Machine Learning and Fraud Detection

February 2020

Tamsin Crossland – Senior Architect

@CrosslandTamsin

icon solutions

World Class Payment and Enterprise Solutions
for the global financial sector

Two main types of article on AI

BIZCATALYST 360°
LIFE • CULTURE • BIZ
REIMAGINED

architectures essentially facilitate diffusion of infor-
aphs. The more GCN layers an architecture contains,
ly the information is smoothed. It is therefore impor-
tant to recognize that the performance of a traditional GCN relies
heavily on a high degree of intraclass clustering [7]. This is a re-
sult of the convolutional kernel being determined by the adjacency
matrix A, which is inherently fixed. Also note that all vertices are
either binary encoded or represented by a single weight. This is a
limiting aspect when it comes to graphs containing more detailed
etween vertices.

ameworks

dard GCN (as displayed in Equa-
pecial case of a message-passing
difference in notation is that an
t the node level. This is more in
on of the architectures proposed
d will therefore be used for the

inary adjacency matrix, Equation

$$+ \sum_{j \in N_i} \boldsymbol{h}_j^{(l)} \Big| \boldsymbol{W}^{(l)} \Big), \tag{2}$$

g of node v_i in layer l, N_i denotes
ode v_i, and c_i is a node-specific
ypically takes a value of $|N_i| + 1$
l self-connections), in case of an

ights w_{ij}, Equation 2 can be ex-

$$+ \sum_{j \in N_i} w_{ij} \boldsymbol{h}_j^{(l)} \Big| \boldsymbol{W}^{(l)} \Big), \tag{3}$$

w vertices to retain their original properties

context of message passing frameworks, this leads to the following propagation rule:

$$\boldsymbol{h}_i^{(l+1)} = \sigma\left(\boldsymbol{h}_i^{(l)} \boldsymbol{W}_s^{(l)} + \sum_{r \in \mathcal{R}} \sum_{j \in N_i^r} \frac{1}{c_{i,r}} \boldsymbol{h}_j^{(l)} \boldsymbol{W}_r^{(l)}\right), \tag{5}$$

where $\boldsymbol{W}_r \in \mathbb{R}^{|\mathcal{F}^{(l)}| \times |\mathcal{F}^{(l+1)}|}$ now are *relation-specific* weight matrices, $\boldsymbol{W}_s$ denotes the weight matrix associated with self-connections (both trainable), and $c_{i,r}$ are relation-specific normalization constants, typically of value $|N_i^r|$. In order to facilitate the directional nature of some interactions, the set of relations $\mathcal{R}$ was expanded to contain a version in both edge directions (incoming and outgoing), by means of both canonical and inverse variations of each relation.

3 LATENT GRAPHS

3.1 Pseudo-relations

We can extend the multi-relational approach of R-GCNs to the case of edges being represented by a vector $\boldsymbol{w}_{ij}$ containing multiple weights across different edge attributes. These attributes can effectively be regarded as non-binary *pseudo*-relations $\mathcal{R}$. The propagation rule now takes a form akin to Equation 3:

$$\boldsymbol{h}_i^{(l+1)} = \sigma\left(\frac{1}{c_i} \sum_{r \in \mathcal{R}} \left[w_s^r \boldsymbol{h}_i^{(l)} + \sum_{j \in N_i} w_{ij}^r \boldsymbol{h}_j^{(l)} \right] \Big| \boldsymbol{W}_r^{(l)} \right), \tag{6}$$

with c_i now taking the form of:

$$c_i = \sum_{r \in \mathcal{R}} \left(w_s^r + \sum_{j \in N_i} w_{ij}^r \right). \tag{7}$$

Note that $\boldsymbol{W}_s^{(l)}$ from Equation 5 has been absorbed into $\boldsymbol{W}_r^{(l)}$. The weight of self-connections can now be learned by means of a self-weight vector $\boldsymbol{w}_s$. In essence, this method obtains an updated node representation for every pseudo-relation separately, after which a weighted summation takes place in order to arrive at the final embedding. The original publication by Schlichtkrull et al. [10] alludes to the possibility of replacing the weighted sum with a more elaborate learning mechanism, such as a fully connected layer, but this was left for future work.

icon solutions

Machine Learning and Fraud Detection

- Payments
- Demonstration
- Thoughts

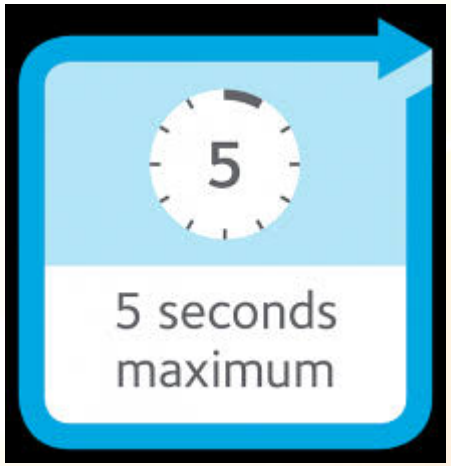
Machine Learning and Fraud Detection

- Payments
- Demonstration
- Thoughts

The increasing scale, diversity, and complexity of fraud.

- Vulnerabilities in payments services have increased as the shift to digital and mobile customer platforms accelerates.

The increasing scale, diversity, and complexity of fraud.



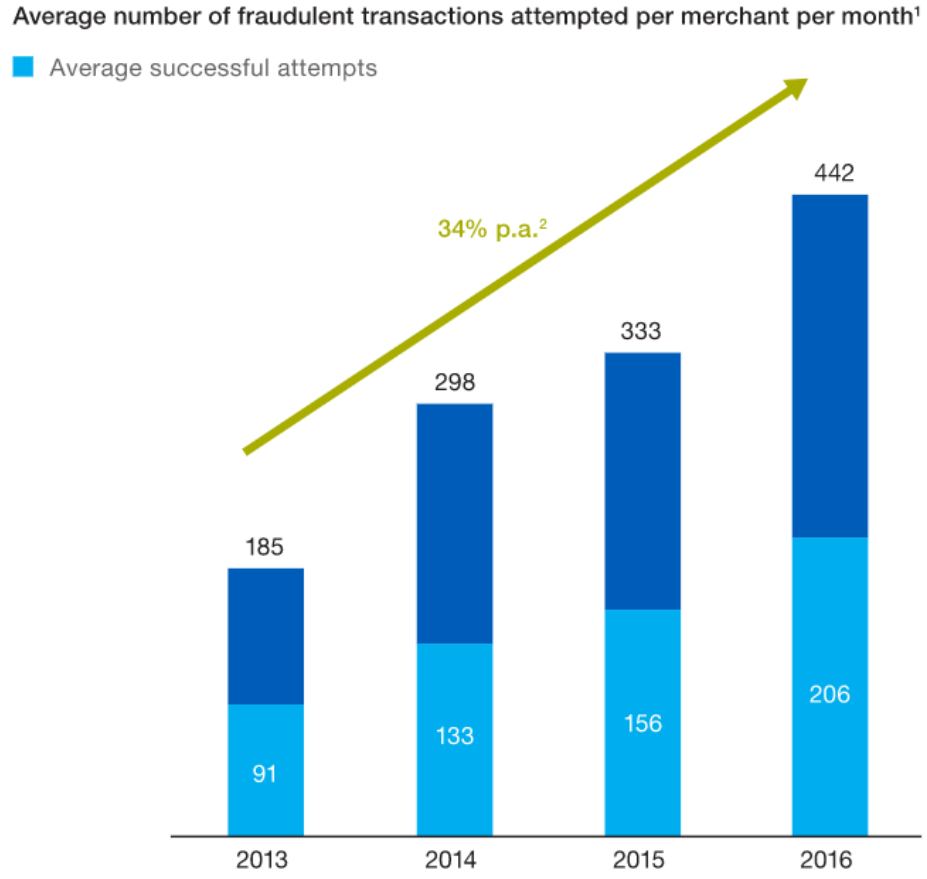
- New solutions have also led to payments transactions being executed more quickly, leaving banks and processors with less time to identify, counteract, and recover the underlying funds when necessary.

The increasing scale, diversity, and complexity of fraud.

- The sophistication of fraud has increased:
  - greater collaboration among bad actors, including:
    the exchange of stolen data,
    new techniques, and
    expertise on the dark web.

The fraud threat facing banks and payments firms has grown dramatically in recent years.

Estimates of fraud's impact on consumers and financial institutions vary significantly but losses to banks alone are conservatively estimated **to exceed $31 billion** globally by 2018.



[1]Weighted merchant responses to LexisNexis survey questions: In a typical month, approximately how many fraudulent transactions are prevented by your company/successfully completed by fraudsters? What is the average value of successful fraud transactions?
[2]Per annum.

McKinsey&Company | **Source:** *2016 LexisNexis True Cost of Fraud Study;* US Department of Commerce

Instant Payments

IPF

Faster Payments

EBA CLEARING

R|T|P
Powering Smarter Payments

stet

EUROPEAN CENTRAL BANK
EUROSYSTEM

icon solutions

Rule Based Systems

Example: if a credit card transaction is more than ten times larger than the average for this customer

Allow the human experts to apply their subject matter expertise.

Difficult and time-consuming to implement well.

