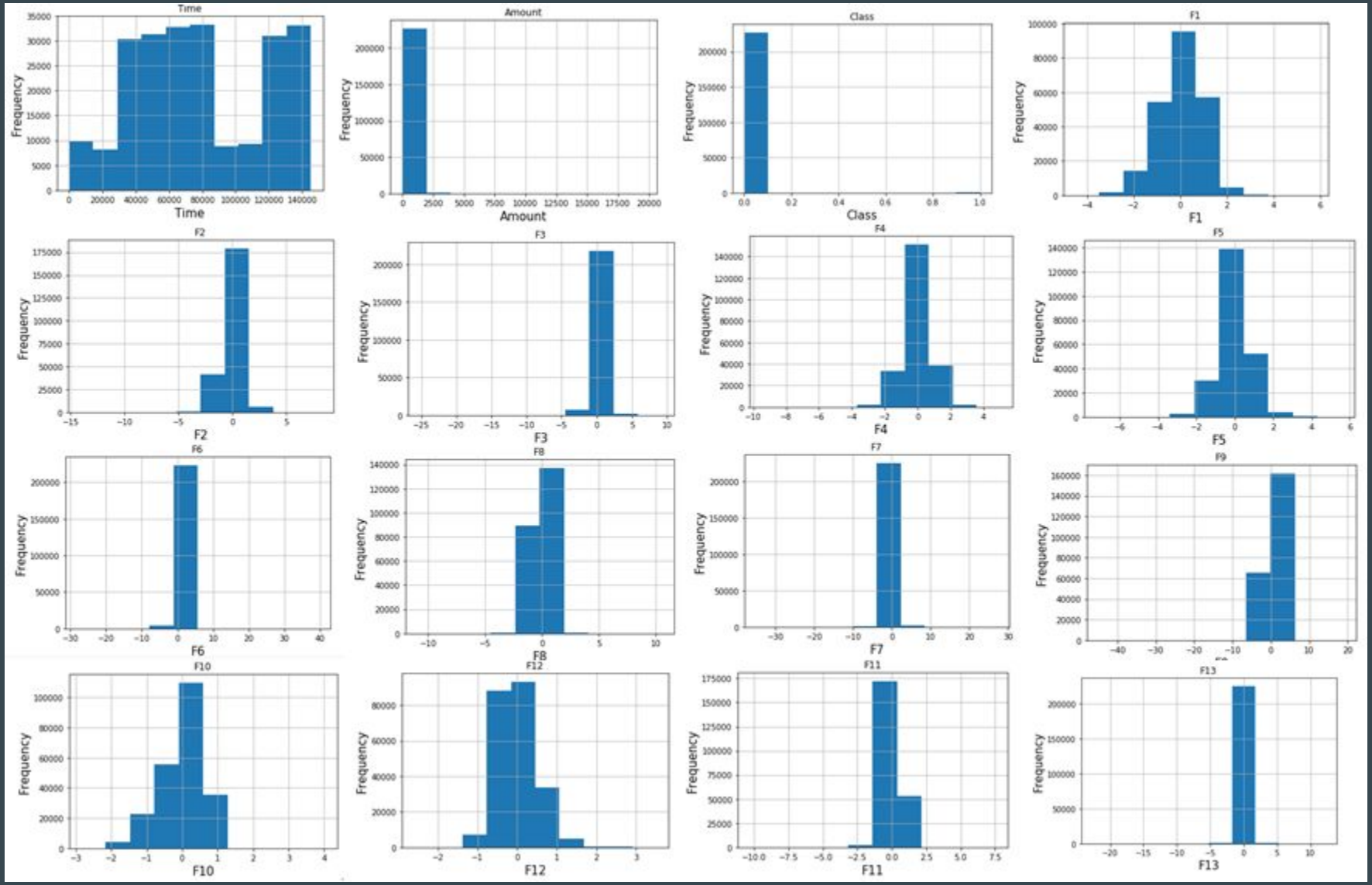
- Data exploration & preparation
- Choosing our metrics
- Choosing, training & validating the model
- Business implementation & recommendations

Exploratory Data Analysis

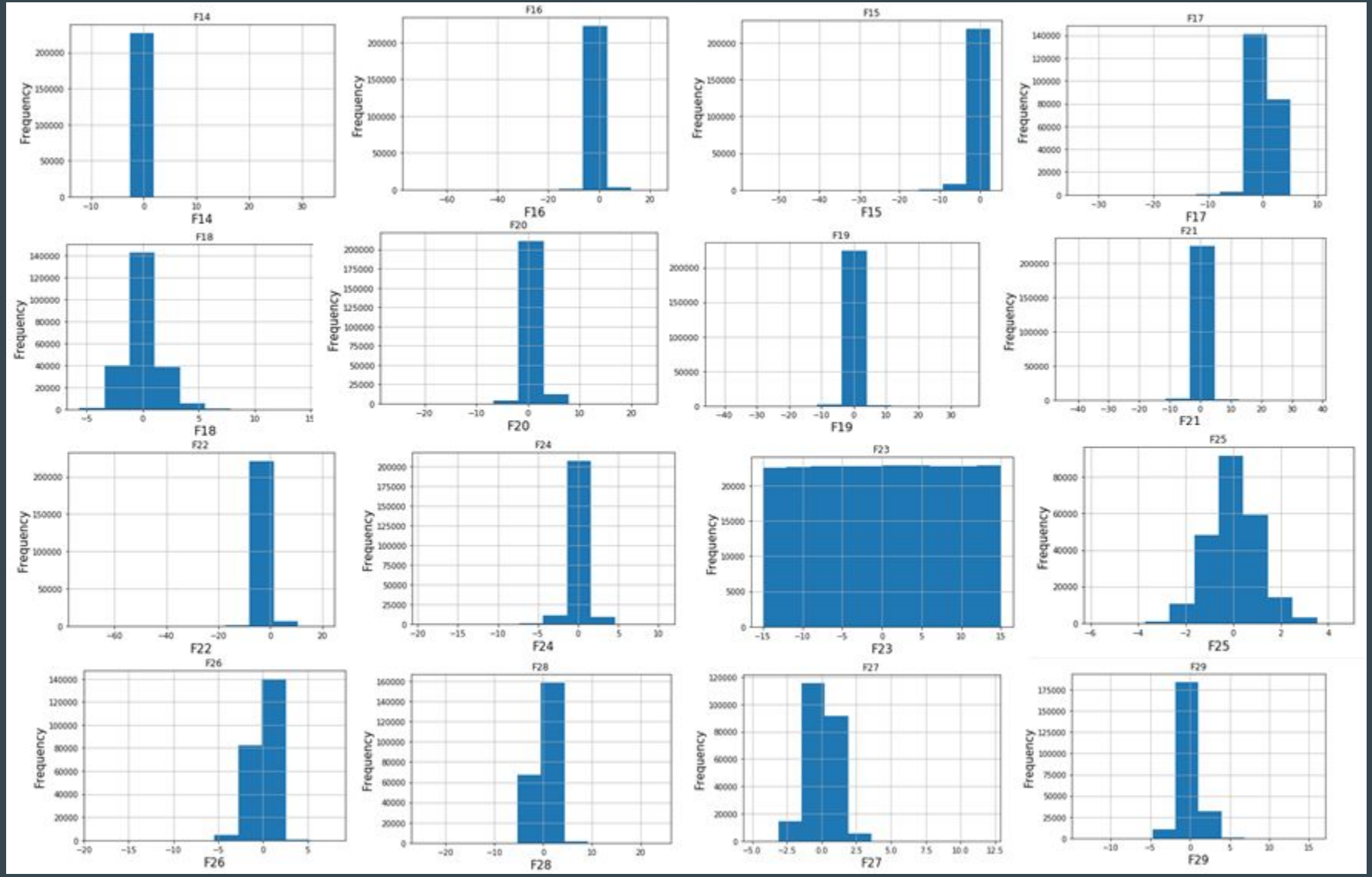
- Import dataset into pandas dataframe
- Understanding dataset using pyplot and seaborn

```
df = pd.read_csv(os.environ['DSX_PROJECT_DIR']+'/datasets/creditcard-training set v2.csv')
df = df.rename(columns={ df.columns[2]: "Fraud" })

#check the distribution
for row in df.head():
    fig=plt.figure(figsize=(17,10))
    df.hist(column=row)
    plt.xlabel(row,fontsize=15)
    plt.ylabel("Frequency",fontsize=15)
```

Exploratory Data Analysis



Exploratory Data Analysis



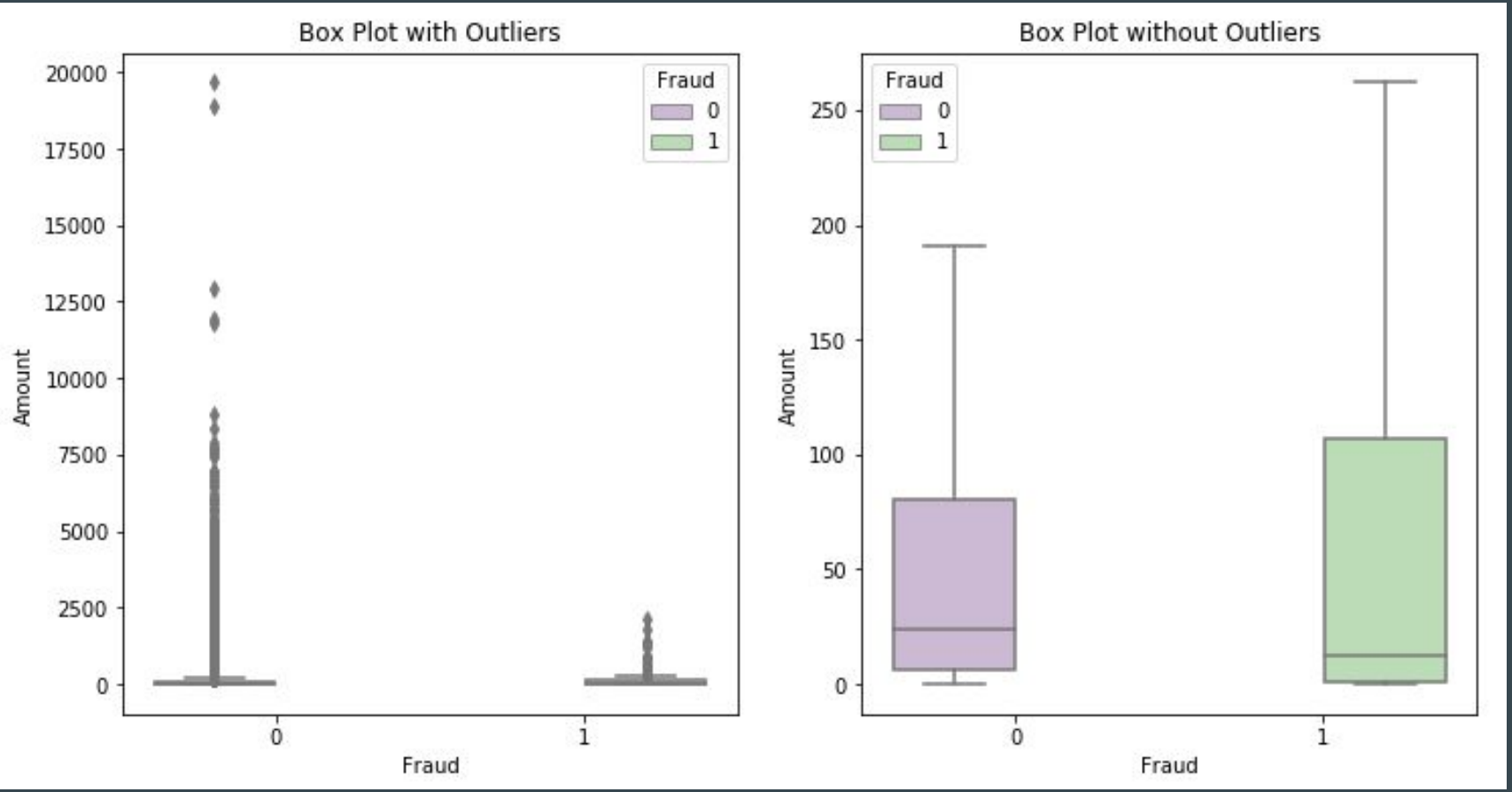
Highly Imbalanced Dataset

> A mere 0.18% of the dataset is classified as fraud, while the rest is valid. This indicates that the dataset is highly imbalanced.

```
df_class_0 = df[df['Class'] == 0]
df_class_1 = df[df['Class'] == 1]

print('No Fraud:',df_class_0.Class.count())
print('Fraud:',df_class_1.Class.count())
print('% Fraud:', str(round(df_class_1.Class.count()/df_class_0.Class.count()*100,2)) + "%")
```

```
No Fraud: 227351
Fraud: 411
% Fraud: 0.18%
```