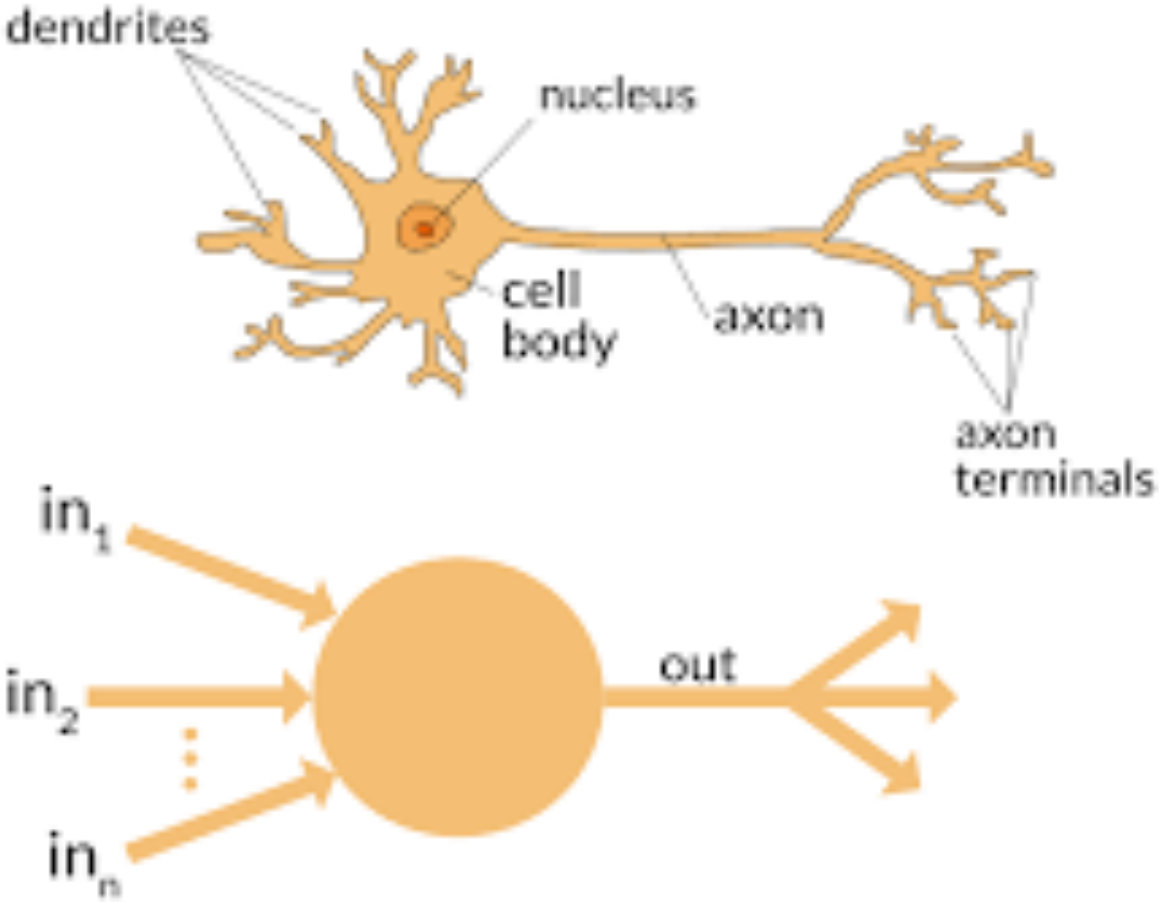
Includes the painstaking definition of every single rule for anomaly possible

If experts make an omission, undetected anomalies will happen and nobody will suspect it.

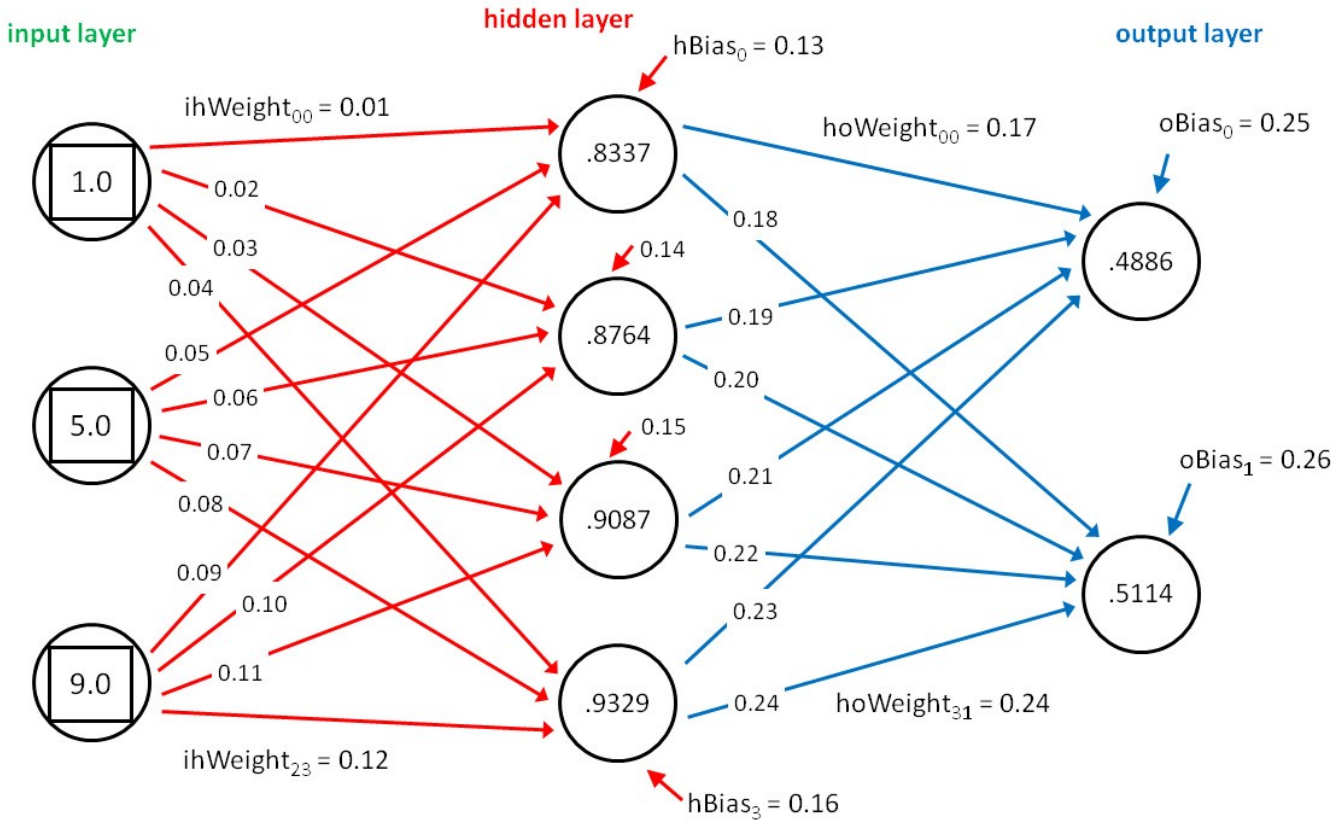
Today, legacy systems apply about 300 different rules on average to approve a transaction