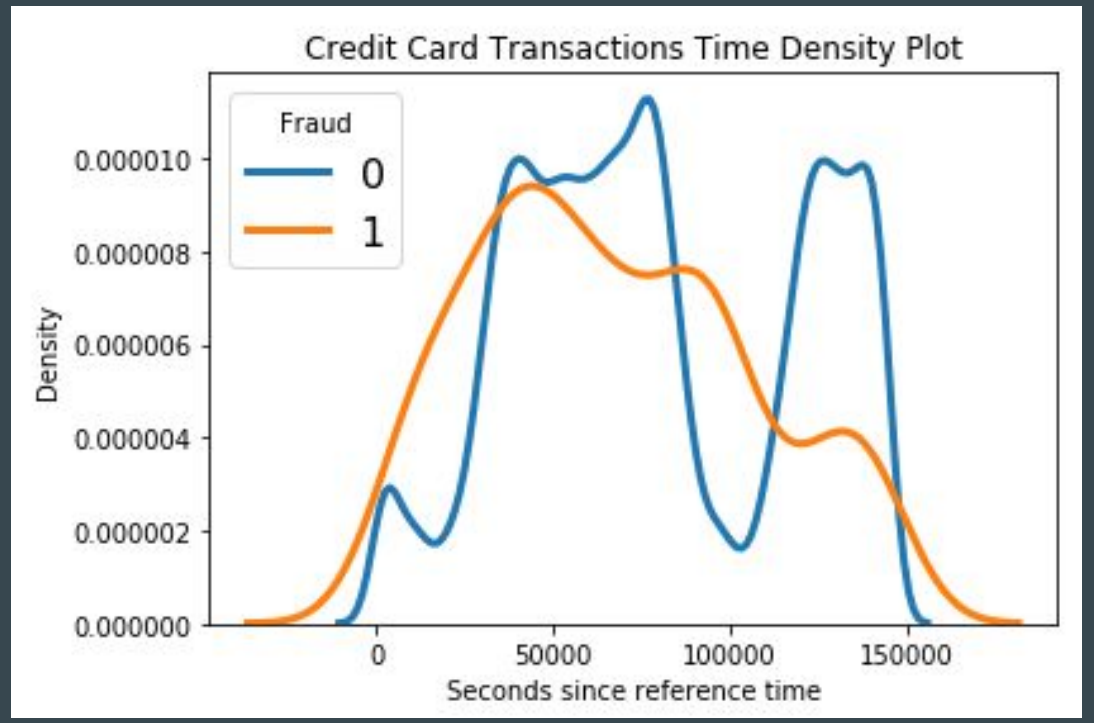
Box Plot for each Classes




```
fig, (ax1, ax2) = plt.subplots(ncols=2, figsize=(12,6))
s = sns.boxplot(ax = ax1, x="Fraud", y="Amount", hue="Fraud",data=df, palette="PRGn",showfliers=True).set_title("Box Plot with Outliers")
s = sns.boxplot(ax = ax2, x="Fraud", y="Amount", hue="Fraud",data=df, palette="PRGn",showfliers=False).set_title("Box Plot without Outliers")
plt.show();
```

Credit Card Transaction Time Density Plot

- Fraudulent transactions have a more even distribution than valid transactions as they are equally distributed in time

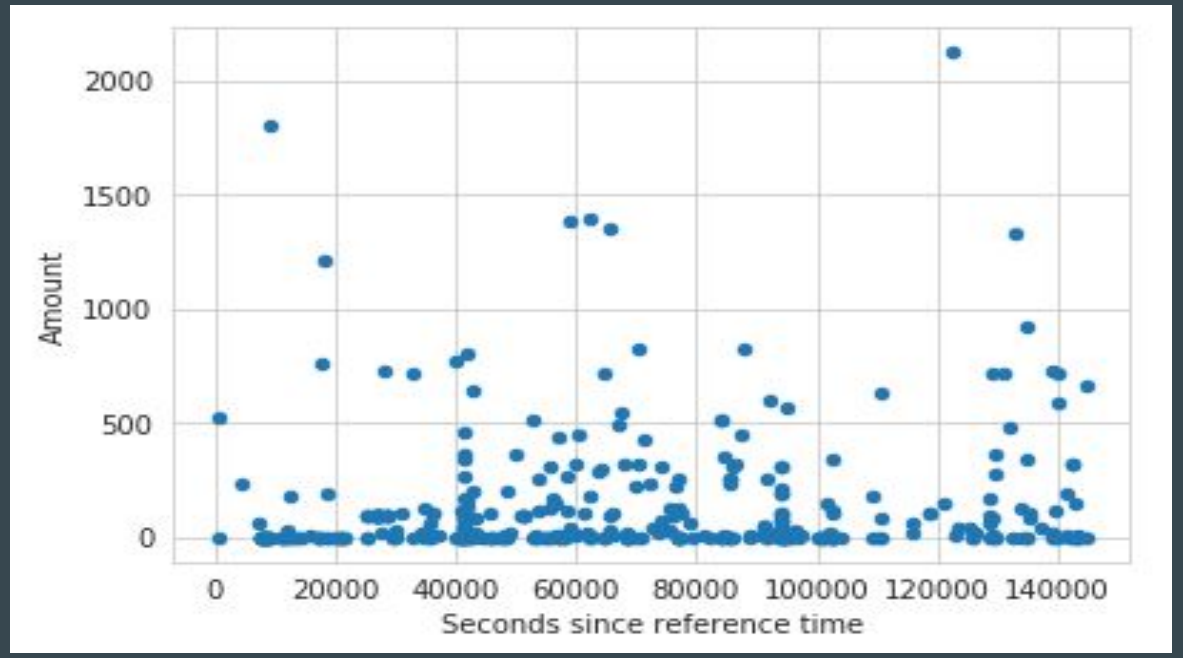



```
for i in range(2):
    subset = df[df['Fraud'] == i]

    # Draw the density plot
    sns.distplot(subset['Seconds since reference time'], hist = False, kde = True,
                 kde_kws = {'linewidth': 3},
                 label = i)

# Plot formatting
plt.legend(prop={'size': 16}, title = 'Fraud')
plt.title('Credit Card Transactions Time Density Plot')
plt.xlabel('Seconds since reference time')
plt.ylabel('Density')
```

Interesting Finding

```
fraud = df.loc[df['Fraud'] == 1]
ax2 = fraud.plot.scatter(x='Seconds since reference time', y='Amount')
print(fraud['Amount'].describe())
```

> Fraudsters are consistently spending small amount of money to prevent getting caught



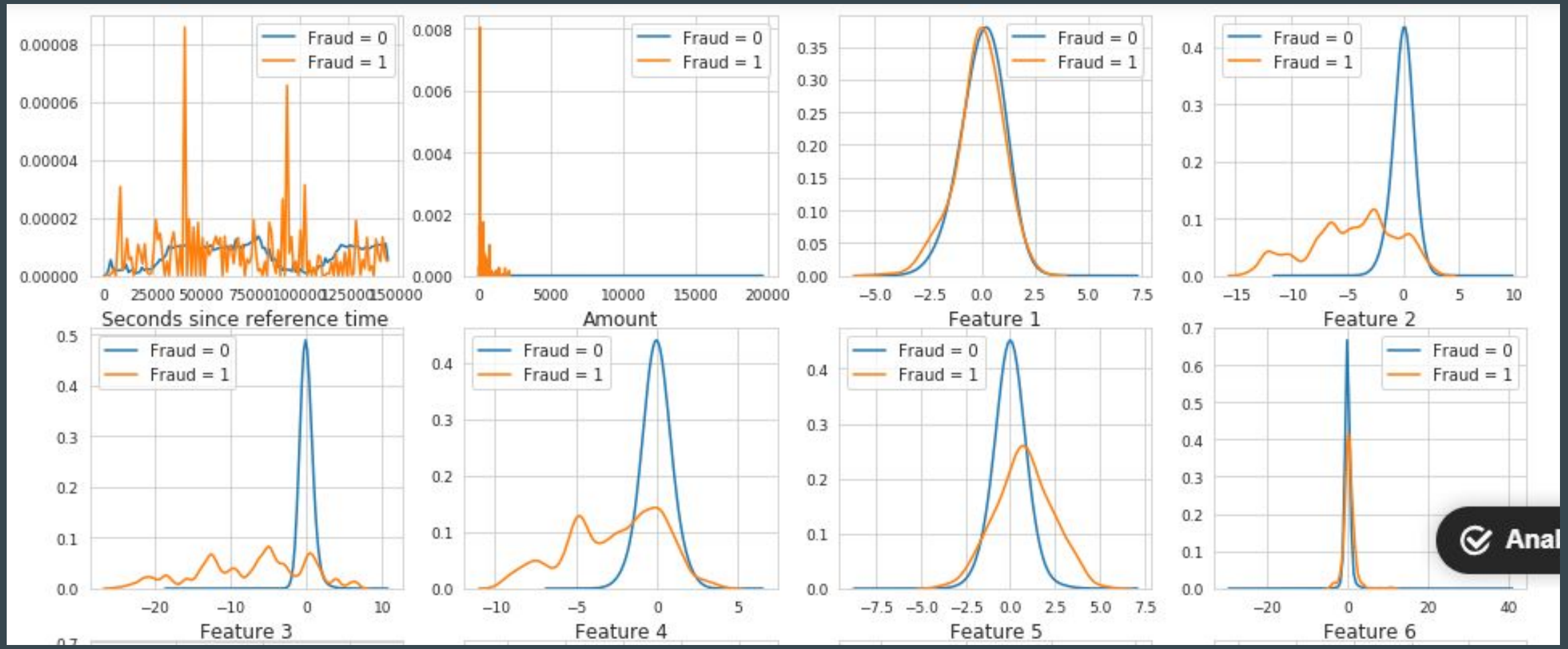

```
count      417.000000
mean       125.656379
std        256.880976
min          0.000000
25%          1.000000
50%         12.310000
75%        106.900000
max       2125.870000
Name: Amount, dtype: float64
```