icon solutions

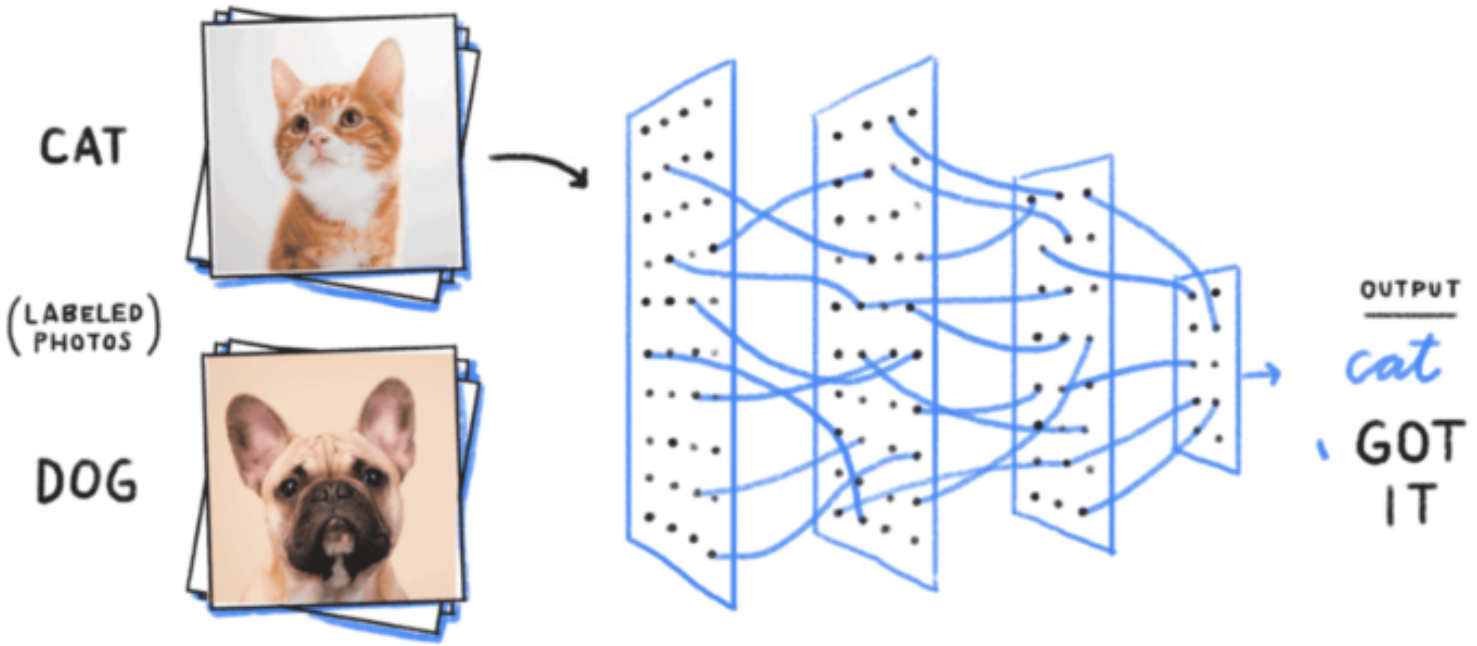
Neural Network



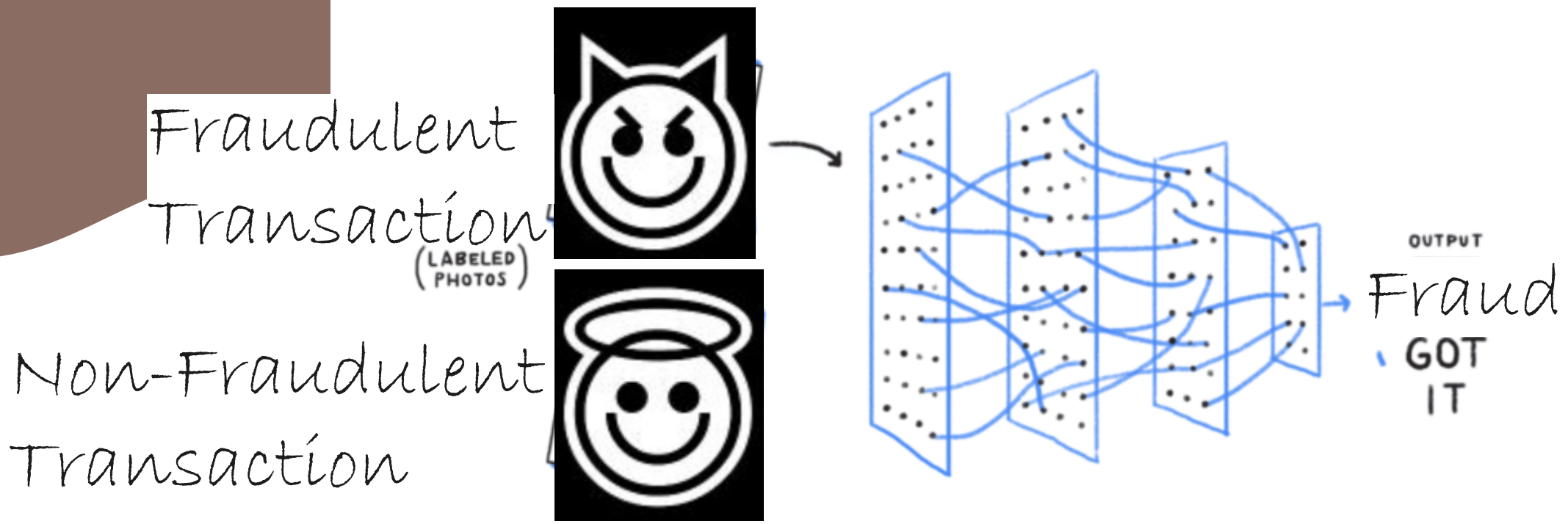
Weights and Biases



Training



Training


Fraudulent Transaction
(LABELED PHOTOS)
Non-Fraudulent Transaction
OUTPUT
Fraud
GOT IT

Use Case

CONTEXT

12

Data: Bank's **12**-months control environment

10000000

Number of bank transactions: **+10 million** unique payments

32%

Previous rule-based control environment could analyze only 32% of unique payments

100%

NetGuardians' machine learning risk platform 100% - Analysis of all unique payments

RESULTS

83%

83% reduction in the number of false positives compared to the previous rule-based control environment

93%

93% reduction in fraud investigation time

118%

118% all fraud cases caught by the previous rule-based control environment also detected by NetGuardians. Plus, new fraud cases are discovered.

NetGuardians

icon solutions

Rule Based versus Machine Learning

Rule Based	Machine Learning
Catches obvious fraudulent scenarios	Finds hidden correlations in data
Large amount of manual work to enumerate all possible detection rules	Automatic detection of possible fraud scenarios
Easier to explain	More difficult to explain

Machine Learning and Fraud Detection

- Payments
- Demonstration
- Thoughts

Install Tensorflow

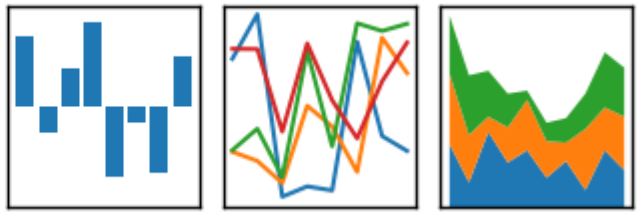
```
tamsincrossland@A318 MINGW64 ~/Documents/AI/creditcardfraud
$ winpty docker run -it --rm -v /${PWD}:/tf/tensorflow-tutorials/wkDir  -p 8888:8888 tensorflow/tensorflow:latest-py3-jupyter
```

icon solutions

Install Libraries

pandas

$y_{it} = \beta' x_{it} + \mu_i + \epsilon_{it}$



Data Analysis

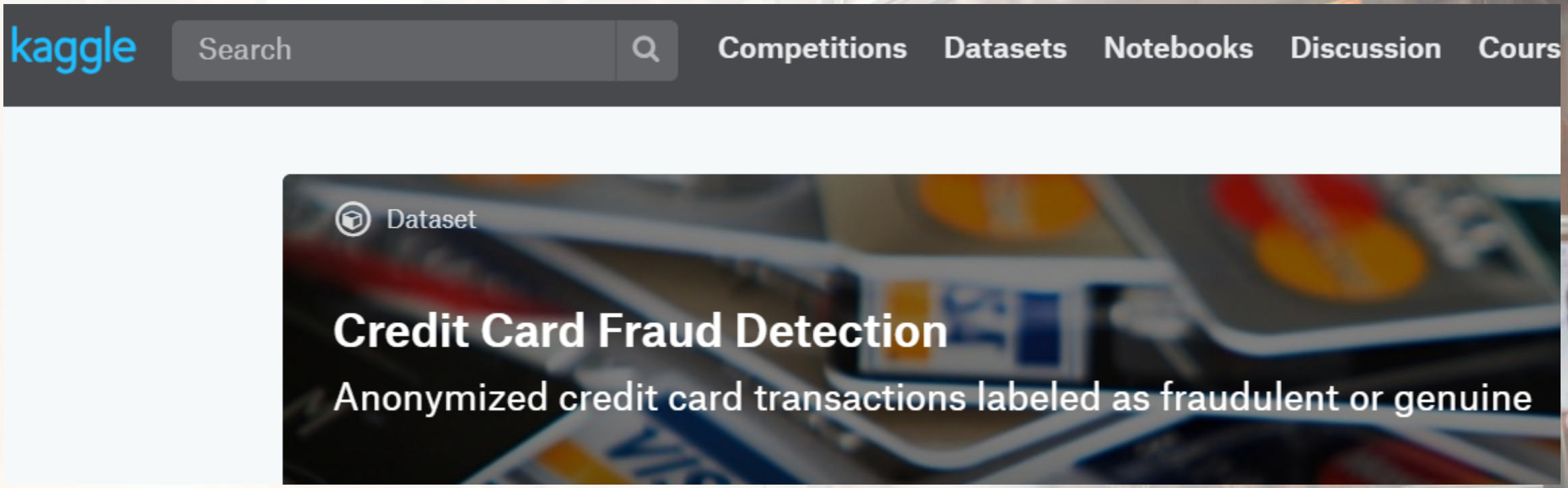
data mining and data analysis

winpty docker exec -i -t 07a24f61e7b6 bash

pip install pandas

pip install -U scikit-learn

icon solutions



Contains two days worth of credit card transactions made in September 2013 by European cardholders.

492 frauds out of 284,807 transactions (0.172%).

Contains only numerical input variables which are the result of a Principal Component Analysis transformation (a method of extracting relevant information from confusing data sets).

Due to confidentiality issues, cannot provide the original features and more background information about the data.

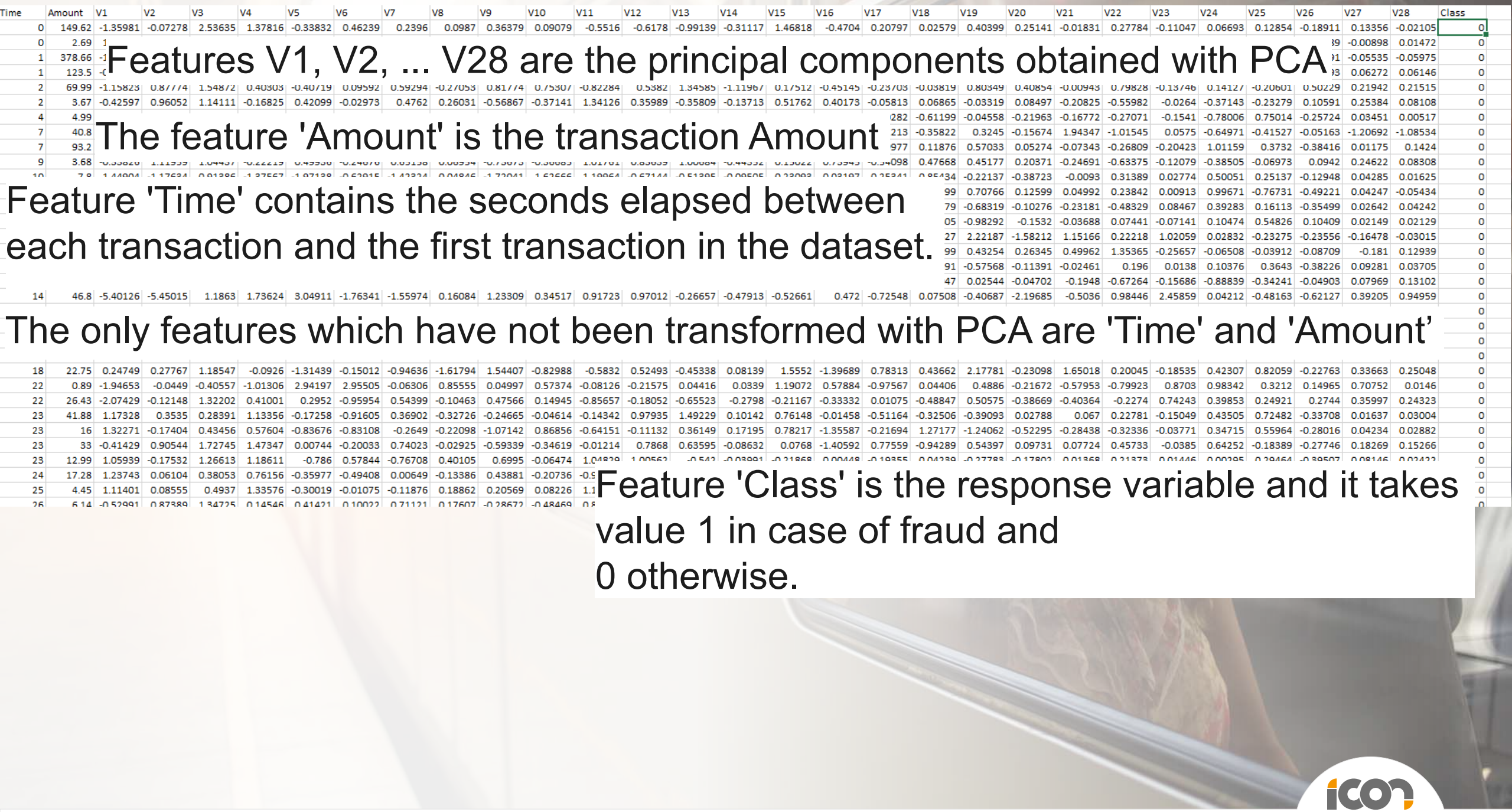
Features V1, V2, ... V28 are the principal components obtained with PCA

The feature 'Amount' is the transaction Amount

Feature 'Time' contains the seconds elapsed between each transaction and the first transaction in the dataset.