Distribution of each features for each classes

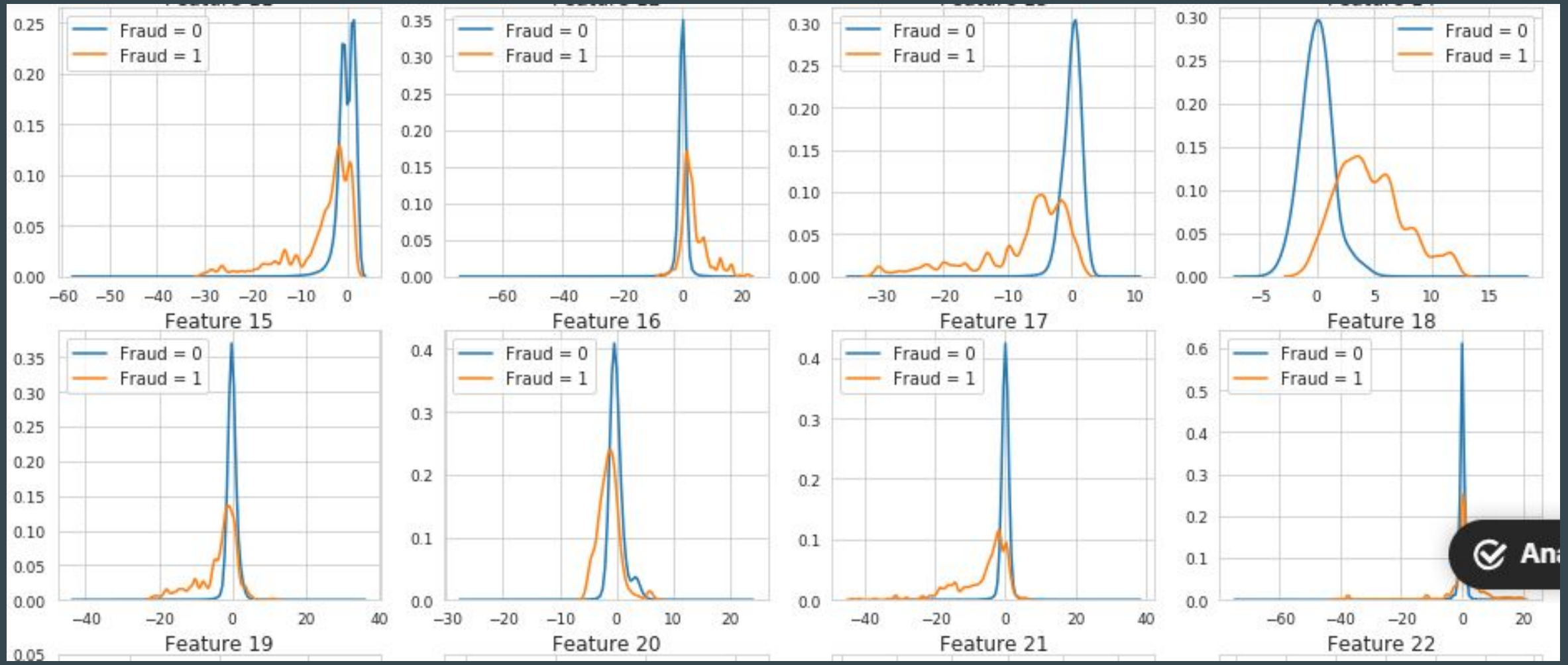
- To understand the distribution of each features for each classes.

```
var = df.columns.values

i = 0
t0 = df.loc[df['Fraud'] == 0]
t1 = df.loc[df['Fraud'] == 1]

sns.set_style('whitegrid')
plt.figure()
fig, ax = plt.subplots(8,4,figsize=(16,28))

for feature in var:
    i += 1
    plt.subplot(8,4,i)
    sns.kdeplot(t0[feature], bw=0.5,label="Fraud = 0")
    sns.kdeplot(t1[feature], bw=0.5,label="Fraud = 1")
    plt.xlabel(feature, fontsize=12)
    locs, labels = plt.xticks()
    plt.tick_params(axis='both', which='major', labelsize=12)
plt.show();
```

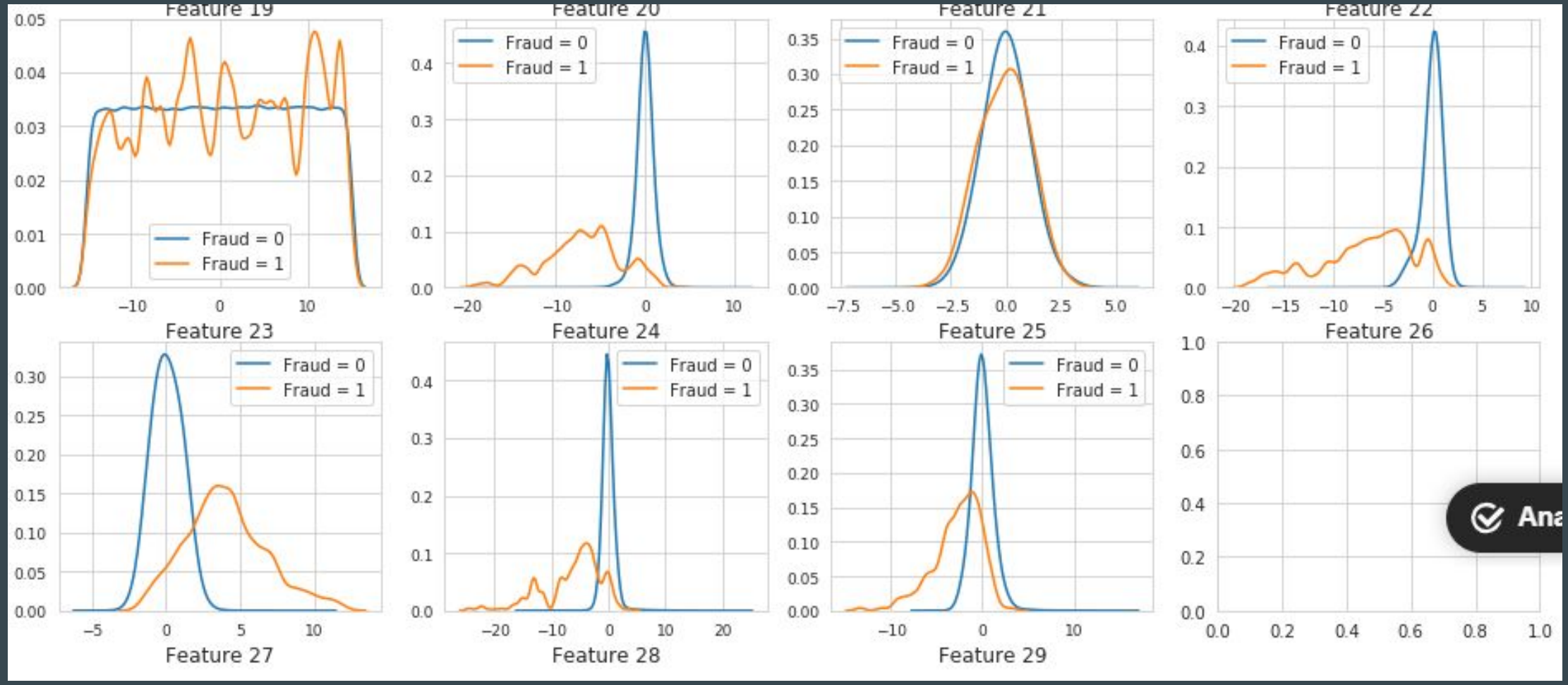
Distribution of each features for each classes



Distribution of each features for each classes



Distribution of each features for each classes



Limitations of Dataset

- Highly imbalanced dataset (A mere 0.18% of the dataset is classified as fraud, while the rest is non-fraud)
  - Requires us to conduct some sampling technique (depending on the technique, it will affect the training time and the amount of memory required to hold the training set)
- Do not know what are the features represents
  - We are unable to apply domain knowledge to conduct feature selection / apply weight
- Unable to conduct feature selection as features have already underwent PCA and is not a good practice to drop features after PCA

Data Preparation (Checking for duplicated rows)

```
#Check for duplicated rows
count = 0
for rows in df.duplicated():
    if rows == True:
        count += 1
if count > 0:
    print("There are",count,"duplicated rows")
else:
    print("There are no duplicated rows")
```

```
There are no duplicated rows
```

Data Preparation (Dropping Unwanted Records)

```
#check for null values
total = df.isnull().sum().sort_values(ascending = False)
percent = (df.isnull().sum()/df.isnull().count()*100).sort_values(ascending = False)
pd.concat([total, percent], axis=1, keys=['Total', 'Percent']).transpose()
```

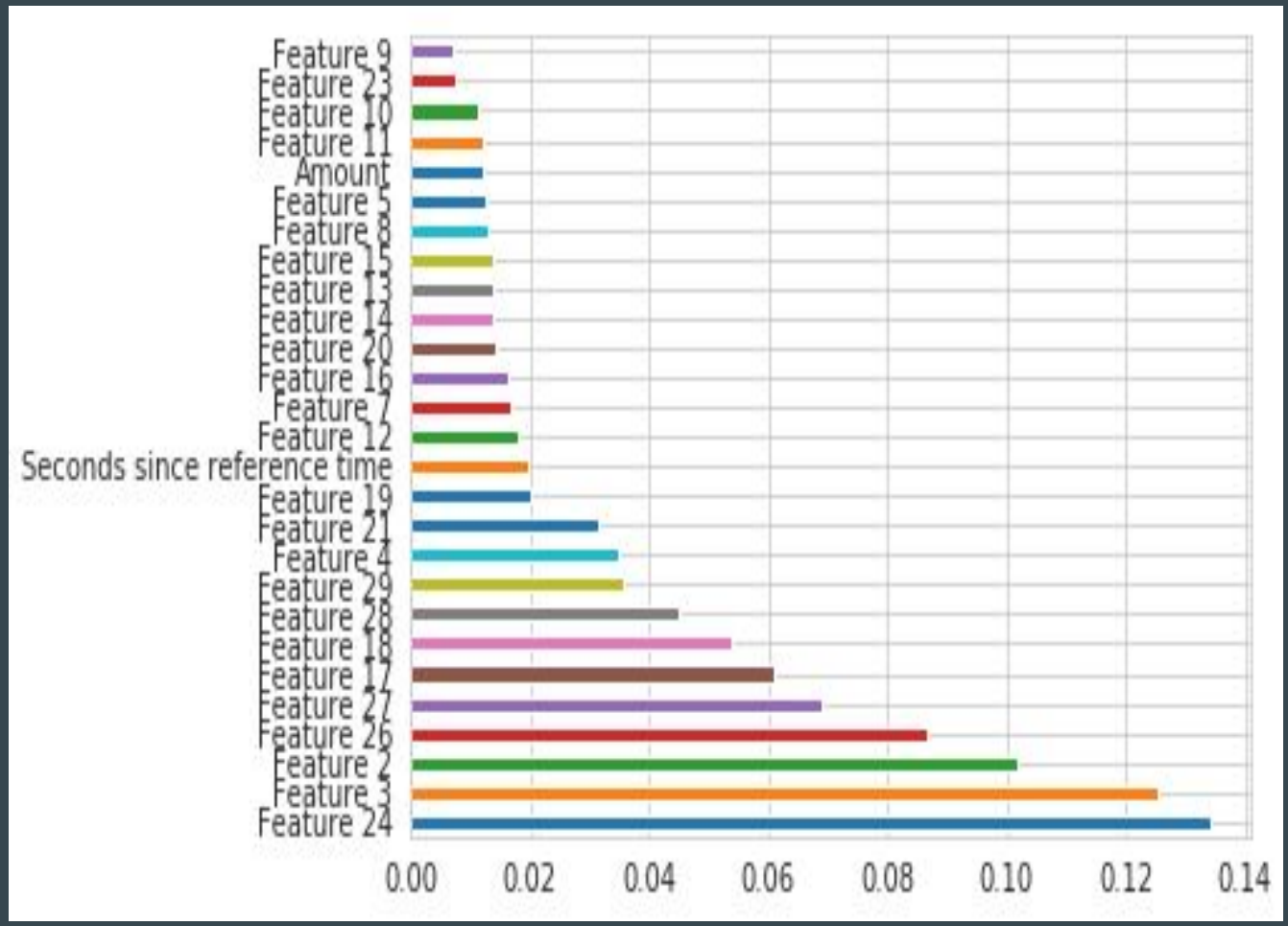
	Feature 6	Feature 5	F
Total	59.000000	27.00000	
Percent	0.025895	0.01185	

- Features 5 and 6 have 27 and 59 missing values respectively.
- Percentage of null values is less than 1% (yield is >99.4%, at least 4 sigma)
- Statistically safe to drop the rows with null values in terms of trade-off between analysis accuracy and data completeness.