The only features which have not been transformed with PCA are 'Time' and 'Amount'

Feature 'Class' is the response variable and it takes value 1 in case of fraud and
0 otherwise.



Demonstration 1

People with no idea about AI saying it will take over the world:

My Neural Network:

Balance data

```
df = df.sample(frac=1)

# amount of fraud classes 492 rows.
fraud_df = df.loc[df['Class'] == 1]
non_fraud_df = df.loc[df['Class'] == 0][:492]

normal_distributed_df = pd.concat([fraud_df, non_fraud_df])

# Shuffle dataframe rows
new_df = normal_distributed_df.sample(frac=1, random_state=42)
```

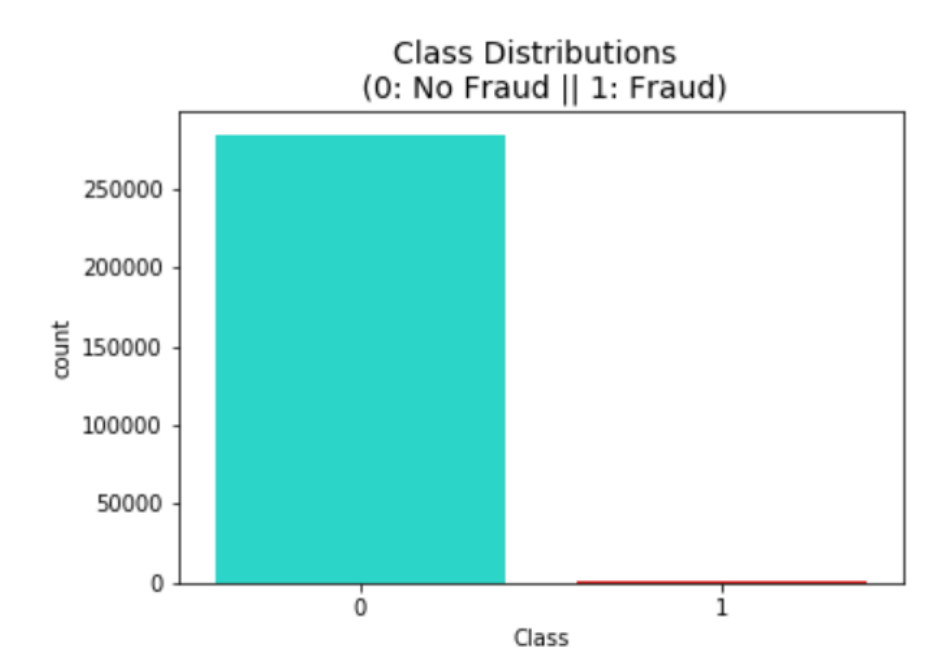
49%

```
# new X_test contains all the transaction not in X_train.
new_X_test = new_df.loc[~new_df.index.isin(new_X_train.index)]
new_train_data = new_X_train.iloc[:, 0:29]
new_train_labels = new_X_train.iloc[:, 30]
new_model = tf.keras.models.Sequential([
  tf.keras.layers.Flatten(input_shape=(29,)),
  tf.keras.layers.Dense(128, activation='relu'),
  tf.keras.layers.Dropout(0.2),
  tf.keras.layers.Dense(2, activation='softmax')
])
# %% compile model
new_model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])
new_model.fit(new_train_data, new_train_labels, epochs=5)
```

```
Epoch 1/5
787/787 [==============================] - 0s 155us/sample - loss: 3847.0455 - accuracy: 0.4968
Epoch 2/5
787/787 [==============================] - 0s 45us/sample - loss: 2625.2003 - accuracy: 0.4943
Epoch 3/5
787/787 [==============================] - 0s 46us/sample - loss: 2379.6559 - accuracy: 0.4956
Epoch 4/5
787/787 [==============================] - 0s 44us/sample - loss: 2528.0340 - accuracy: 0.4460
Epoch 5/5
787/787 [==============================] - 0s 47us/sample - loss: 2153.2511 - accuracy: 0.4930
```

Data Loss

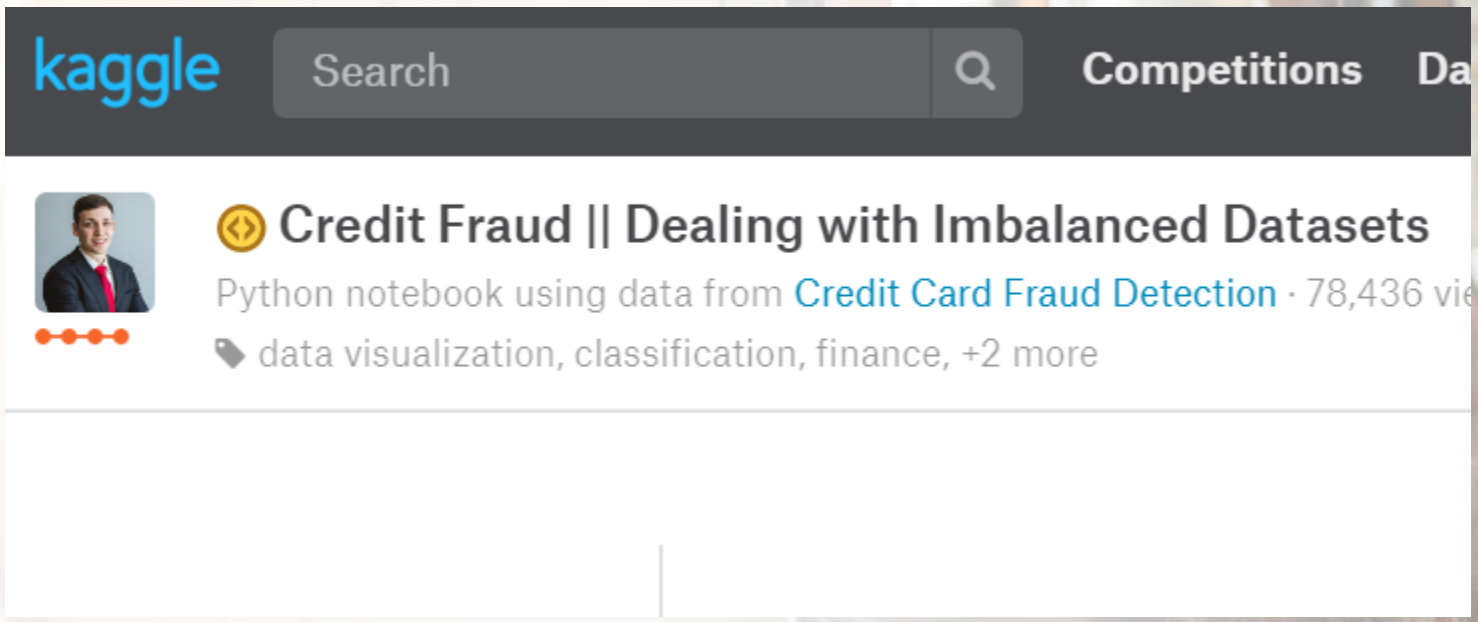
284315 -> 492




THIS IS YOUR MACHINE LEARNING SYSTEM?
YUP! YOU POUR THE DATA INTO THIS BIG PILE OF LINEAR ALGEBRA, THEN COLLECT THE ANSWERS ON THE OTHER SIDE.
WHAT IF THE ANSWERS ARE WRONG?
JUST STIR THE PILE UNTIL THEY START LOOKING RIGHT.
DATA
ANSWERS