Feature Importance

- Conducted Extra Tree Classifier
  - No sharp decline in importance of features
  - Dropping last 3 features gave a drop in performance

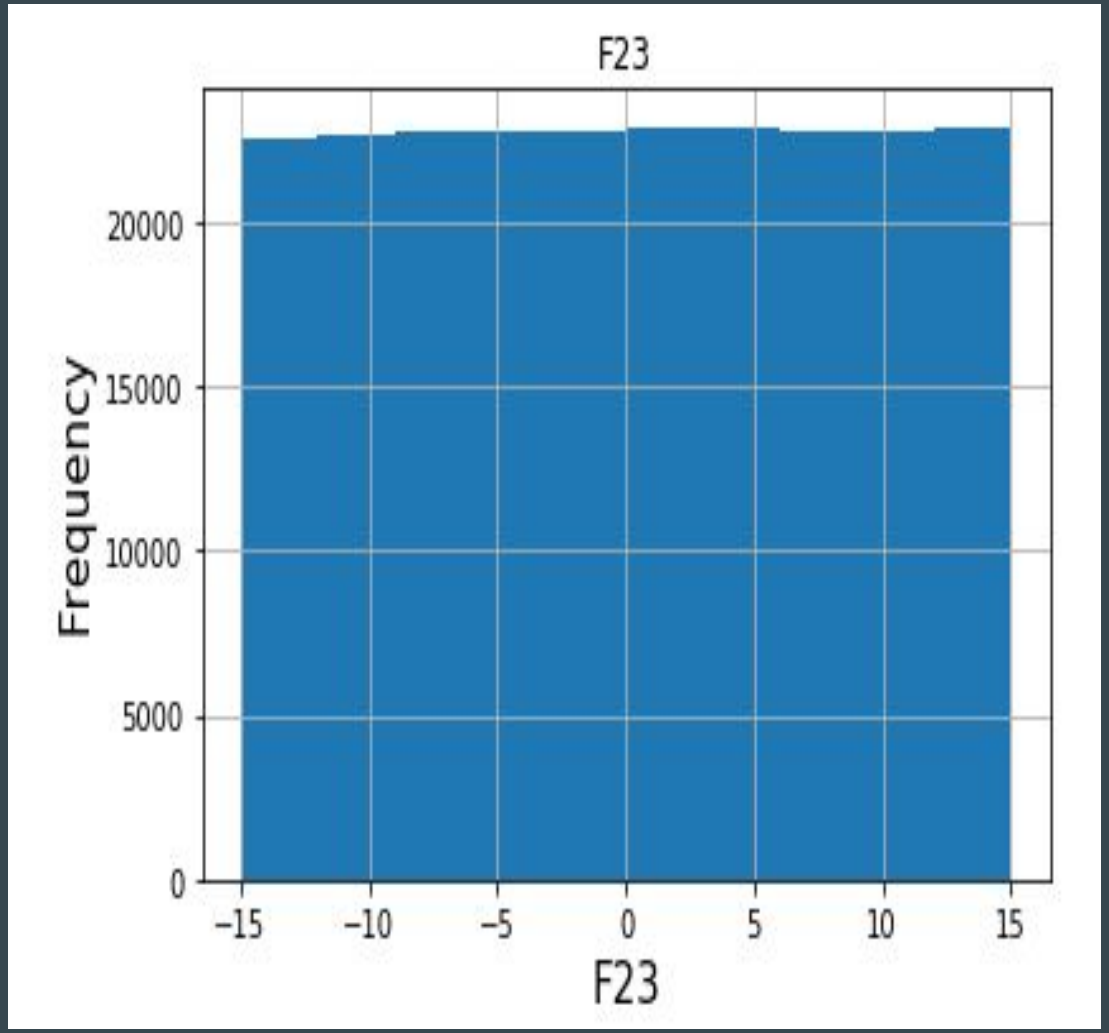
```
#feature importance (feature selection)
model = ExtraTreesClassifier()
model.fit(X,Y)
#plot graph of feature importances for visualization
feat_importances = pd.Series(model.feature_importances_, index=X.columns)
feat_importances.nlargest(31).plot(kind='barh')
plt.show()
```



Data Preparation (Feature Selection)

Feature 23 is ‘noisy’ with a near-uniform spread throughout the full range of the histogram.

We felt that it would cause overfitting of our model and greatly affect the metrics we chose.

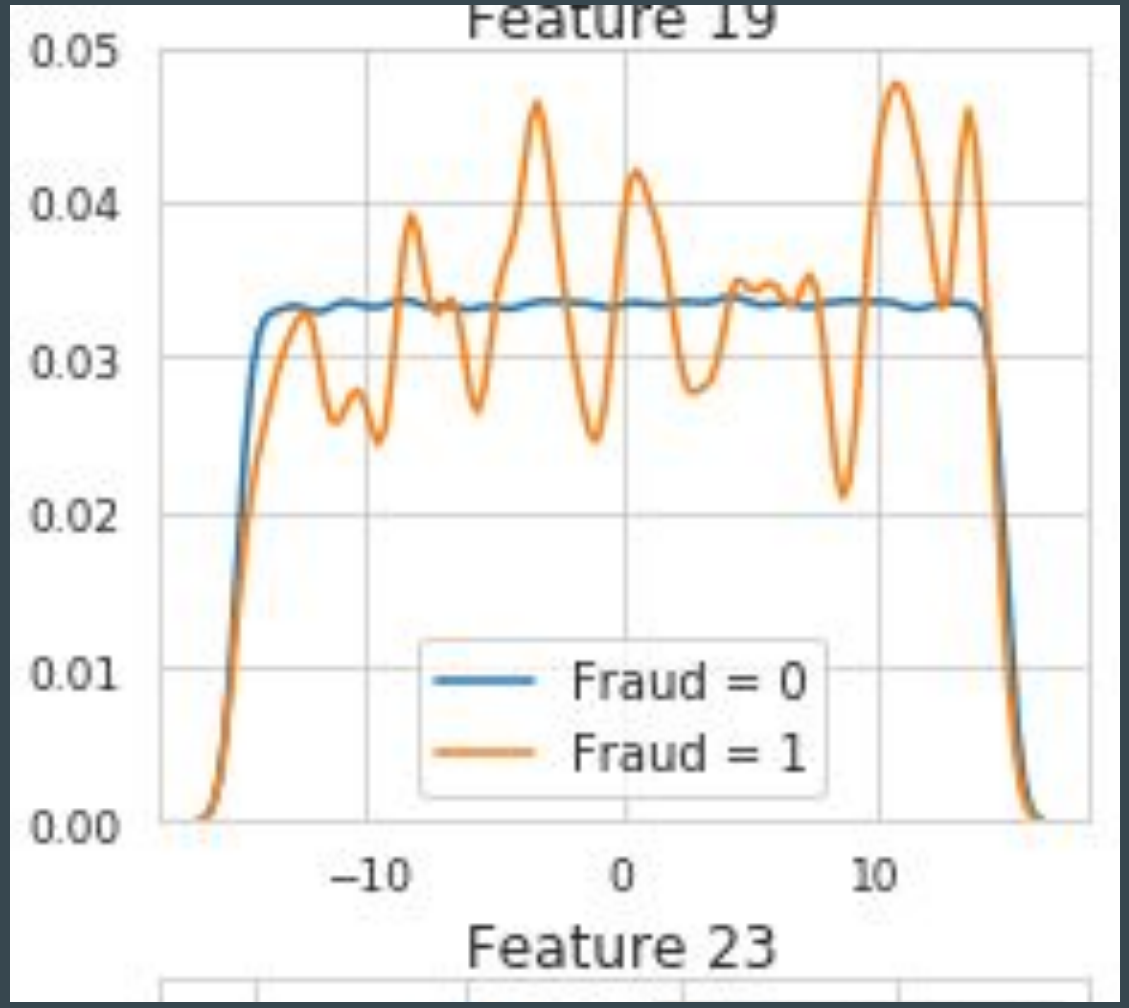


Data Preparation (Feature Selection)

However

Based on our feature density plot, we can observe that there is a distinct difference in distribution between valid and fraudulent transactions.

Decision: Include all features for our model



Data Preparation (Standardizing our data)

- Standardize features by removing the mean and scaling to unit variance
- Common requirement for many machine learning estimators
- Behaves badly if features do not look like standard normally distributed data
- Z score = (x - mean) / standard deviation

```
#Normalize our data
sc = StandardScaler()
X = sc.fit_transform(X)
```

```
# Splitting our X and Y variables
Y = df.Fraud
Y = Y.values.reshape(Y.shape[0],1)

df = df.drop('Fraud', 1)
X = df.values
```

Data Preparation (Random Oversampling)

Benefits of Random Oversampling:

1. Increase the footprint of the minority class
2. Model can learn to be more ‘sensitive’ to the minority class.

Why did we choose oversampling instead of undersampling?

Bringing down majority class to the size of minority class will result in

1. Too little data to work with (411 rows)
2. Discarding potentially useful information

Data Preparation (Train/test split and Random Oversampling)

1) Split the dataset into 80% training & 20% testing

2) Conduct random oversampling only on the training set to prevent any information from leaking to the testing data.

```
def Oversample(X_train,Y_train,print_output=False):
    Train_set = np.concatenate((X_train, Y_train), axis=1)

    #Convert back to dataframe for random oversampling
    df = pd.DataFrame.from_records(Train_set)

    # Class count
    count_class_0, count_class_1 = df.iloc[:,30].value_counts()

    # Divide by class
    df_class_0 = df[df.iloc[:,30] == 0]
    df_class_1 = df[df.iloc[:,30] == 1]

    df_class_1_over = df_class_1.sample(count_class_0, replace=True)
    df = pd.concat([df_class_0, df_class_1_over], axis=0)

    #shuffle rows
    df.sample(frac=1)

    # df_test_over.Class.value_counts().plot(kind='bar', title='Count (target)');
    Y_train = df.iloc[:,30].values
    X_train = df.iloc[:,0:30].values

    if print_output == True:
        print('Random over-sampling:')
        print(df.iloc[:,30].value_counts())

    return X_train, Y_train
```

Choosing our metrics

Rejected Metric

Accuracy - Misleading metric

- Heavily imbalanced classes
- Approximately 0.18% fraud and 99.82% otherwise

This means that if we predict every example as ‘non-fraud’, we would obtain the deceptive measure of 99.82% model accuracy.

Chosen Metrics

Precision Score

True positive divided by total predicted positive

This is good measure to determine whether the cost of False Positive is high.

Recall Score

True positives divided by total actual positive

It is important to evaluate since there is a high cost associated with False Negative.

F1 Score

Balance between Precision and Recall

Chosen Metrics

Trade Off between precision and recall:

> Based on the given business context, the cost of not identifying a fraudulent transaction when we are supposed to, has very dire consequences.

> However, incorrectly predicting too many false frauds will hinder the company's operations and possibly losing customers.

> Solution: Assign higher weights to recall score

Chosen Metrics

F-Beta Score

- Assigns weight to either precision or recall
- Prioritize Recall score since undetected fraudulent transactions are of 25Credit's highest priority.
- Allocated approximately 75% weightage to the recall and 25% weightage to precision

AUC

- Area under the Receiver Operating Characteristic curve.
- The higher the area under the curve, the better its able to distinguish between positive class and negative class.

The chosen model

Artificial Neural Network

> Initially, our team attempted using *Logistic Regression*
> Did not perform as well as our Neural Network model (based on our chosen metrics).
> Neural network is a popular solution in fraud detection.
> Success stories include:
  - León Bank in Dominican Republic reduced fraud by 60% in the first 3-months of using the system.
  - Guayaqyuil Bank in Ecuador had 100% detection of fraud cases in the first month.