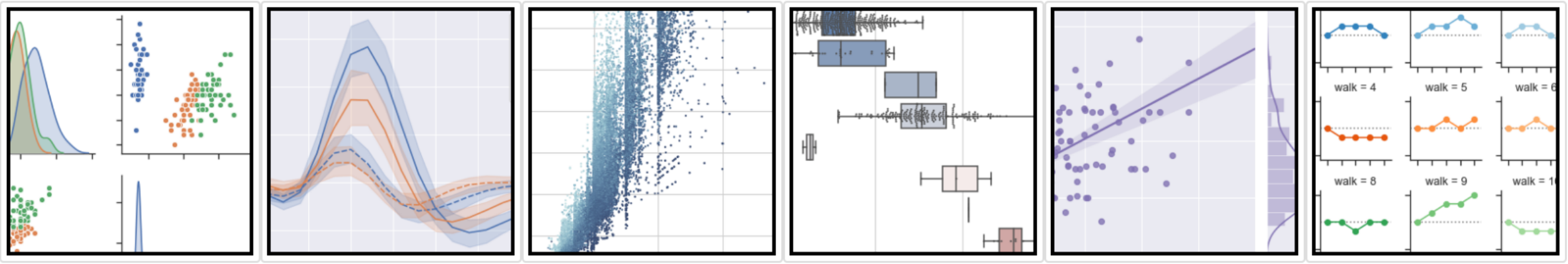
Attempt 3

Janio Martinez Bachmann



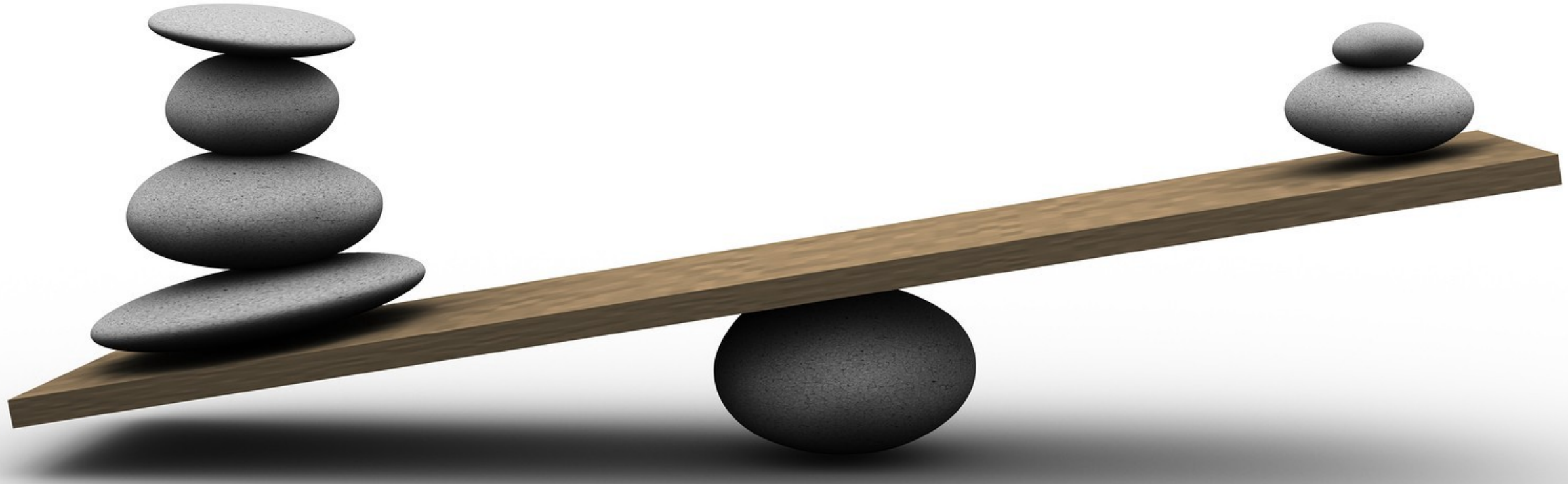
Libraries

seaborn: statistical data visualization

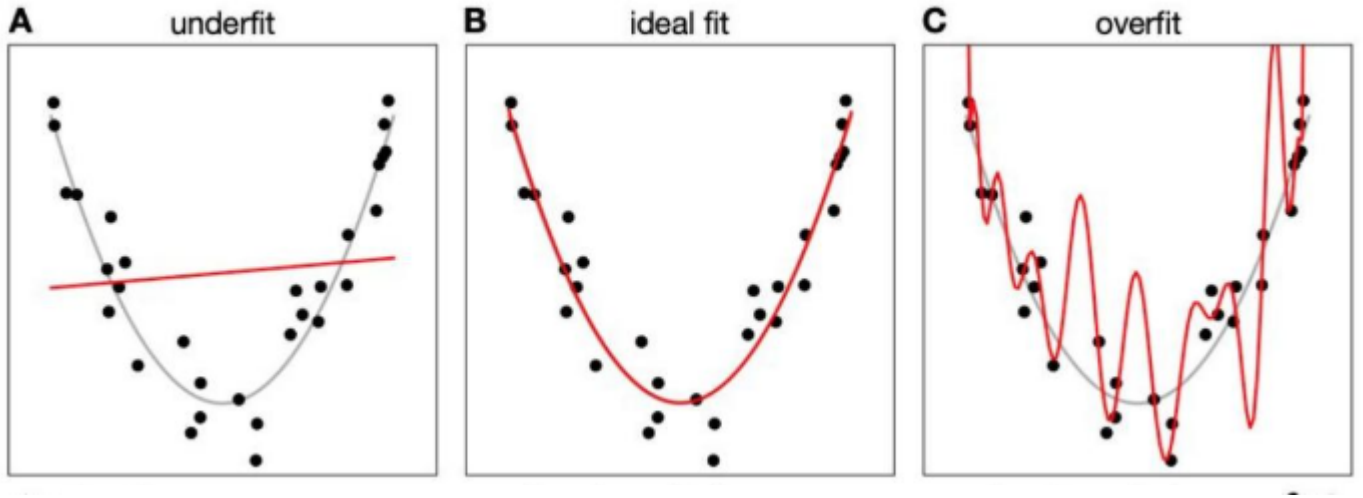


a library for making statistical graphics in Python.

Toolbox for imbalanced dataset in machine learning.

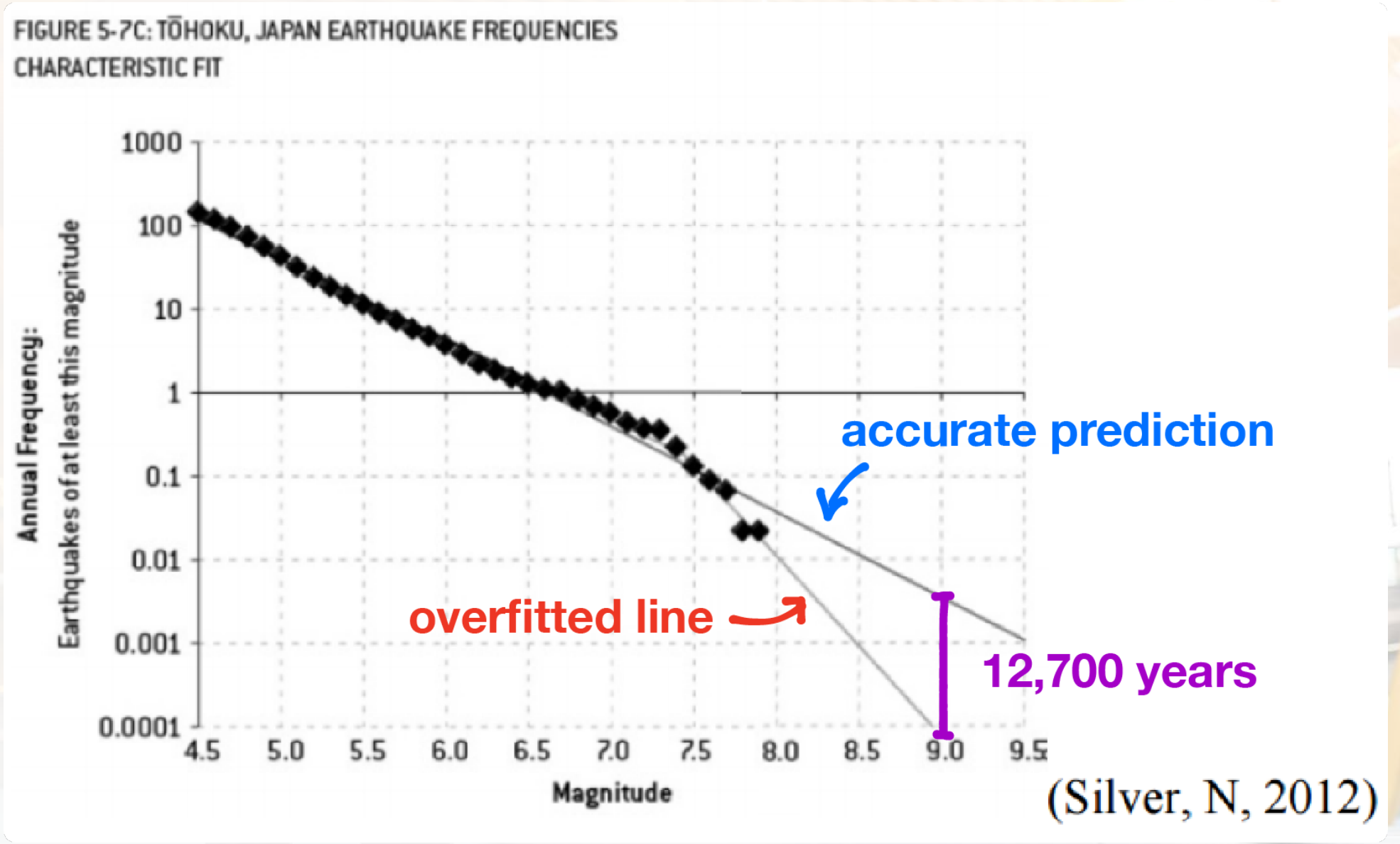


Underfitting and Overfitting

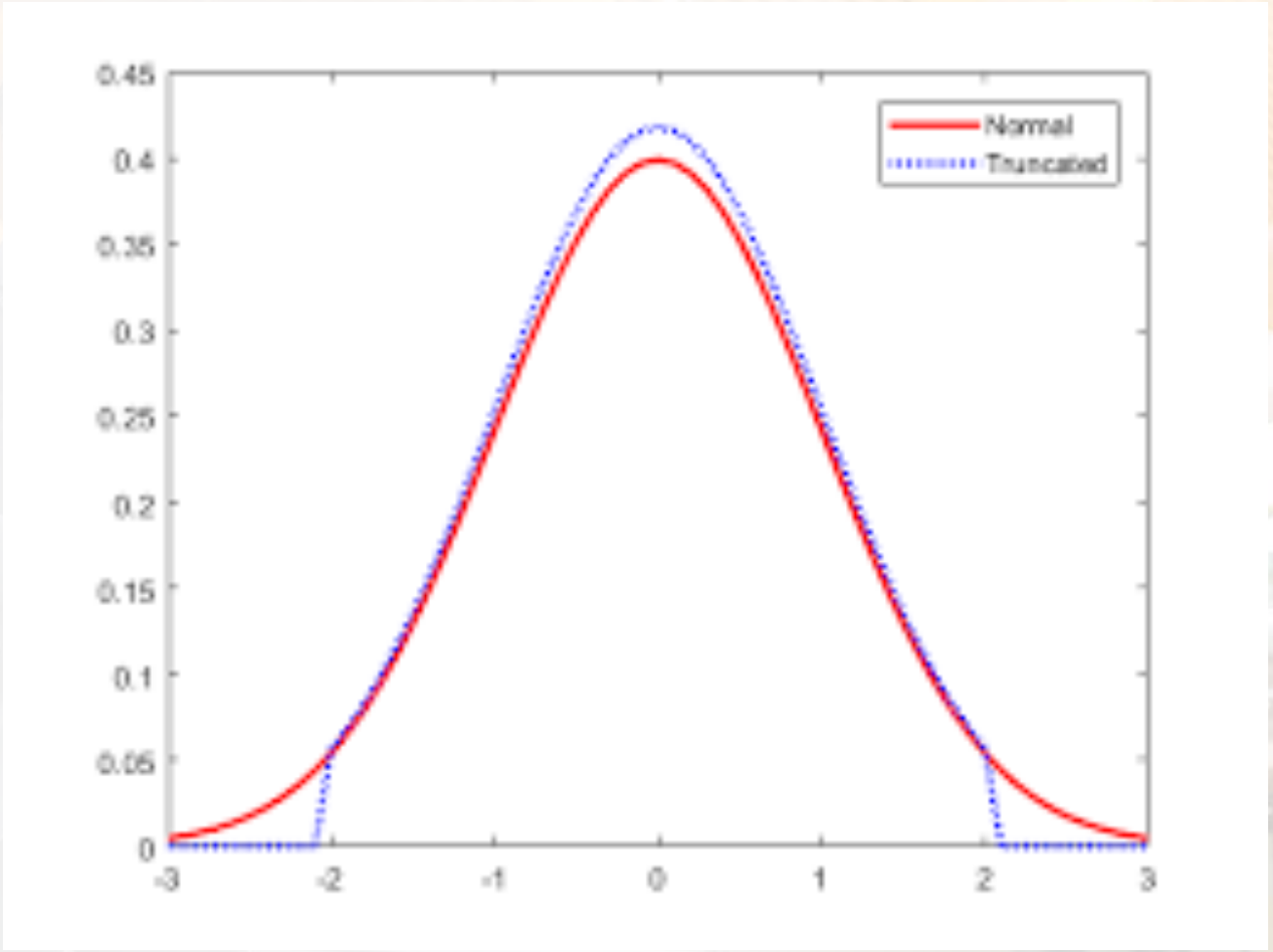


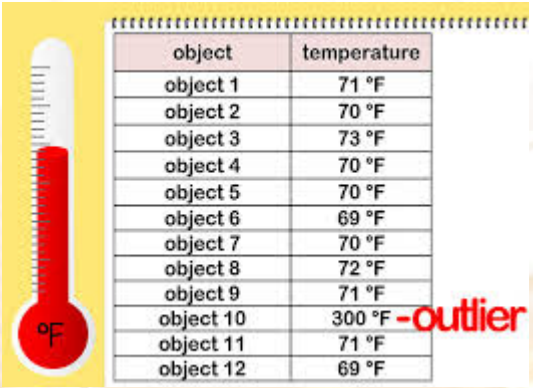
Complexity

Overfitting



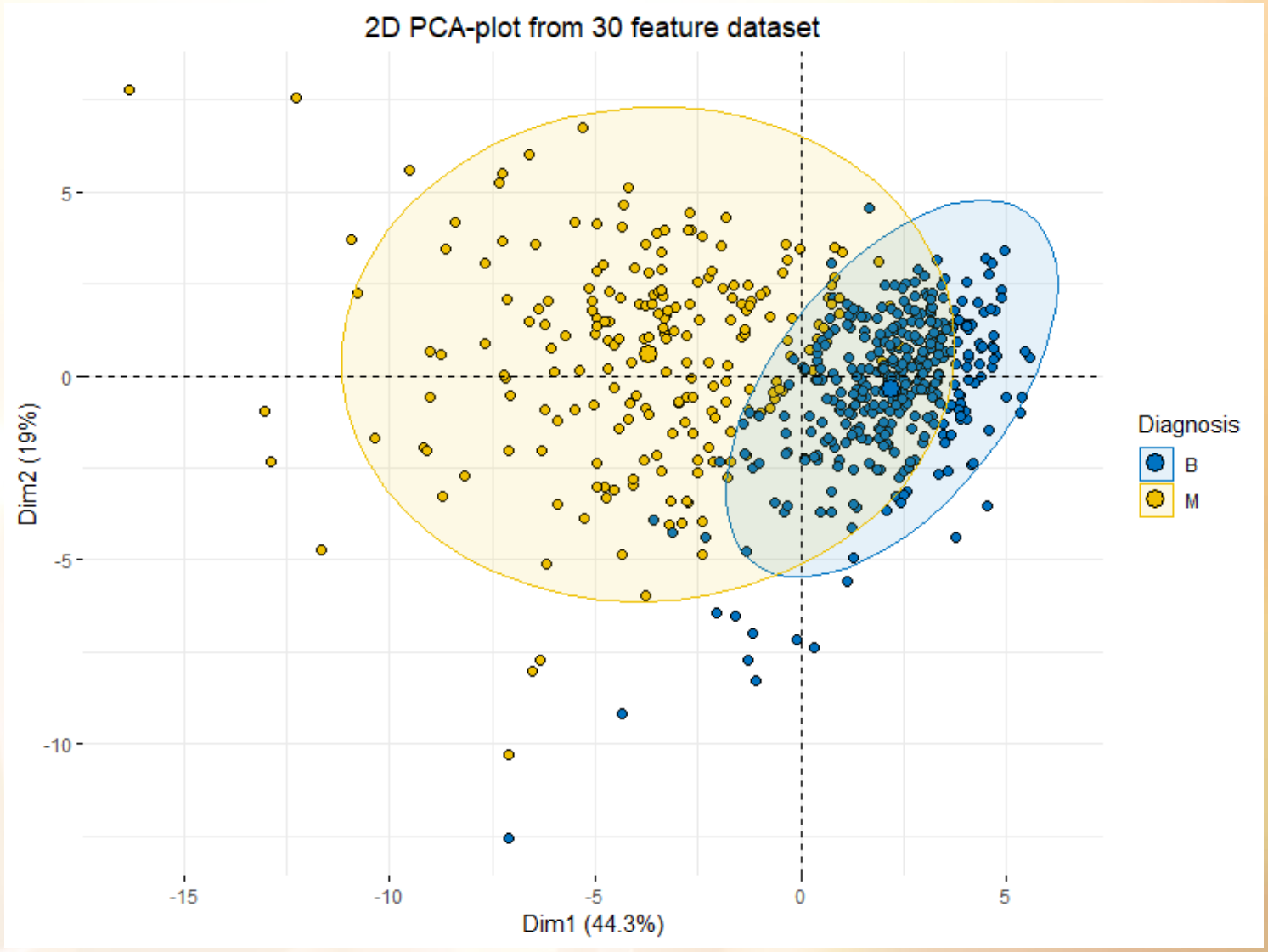
Outliers





object	temperature
object 1	71 °F
object 2	70 °F
object 3	73 °F
object 4	70 °F
object 5	70 °F
object 6	69 °F
object 7	70 °F
object 8	72 °F
object 9	71 °F
object 10	300 °F -outlier
object 11	71 °F
object 12	69 °F

Principal Component Analysis



Demonstration 2

Scale time and amount

```
df = pd.read_csv('creditcard.csv')

df.head()
```

Out[1]:

	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	...	V21	V22	V23	V24	
0	0.0	-1.359807	-0.072781	2.536347	1.378155	-0.338321	0.462388	0.239599	0.098698	0.363787	...	-0.018307	0.277838	-0.110474	0.066928	0.128
1	0.0	1.191857	0.266151	0.166480	0.448154	0.060018	-0.082361	-0.078803	0.085102	-0.255425	...	-0.225775	-0.638672	0.101288	-0.339846	0.167
2	1.0	-1.358354	-1.340163	1.773209	0.379780	-0.503198	1.800499	0.791461	0.247676	-1.514654	...	0.247998	0.771679	0.909412	-0.689281	-0.327
3	1.0	-0.966272	-0.185226	1.792993	-0.863291	-0.010309	1.247203	0.237609	0.377436	-1.387024	...	-0.108300	0.005274	-0.190321	-1.175575	0.647
4	2.0	-1.158233	0.877737	1.548718	0.403034	-0.407193	0.095921	0.592941	-0.270533	0.817739	...	-0.009431	0.798278	-0.137458	0.141267	-0.206

Scaling

Time and **Amount** should be scaled as the other columns.

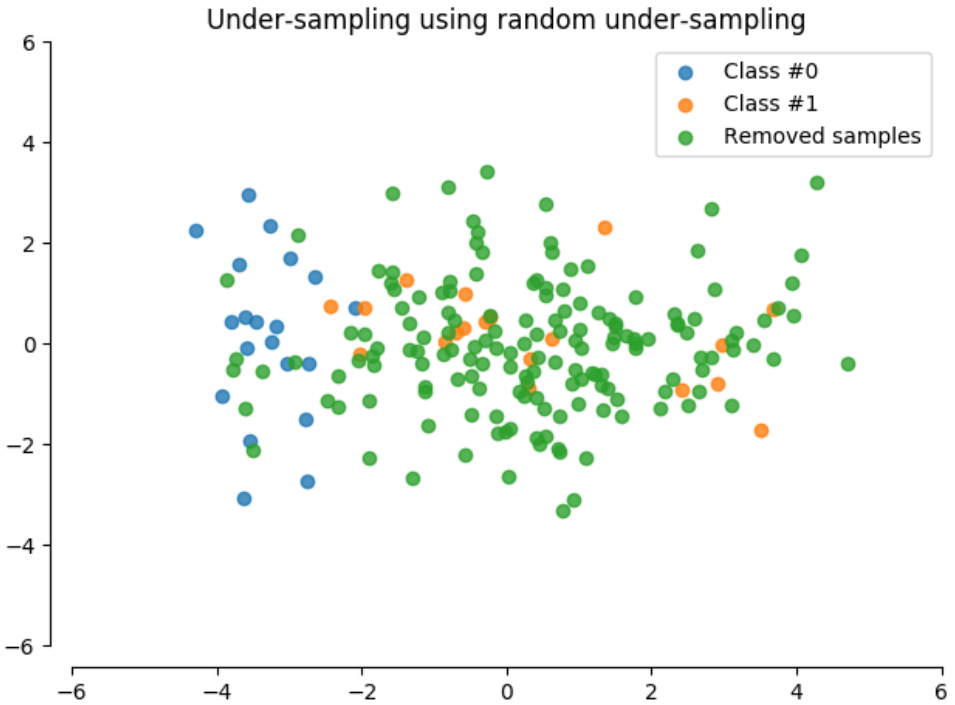
In [2]:

```
# Since most of our data has already been scaled we should scale Amount and Time to be between -1 and 1 too
from sklearn.preprocessing import RobustScaler

rob_scaler = RobustScaler()

df['scaled_amount'] = rob_scaler.fit_transform(df['Amount'].values.reshape(-1,1))
df['scaled_time'] = rob_scaler.fit_transform(df['Time'].values.reshape(-1,1))

df.drop(['Time','Amount'], axis=1, inplace=True)
scaled_amount = df['scaled_amount']
scaled_time = df['scaled_time']

df.drop(['scaled_amount', 'scaled_time'], axis=1, inplace=True)
df.insert(0, 'scaled_amount', scaled_amount)
df.insert(1, 'scaled_time', scaled_time)

# Amount and Time are Scaled!

df.head()
```