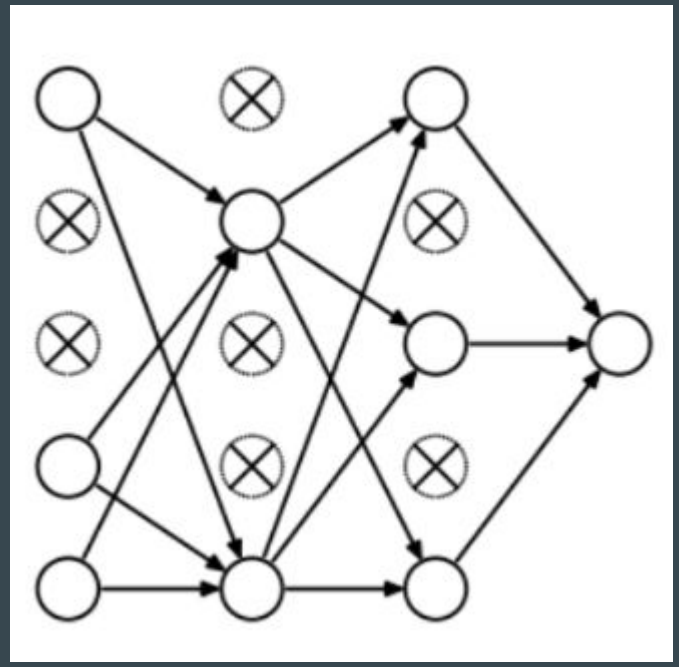
Advantages

- Ability to represent variables with a complex and nonlinear behavior with a very high accuracy.
- Many relationships between inputs and outputs are non-linear & complex
- Easy customizable to different business environment, functions.
- Future proof: Ability to take in lots of input, may require to have image/signature recognition in the future to authenticate high-value transactions

Considerations to prevent overfitting

Regularization

- Discourages learning a more complex or flexible model to avoid risk of overfitting
- Penalizes the weight matrices of the nodes
- Dropout technique
  - At every iteration, randomly selects some nodes
  - Remove them and all of their connections
  - Captures more randomness



Considerations to prevent overfitting

Early Stopping

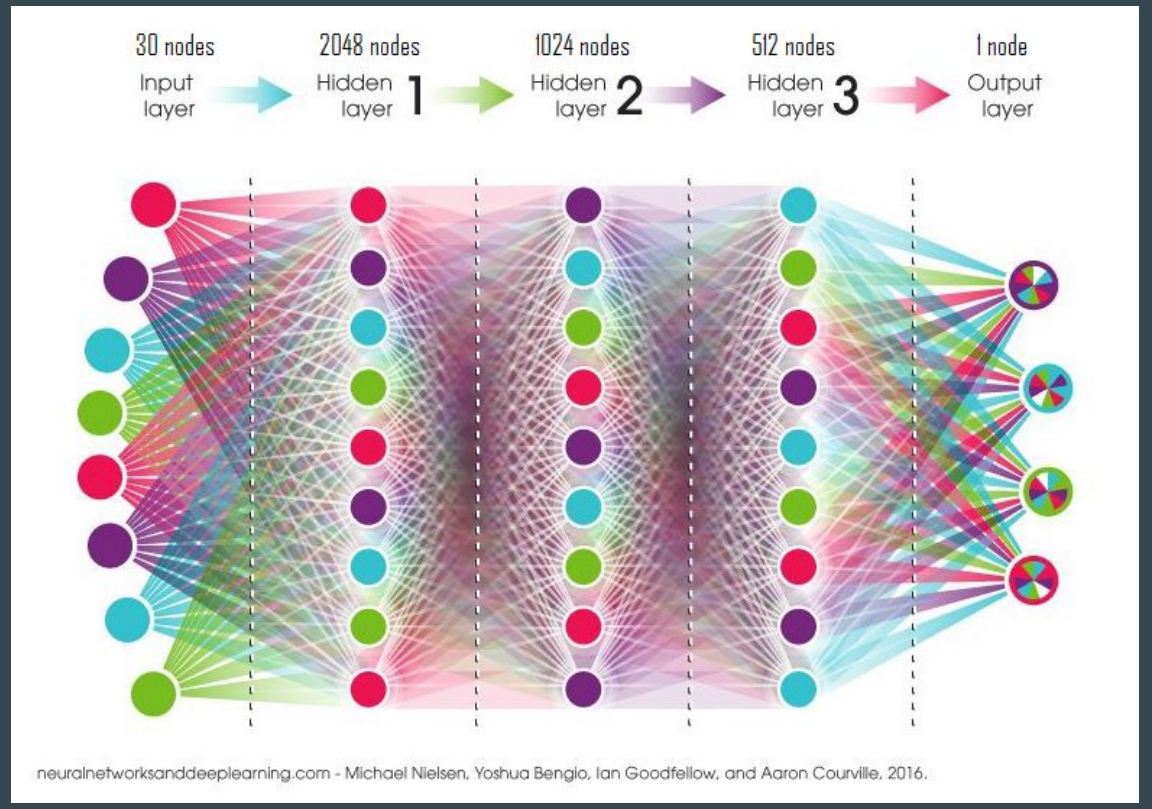
- Too little or too much training results in under or overfitting
- Stops training when performance has stopped improving on validation set
- Assign a large number of epoch
- Stop training when the model does not see an improvement after 7 epochs
- Save the last best known model

```
epochs = 50
batch_size = 72

callbacks = [EarlyStopping(monitor='val_loss', patience=7),
             ModelCheckpoint(filepath='saved_model_trained.hdf5', monitor='val_loss', save_best_only=True)]
```

Network Structure

Defining the neural network layers

1) Input layer: 31 nodes
2) Hidden layer 1: 2048 nodes (15% dropout)
3) Hidden layer 2: 1024 nodes (5% dropout)
4) Hidden layer 3: 512 nodes
5) Output layer: 1




```
model = Sequential()
model.add(Dense(2048,activation='relu'))
model.add(Dropout(0.15))
model.add(Dense(1024,activation='relu'))
model.add(Dropout(0.05))
model.add(Dense(512,activation='relu'))
model.add(Dense(1, activation='sigmoid'))
```

Network Structure

Activation functions

Hidden layers: ReLU (Rectified Linear Unit)

- More computationally efficient and better convergence performance than sigmoid.

Output layer: Sigmoid

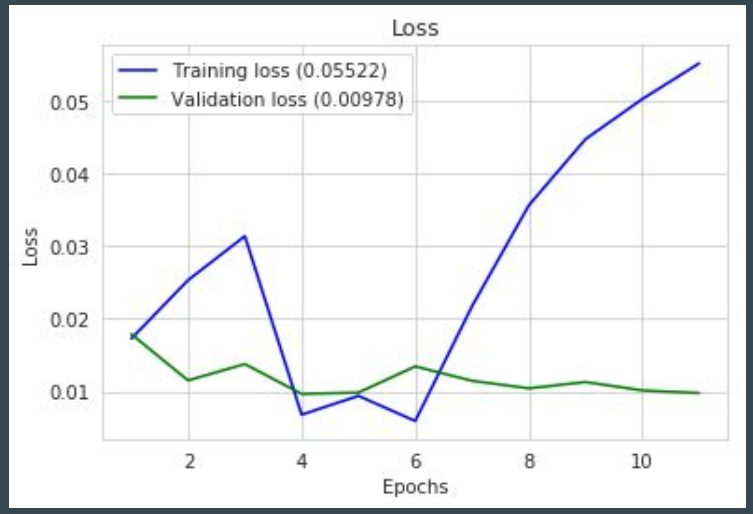
- For predicting the probability of output (useful later when sorting our fraudulent predictions)

Defining loss function

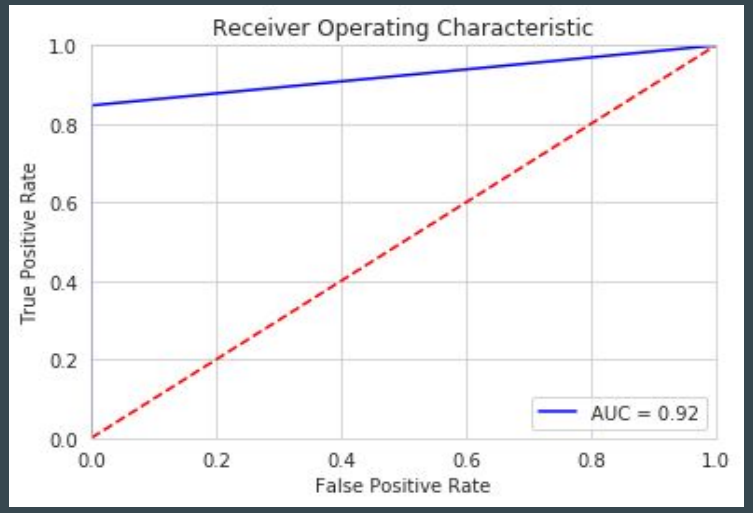
- Binary cross entropy
- Binary classifier problem

Training our Model

Accuracy for Model : 1.0
Precision for Model : 0.79
Sensitivity/Recall for Model : 0.85
F1 Score for Model : 0.82
F-Beta Score for Model : 0.84



Training was stopped before the training and validation loss diverged (this began after 5th epoch)



Cross validating the Model

Why Cross Validate?

> Assess the predictive performance of the model

> Judge how it will perform outside the sample to a new data set

- Without cross validation, we only have information on how our model performs on training data

> Increases our confidence in the model as it helps us to better use our data

> Provides more information about the model's performance.

> Convince and assure 25Credit that our model is reliable

Cross validating the Model

Cross Validation Method

- Using K-fold Cross Validation technique:
  - K = 5 folds
- Same approach as initial training of model (Oversampling, no. of epoch etc)
- Practical balance between time and reliability of the measure (about 4 hours to run)
- We also oversample our training set during this process

```
#Define parameters for k-fold cross val
n_folds = 5
kf = KFold(n_folds)
Metric_array = np.zeros(6)

epochs = 5
batch_size = 72
counter = 1

#Start k-fold cross validation
for train_index, test_index in kf.split(X):

    #Split into training and testing set
    X_train, X_test = X[train_index], X[test_index]
    Y_train, Y_test = Y[train_index], Y[test_index]

    #Oversampling of training set ONLY
    X_train, Y_train = Oversample(X_train,Y_train)

    #run the model
    history = model.fit(X_train, Y_train, batch_size=batch_size, epochs=epochs)

    #Run forward prop to get predicted values
    Y_test_hat = model.predict(X_test)
    Convert_prob_to_class(Y_test_hat)

    print('\nMetrics for fold number: ' + str(counter))
    counter += 1

    #Print metrics and store in numpy array to average after end of cv
    Metric_array += PrintStats(Y_test,Y_test_hat,history,True)
```

Initial Score vs Cross Validation Score

Initial Summary Metrics

- Accuracy for Model : 1.0
- Precision for Model : 0.79
- Sensitivity/Recall for Model : 0.85
- F1 Score for Model : 0.82
- F-Beta Score for Model : 0.84
- AUC for Model : 0.92

Average Summary Metrics (5-fold)

- Accuracy for Model : 1.0
- Precision for Model : 0.78
- Sensitivity/Recall for Model : 0.96
- F1 Score for Model : 0.85
- F-Beta Score for Model : 0.91
- AUC for Model : 0.98

Fitting this into the business

Implementation

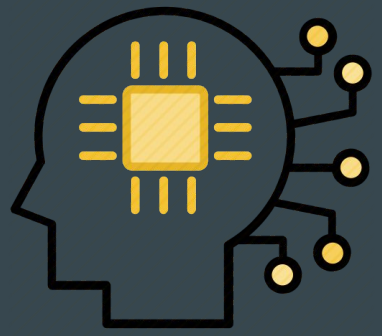
Our current model flags incoming transactions as fraudulent if given a **risk score** between **0.5** and **1** based on:

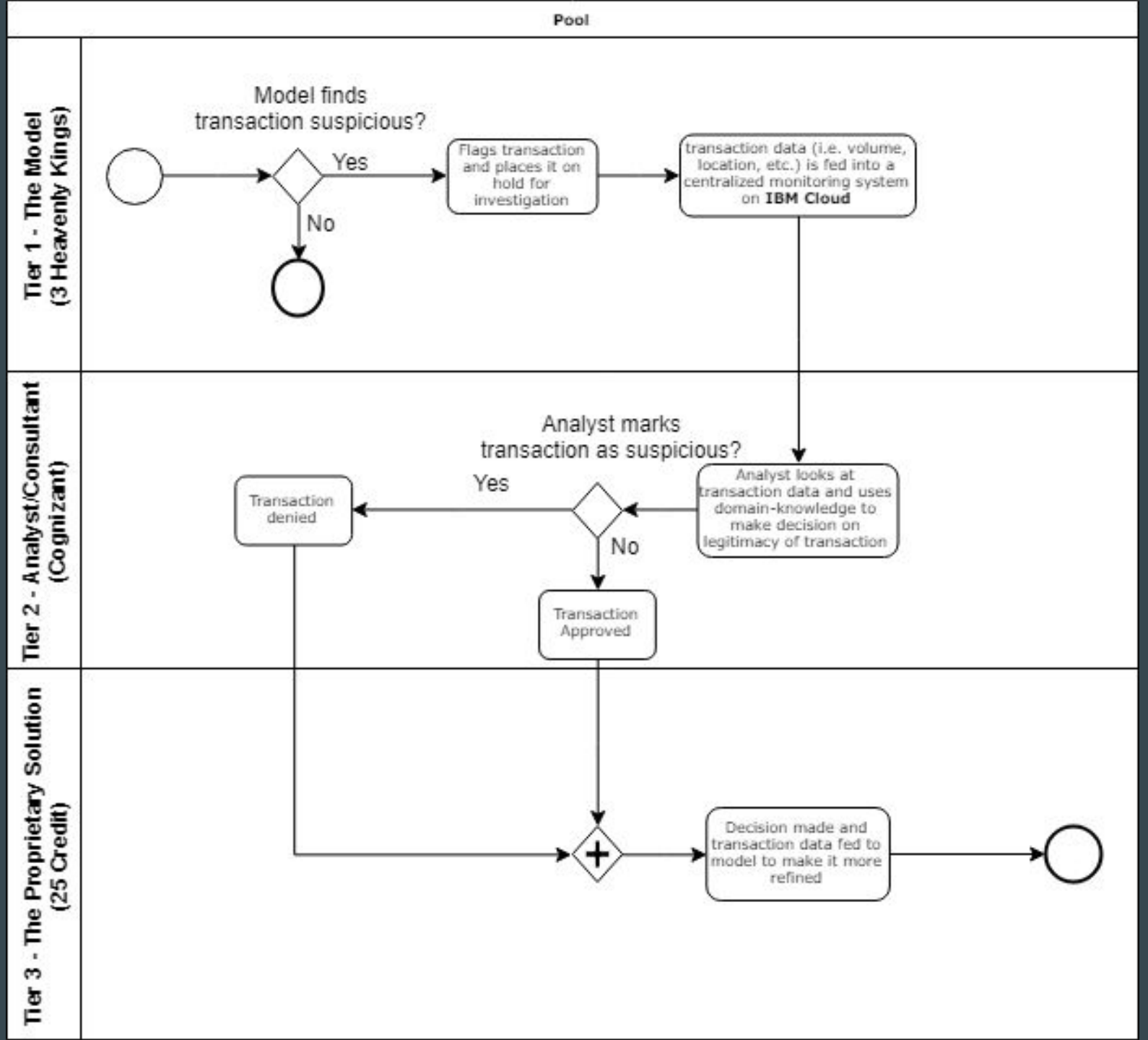
1. Probability of fraud (Given by model)
2. Transaction amount ($)
3. ‘Importance’ or ‘Value’ of customers based on certain factors such as type of card, income/asset level etc.
4. What type of transaction it is (CNP, credit card, refund, etc.)
5. Deviation from usual typical purchase timing (e.g. big ticket item bought in graveyard hours)

Implementation

Process Flow

1. [Tier 1] Our model **flags** suspicious transactions and places them on hold for investigation. Transaction data (i.e. volume, location, etc. of transaction) is stored and fed into a centralized monitoring system.
2. [Tier 2] A trained Analyst looks at these flagged transactions and conducts her investigation to decide on its authenticity based on her **domain knowledge**. It is then within her and her supervisor's jurisdiction to take action to resolve and mark the transaction.
3. [Tier 3] This subsequent decision (whether transaction is cleared or rejected) along with the transaction data is fed back to the model to train it iteratively so that over time, 25Credit has an **ever-more accurate in-house system** on license from Cognizant to monitor its transactions, for the safety of its clients.




Pool
Tier 1 - The Model (3 Heavenly Kings)
Model finds transaction suspicious?
Yes
No
Flags transaction and places it on hold for investigation
transaction data (i.e. volume, location, etc.) is fed into a centralized monitoring system on IBM Cloud
Tier 2 - Analyst/Consultant (Cognizant)
Analyst marks transaction as suspicious?
Yes
No
Transaction denied
Analyst looks at transaction data and uses domain-knowledge to make decision on legitimacy of transaction
Transaction Approved
Tier 3 - The Proprietary Solution (25 Credit)
Decision made and transaction data fed to model to make it more refined

Limitations of our model

- PCA features are stripped of original meaning/context - challenge to feature selection
- Computationally demanding and expensive
- Time consuming to train - however with proprietary use of extra machines can be overcome
- No clear rules on how to set parameters for our model (mostly following rule of thumb) and trial and error
- Our neural network model exists as a black box
  - Features are *not* human interpretable
  - Hard to understand the cause of prediction and mistake; Challenge of explaining why the transaction is flagged
  - Solution: **A move towards XAI Neural Networks**

Overcoming the limitations of model & solution

When false positive detection occurs..

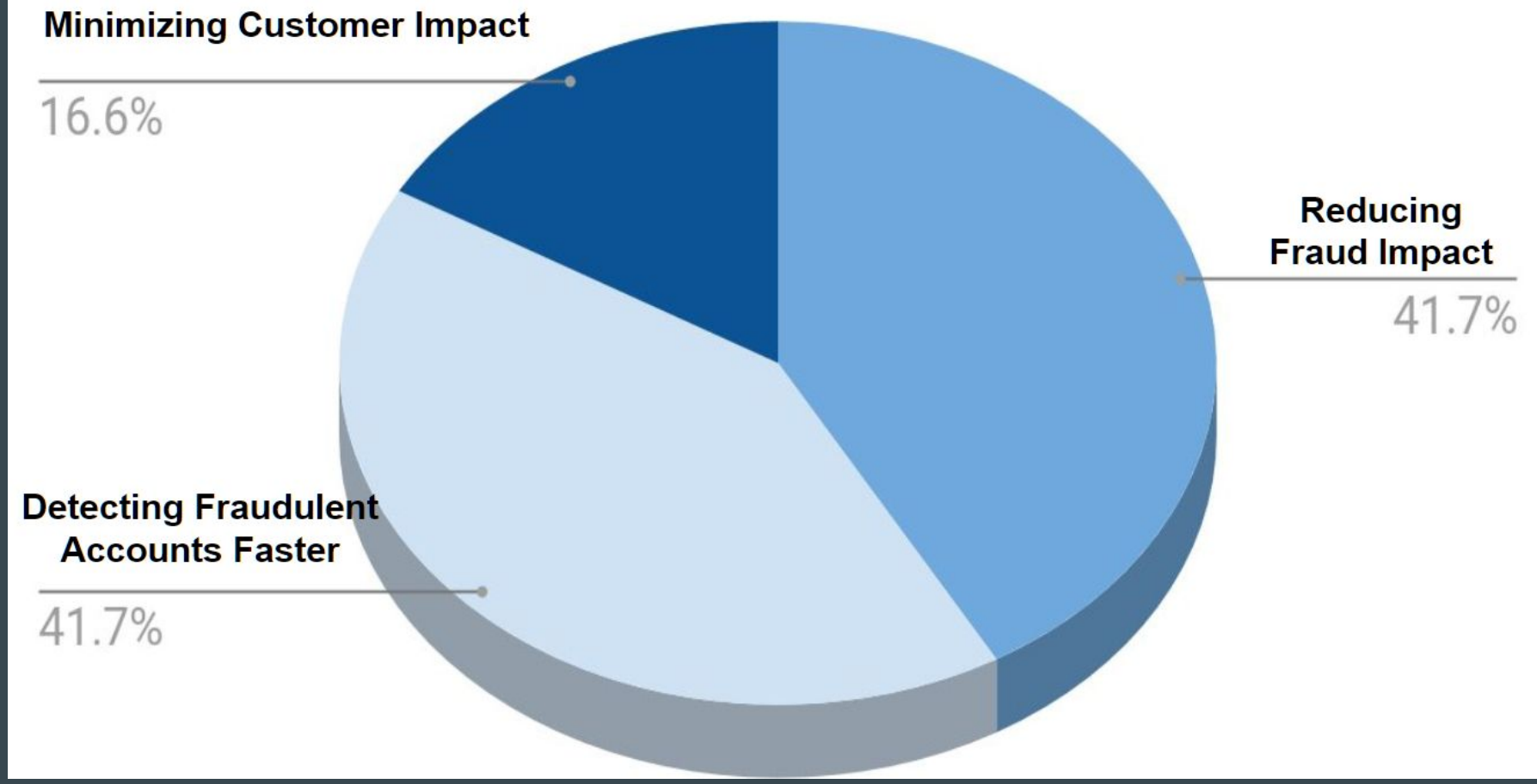
- Do **Root Cause Analysis** to find out root cause of falsely detection
  - Through cross-validation of other models + analyst investigation
- Upon discovery of false detection, 25Credit dispenses rewards/loyalty points to the customer.
- The analyst then looks through the logged data in our **IBM cloud** store to check if similar occurrences happened before. If so, the model shall be tweaked to this new information to identify fraudulent transactions as fraud patterns always evolve with time.

Further potential improvements to model

1. Understanding 25 Credit's main fraud goal

Top 3 Fraud Goals of Financial Institutions

Below are the Top 3 fraud goals listed by FIs:



Further potential improvements to model

1. Understanding 25 Credit's main fraud goal
2. Application of domain knowledge to apply weights to targeted features

Blood in the Water - Incipient Crises in Credit Fraud

As per *Rippleshot (State of Credit Fraud 2018):*

Account Takeover Fraud

- "account takeovers jumped by 300 percent in the 2017 alone."
- "average resolution time of 16 hours" - our **business goal**

Synthetic Fraud

- "estimated to account for **85%** of **all identity fraud** in the U.S."
- "accounts for **80%** of **all credit card fraud losses**, and nearly **1/5** of **credit card charge-offs**."; "By 2021, synthetic fraud is estimated to account for 40% of all credit card charge-offs"
- "develop over a period of 6 months to 5 years" - our **machine calibration**

Blood in the Water - Incipient Crises in Credit Fraud

ATM Fraud

- "When an ATM breach transpires, fraud occurs within **48 hours**."; "*average* ATM skimming fraud costs **$600 per card**" - our **machine calibration**
- Taking action:
  - **First 2 Hours**: Detect ATM Skimming Locations
  - **4 Hour** mark: Identify compromised cards and reset pins
  - **End Goal**: **80% Fraud Reduction**

Gas Station Skimming Fraud

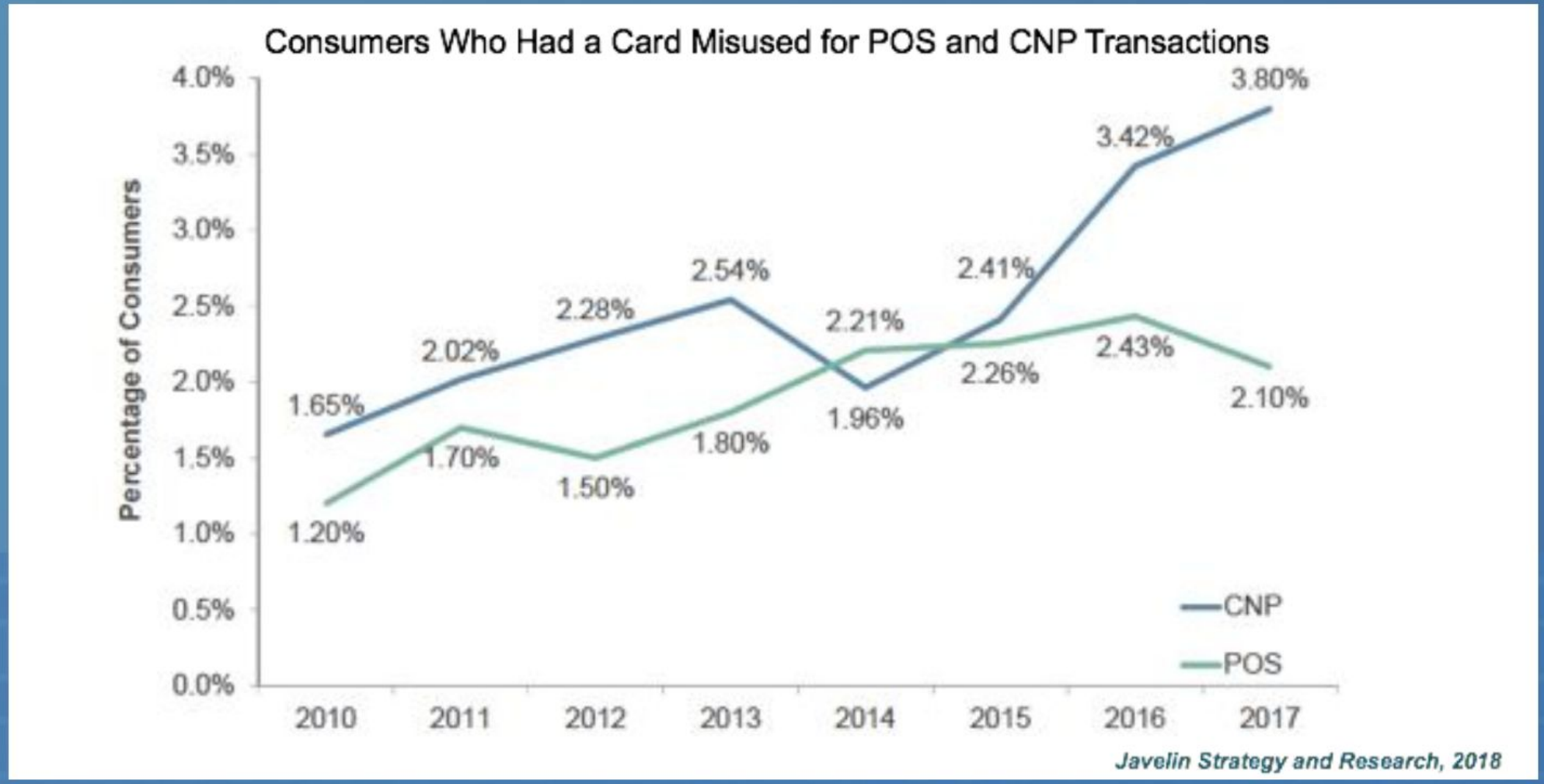
- "single compromised pump can capture data from roughly 30-100 cards per day."
- U.S.: "59 skimmers at 85 locations in 21 states." - integration of **geospatial data** to dataset in analysis of fraud *within ATM and POS categories*