Out[2]:

	scaled_amount	scaled_time	V1	V2	V3	V4	V5	V6	V7	V8	...	V20	V21	V22	
0	1.783274	-0.994983	-1.359807	-0.072781	2.536347	1.378155	-0.338321	0.462388	0.239599	0.098698	...	0.251412	-0.018307	0.277838	-0.11
1	-0.269825	-0.994983	1.191857	0.266151	0.166480	0.448154	0.060018	-0.082361	-0.078803	0.085102	...	-0.069083	-0.225775	-0.638672	0.10

Random under-sampling



In this phase of the project we will implement *"Random Under Sampling"* which basically consists of removing data in order to have a more **balanced dataset** and thus avoiding our models overfitting.

```
df = df.sample(frac=1)

# amount of fraud classes 492 rows.
fraud_df = df.loc[df['Class'] == 1]
non_fraud_df = df.loc[df['Class'] == 0][:492]

normal_distributed_df = pd.concat([fraud_df, non_fraud_df])

# Shuffle dataframe rows
new_df = normal_distributed_df.sample(frac=1, random_state=42)

colors = ["#0101DF", "#DF0101"]

sns.countplot('Class', data=new_df, palette=colors)
plt.title('Class Distributions \n (0: No Fraud || 1: Fraud)', fontsize=14)
```

Out[3]: Text(0.5, 1.0, 'Class Distributions \n (0: No Fraud || 1: Fraud)')

Correlation Matrix

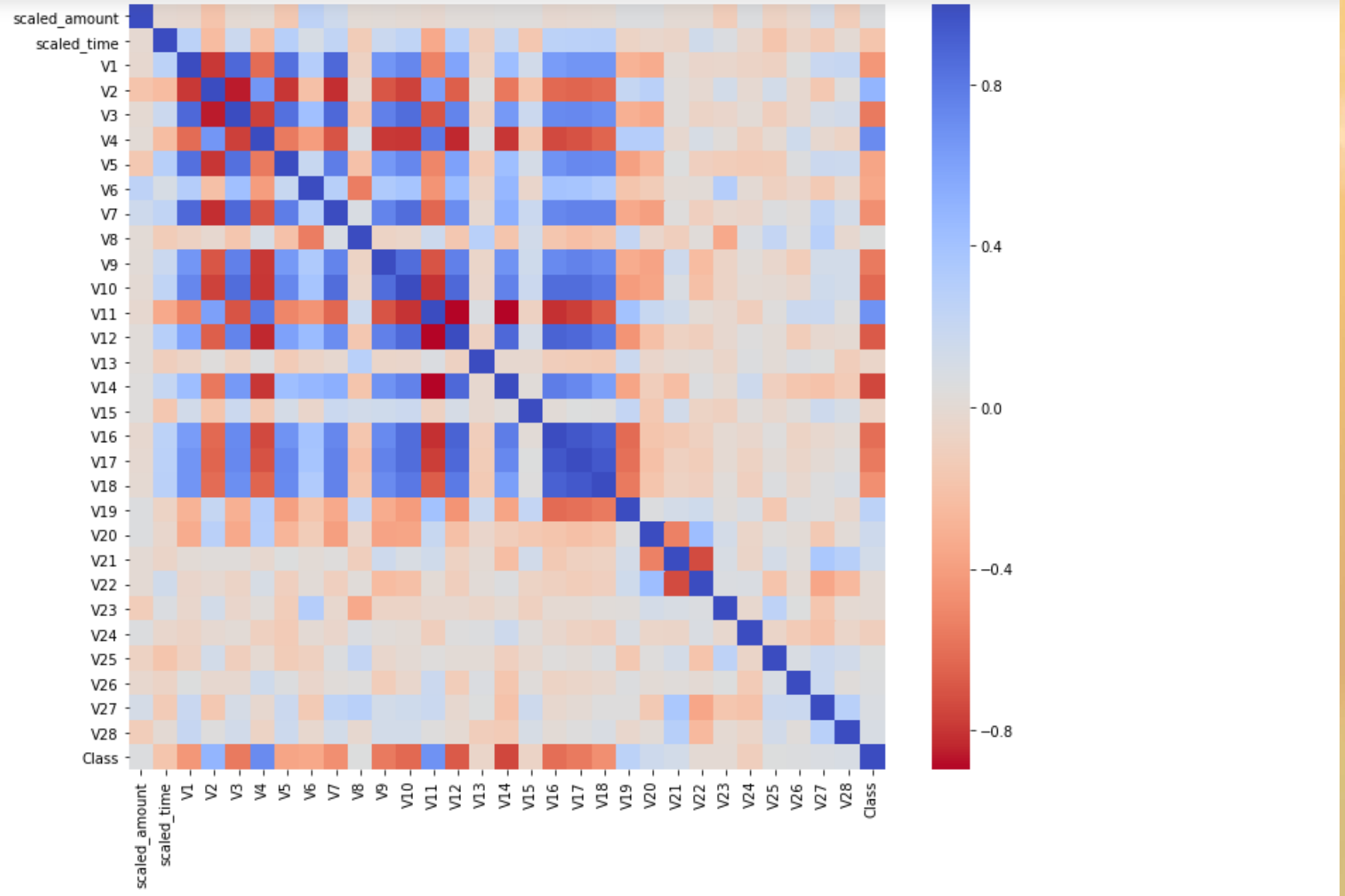
Used to show which features heavily influence whether a transaction is a fraud

Now that we have our dataframe correctly balanced, we can go further with our **analysis** and **data preprocessing**.

Correlation Matrices

A correlation matrix is used to show which features heavily influence whether a specific transaction is a fraud.

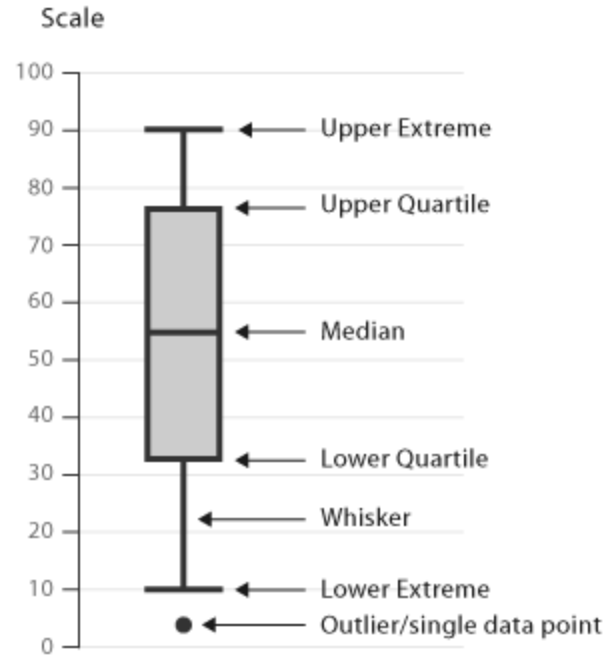
```
# Make sure we use the subsample in our correlation

f, (ax1) = plt.subplots(figsize=(12,10))

sub_sample_corr = new_df.corr()
sns.heatmap(sub_sample_corr, cmap='coolwarm_r', annot_kws={'size':20}, ax=ax1)
ax1.set_title('SubSample Correlation Matrix \n', fontsize=14)
plt.show()
```



Summary and Explanation:

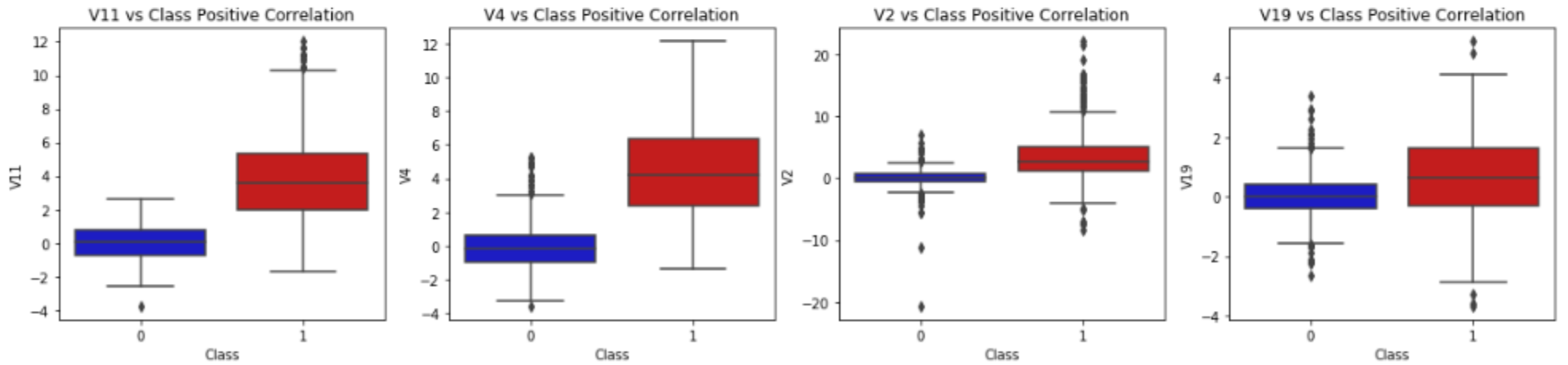
- **Positive Correlations:** V2, V4, V11, and V19 are positively correlated. The higher these values are, the more likely the end result will be a fraud transaction.
- **Negative Correlations:** V17, V14, V12 and V10 are negatively correlated. The lower these values are, the more likely the end result will be a fraud transaction.

Anomaly detection



Our main aim in this section is to remove "extreme outliers" from features that have a high correlation with our classes. This will have a positive impact on the accuracy of our models.

```
f, axes = plt.subplots(ncols=4, figsize=(20,4))

# Positive correlations (The higher the feature the probability increases that it will be a fraud transaction)
sns.boxplot(x="Class", y="V11", data=new_df, palette=colors, ax=axes[0])
axes[0].set_title('V11 vs Class Positive Correlation')

sns.boxplot(x="Class", y="V4", data=new_df, palette=colors, ax=axes[1])
axes[1].set_title('V4 vs Class Positive Correlation')


sns.boxplot(x="Class", y="V2", data=new_df, palette=colors, ax=axes[2])
axes[2].set_title('V2 vs Class Positive Correlation')


sns.boxplot(x="Class", y="V19", data=new_df, palette=colors, ax=axes[3])
axes[3].set_title('V19 vs Class Positive Correlation')

plt.show()
```



```python
# Removing outliers V10 Feature
v10_fraud = new_df['V10'].loc[new_df['Class'] == 1].values
q25, q75 = np.percentile(v10_fraud, 25), np.percentile(v10_fraud, 75)
v10_iqr = q75 - q25

v10_cut_off = v10_iqr * 1.5
v10_lower, v10_upper = q25 - v10_cut_off, q75 + v10_cut_off
print('V10 Lower: {}'.format(v10_lower))
print('V10 Upper: {}'.format(v10_upper))
outliers = [x for x in v10_fraud if x < v10_lower or x > v10_upper]
print('V10 outliers: {}'.format(outliers))
print('Feature V10 Outliers for Fraud Cases: {}'.format(len(outliers)))
new_df = new_df.drop(new_df[(new_df['V10'] > v10_upper) | (new_df['V10'] < v10_lower)].index)
print('Number of Instances after outliers removal: {}'.format(len(new_df)))
```

After implementing outlier reduction our accuracy has been improved by **over 3%**! Some outliers can distort the accuracy of our models but remember, we have to avoid an extreme amount of information loss or else our model runs the risk of underfitting.