Further potential improvements to model

1. Understanding 25 Credit's main fraud goal
2. Application of domain knowledge to apply weights to targeted features
3. Segmentation based on purchase method

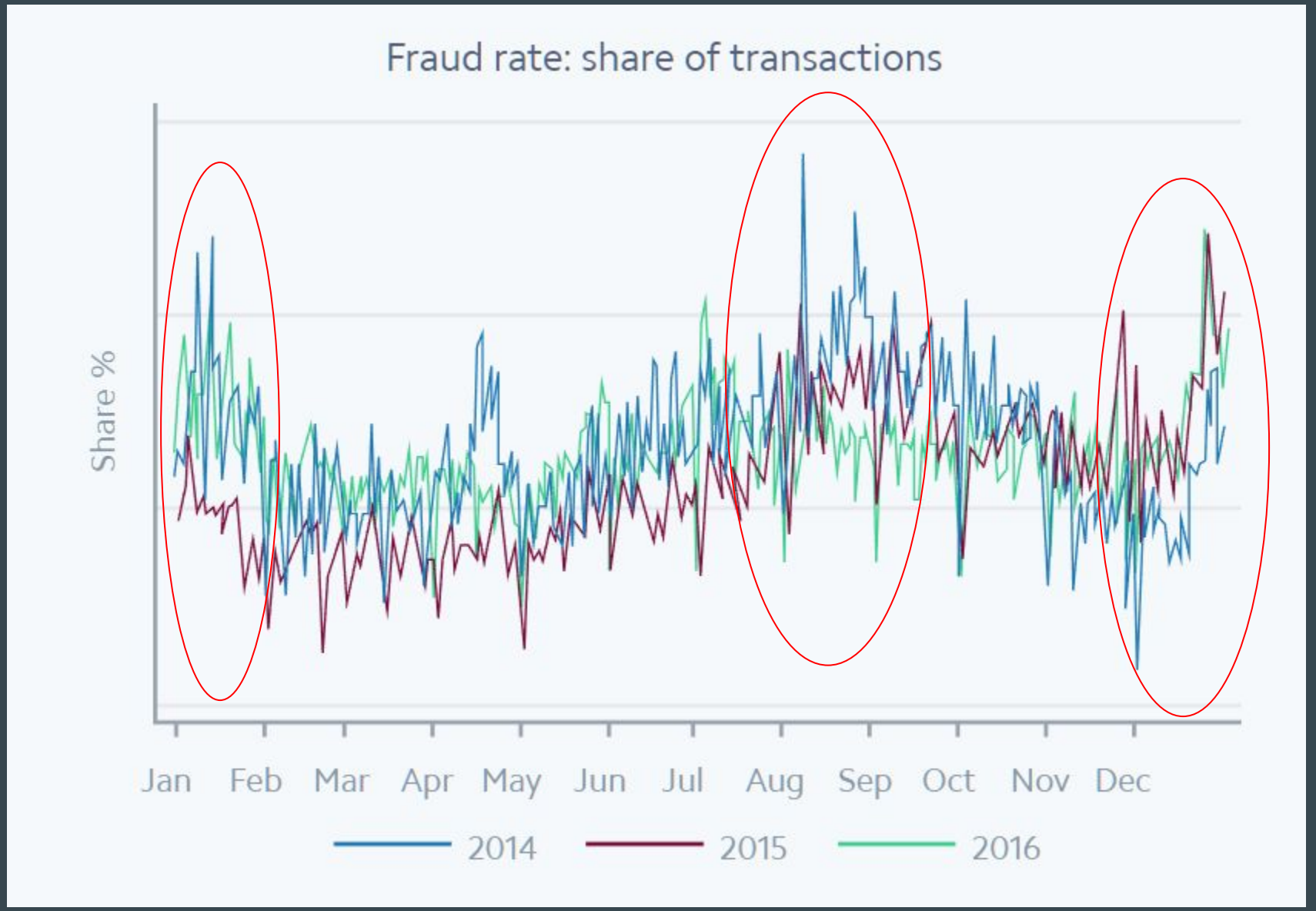
Analysis by segmentation of type of credit fraud

POS fraud is trending down — but CNP fraud is skyrocketing

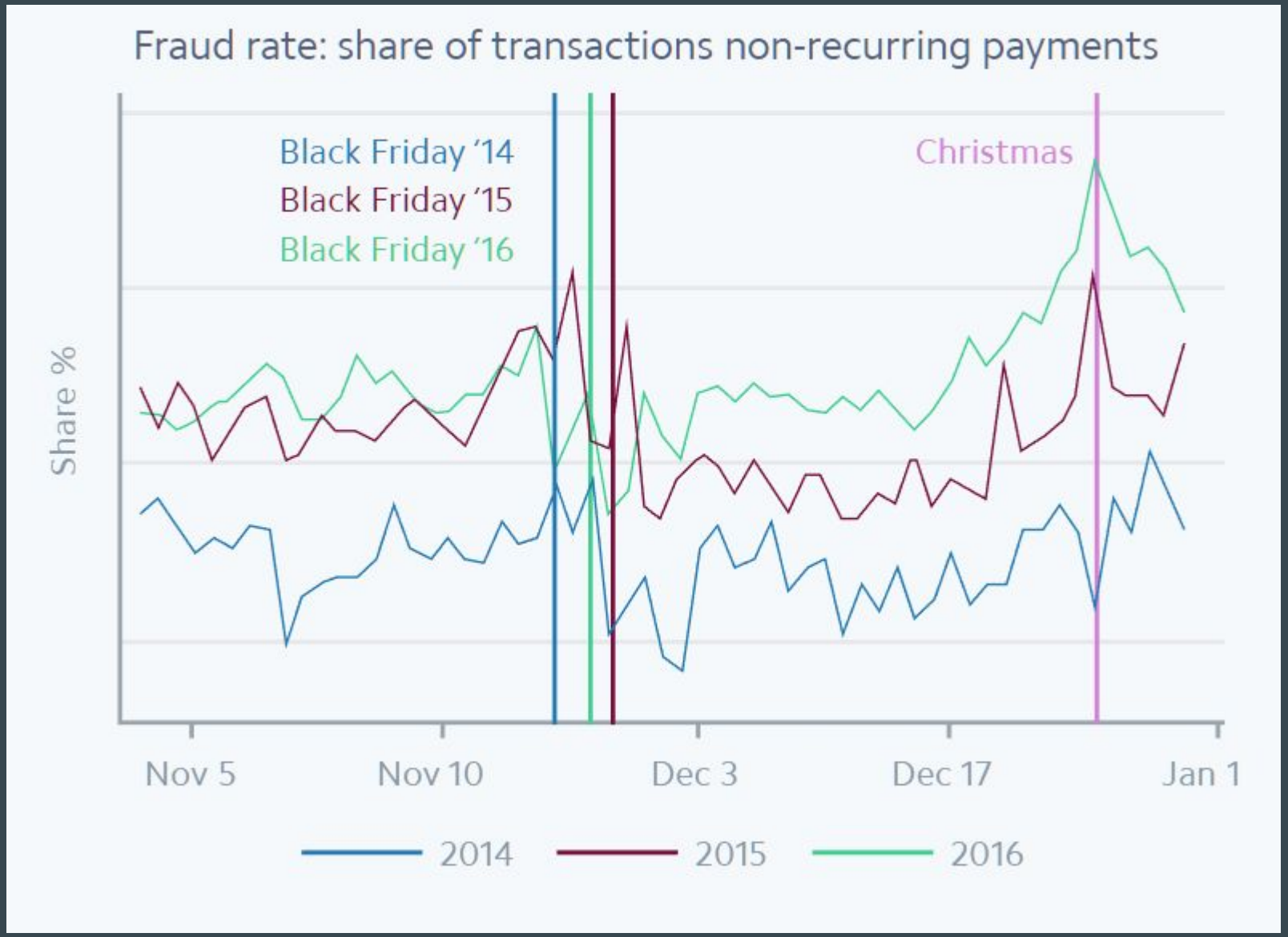


Further potential improvements to model

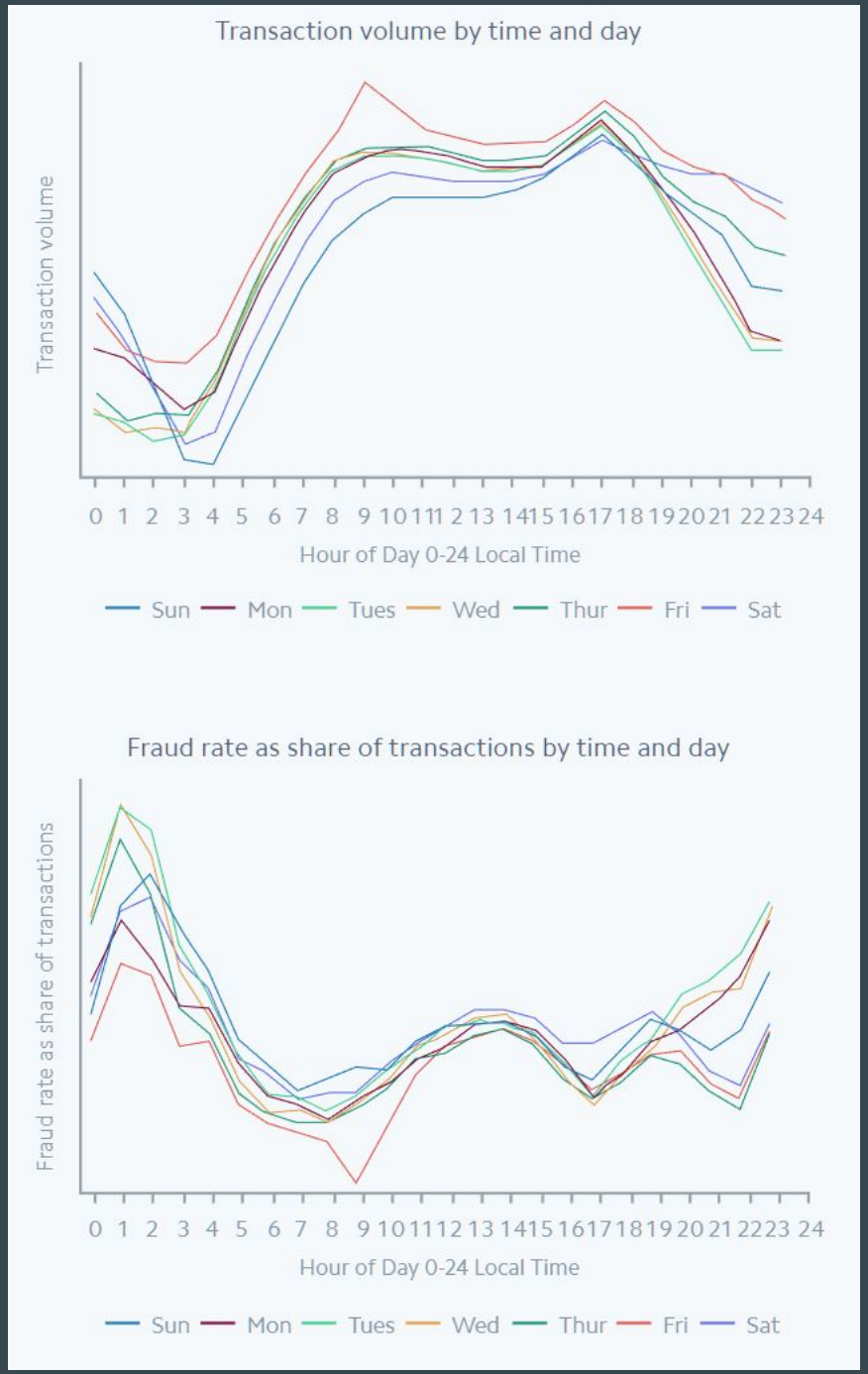
1. Understanding 25 Credit's main fraud goal
2. Application of domain knowledge to apply weights to targeted features
3. Segmentation based on purchase method
4. Increased temporal element in analysis