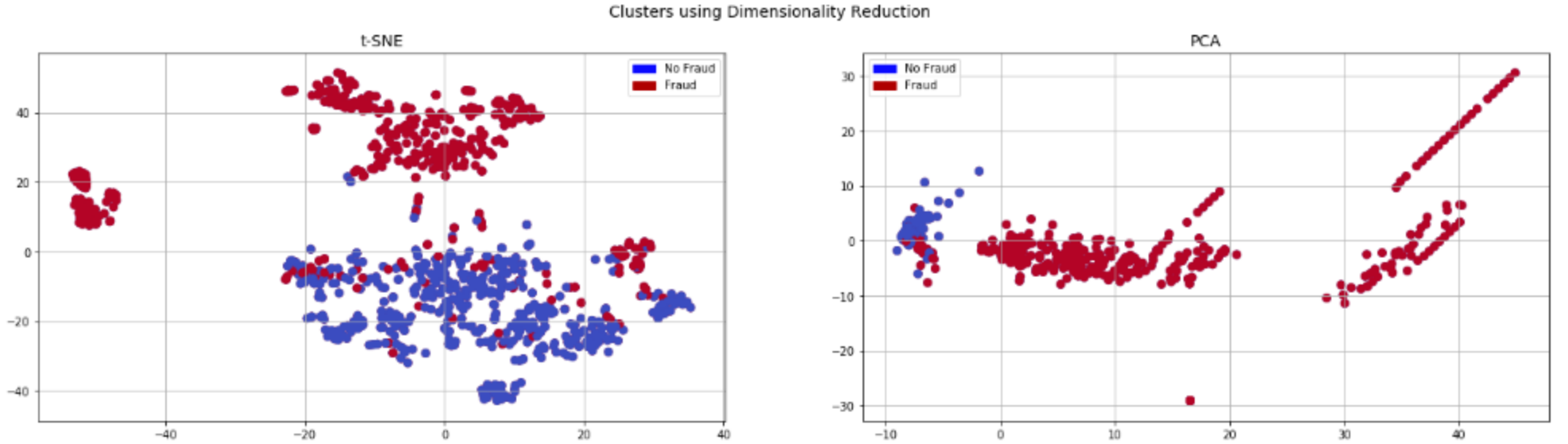
```
V10 Lower: -14.89885463232024
V10 Upper: 4.920334958342141
V10 outliers: [-20.949191554361104, -15.346098846877501, -17.141513641289198, -14.9246547735487, -16.7460441053944, -15.2318333
653018, -18.9132433348732, -15.124162814494698, -15.2399619587112, -16.6496281595399, -15.563791338730098, -15.563791338730098,
-23.2282548357516, -19.836148851696, -15.2399619587112, -15.1237521803455, -22.1870885620007, -16.6011969664137, -18.2711681738
888, -16.2556117491401, -22.1870885620007, -24.5882624372475, -16.3035376590131, -24.403184969972802, -22.1870885620007, -14.92
46547735487, -22.1870885620007]
Feature V10 Outliers for Fraud Cases: 27
Number of Instances after outliers removal: 945
```

Dimensionality Reduction and Clustering

of the information.

```
# New_df is from the random undersample data (fewer instances)
X = new_df.drop('Class', axis=1)
y = new_df['Class']


# T-SNE Implementation
t0 = time.time()
X_reduced_tsne = TSNE(n_components=2, random_state=42).fit_transform(X.values)
t1 = time.time()

# PCA Implementation
t0 = time.time()
X_reduced_pca = PCA(n_components=2, random_state=42).fit_transform(X.values)
t1 = time.time()

f, (ax1, ax2) = plt.subplots(1, 2, figsize=(24,6))
# labels = ['No Fraud', 'Fraud']
f.suptitle('Clusters using Dimensionality Reduction', fontsize=14)


blue_patch = mpatches.Patch(color='#0A0AFF', label='No Fraud')
red_patch = mpatches.Patch(color='#AF0000', label='Fraud')


# t-SNE scatter plot
ax1.scatter(X_reduced_tsne[:,0], X_reduced_tsne[:,1], c=(y == 0), cmap='coolwarm', label='No Fraud', linewidths=2)
ax1.scatter(X_reduced_tsne[:,0], X_reduced_tsne[:,1], c=(y == 1), cmap='coolwarm', label='Fraud', linewidths=2)
ax1.set_title('t-SNE', fontsize=14)

ax1.grid(True)

ax1.legend(handles=[blue_patch, red_patch])


# PCA scatter plot
ax2.scatter(X_reduced_pca[:,0], X_reduced_pca[:,1], c=(y == 0), cmap='coolwarm', label='No Fraud', linewidths=2)
ax2.scatter(X_reduced_pca[:,0], X_reduced_pca[:,1], c=(y == 1), cmap='coolwarm', label='Fraud', linewidths=2)
ax2.set_title('PCA', fontsize=14)

ax2.grid(True)

ax2.legend(handles=[blue_patch, red_patch])

plt.show()
```

Dimensionality Reduction and Clustering



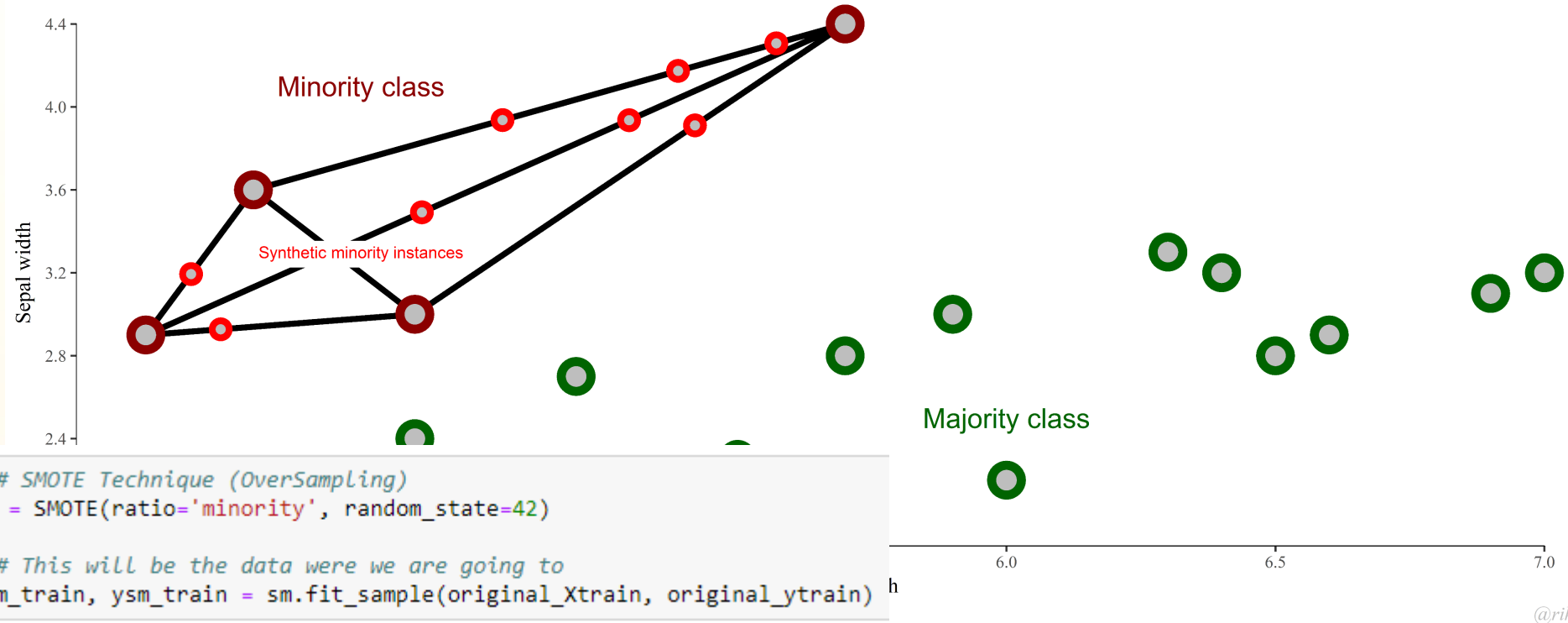
Summary:

- t-SNE algorithm can pretty accurately cluster the cases that were fraud and non-fraud in our dataset.
- Although the subsample is pretty small, the t-SNE algorithm is able to detect clusters pretty accurately in every scenario (I shuffle the dataset before running t-SNE)
- This gives us an indication that further predictive models will perform pretty well in separating fraud cases from non-fraud cases.

t-SNE takes a high-dimensional dataset and reduces it to a low-dimensional graph whilst still retaining a lot of the information.

SMOTE

Addressing class imbalance problems of ML via SMOTE: synthesising new dots between existing dots




```
# # SMOTE Technique (OverSampling)
sm = SMOTE(ratio='minority', random_state=42)

# # This will be the data were we are going to
Xsm_train, ysm_train = sm.fit_sample(original_Xtrain, original_ytrain)
```

Synthetic Minority Over-sampling Technique

Solving the Class Imbalance: SMOTE creates synthetic points from the minority class in order to reach an equal balance between the minority and majority class.

Location of the synthetic points: SMOTE picks the distance between the closest neighbors of the minority class, in between these distances it creates synthetic points.

Final Effect: More information is retained since we didn't have to delete any rows unlike in random undersampling.

icon solutions

Compile the model