Data: Stripe (2018)

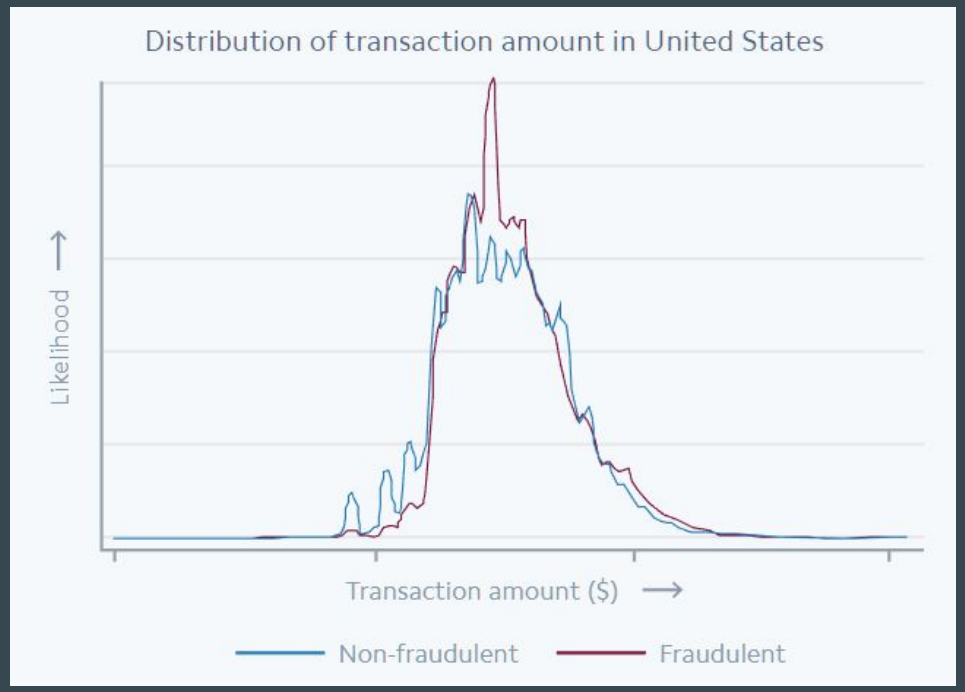


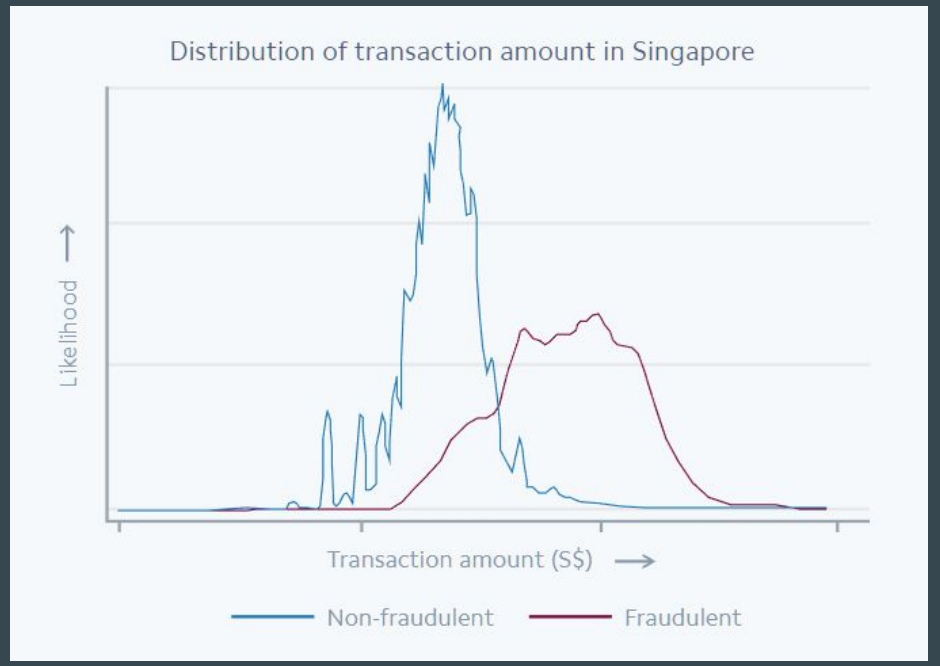
Data: Stripe (2018)

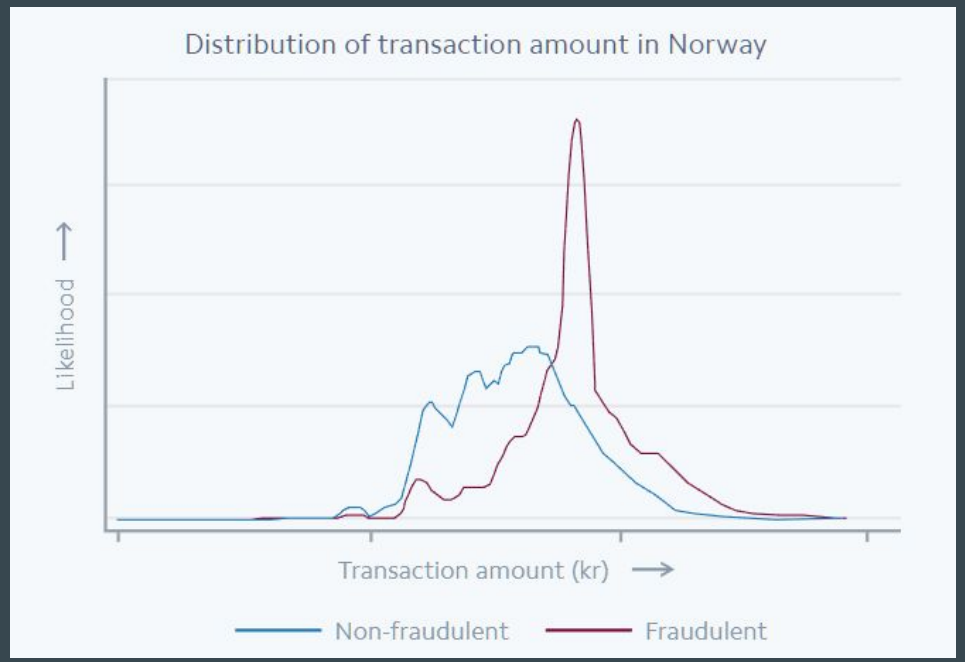

Transaction volume by time and day
Transaction volume
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24
Hour of Day 0-24 Local Time
Sun Mon Tues Wed Thur Fri Sat
Fraud rate as share of transactions by time and day
Fraud rate as share of transactions
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24
Hour of Day 0-24 Local Time
Sun Mon Tues Wed Thur Fri Sat

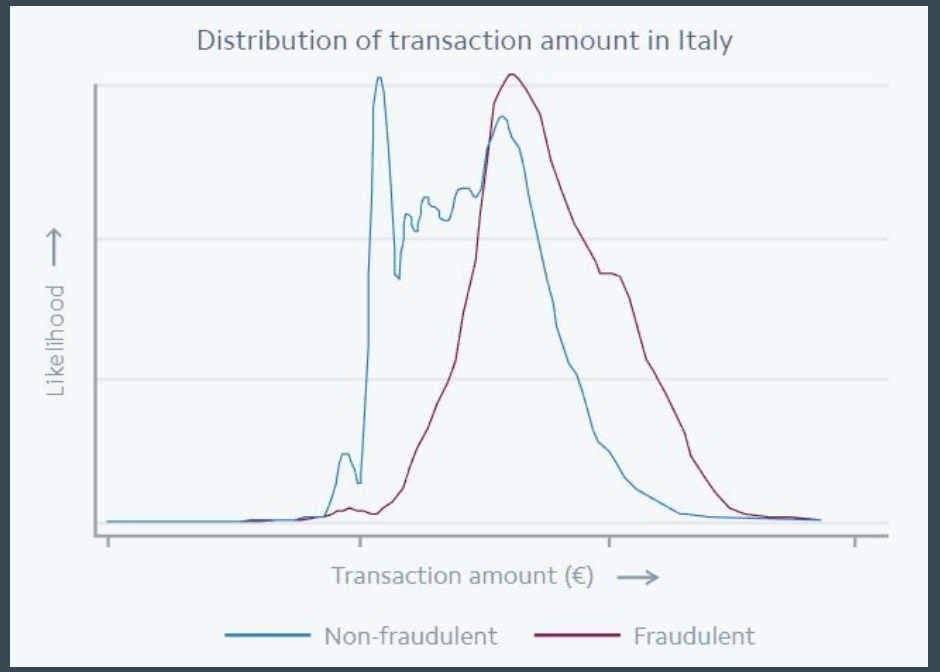
Further potential improvements to model

1. Understanding 25 Credit's main fraud goal
2. Application of domain knowledge to apply weights to targeted features
3. Segmentation based on purchase method
4. Increased temporal element in analysis
5. Tracking user IP and geolocation to spot potential unauthorized transactions (**Geolocation validation**)


Distribution of transaction amount in United States
Likelihood
Transaction amount ($)
Non-fraudulent
Fraudulent


Distribution of transaction amount in Singapore
Likelihood
Transaction amount (S$)
Non-fraudulent
Fraudulent


Distribution of transaction amount in Norway
Likelihood
Transaction amount (kr)
Non-fraudulent
Fraudulent


Distribution of transaction amount in Italy
Likelihood
Transaction amount (€)
Non-fraudulent
Fraudulent

Further potential improvements to model

1. Understanding 25 Credit's main fraud goal
2. Application of domain knowledge to apply weights to targeted features
3. Segmentation based on purchase method
4. Increased temporal element in analysis
5. Tracking user IP and geolocation to spot potential unauthorized transactions (**Geolocation validation**)
6. Using SMOTE oversampling to increase the generalizability of the model
    - Uses **k-nearest neighbours** to exclude members of the majority class while simultaneously creating synthetic examples of a minority class.

Further potential improvements to model

1. Understanding 25 Credit's main fraud goal
2. Application of domain knowledge to apply weights to targeted features
3. Segmentation based on purchase method
4. Increased temporal element in analysis
5. Tracking user IP and geolocation to spot potential unauthorized transactions (**Geolocation validation**)
6. Using SMOTE oversampling to increase the generalizability of the model
    - Uses **k-nearest neighbours** to exclude members of the majority class while simultaneously creating synthetic examples of a minority class.
7. Combining our model with other machine learning techniques such as random forest to improve performance [**Ensemble Learning**]

End Targets for our Model

- Exceed industry standards of fraud detection completeness while minimizing human resource costs
  - Presently, **1/12 of a percent (0.083%)** of transactions flagged as fraud; **0.18%** fraudulent transactions in provided dataset by 25Credit - a huge amount given the volume of transactions faced
  - Solution: Set crunch season of first quarter to analyse **ALL** flagged transactions, subsequently, as per actual fraud results, automatically decline the top x% (e.g. 80%) transactions of fraud likelihood, only manually check the *borderline cases*. Results of analysis fed back into model - greater risk score accuracy over time.

Example: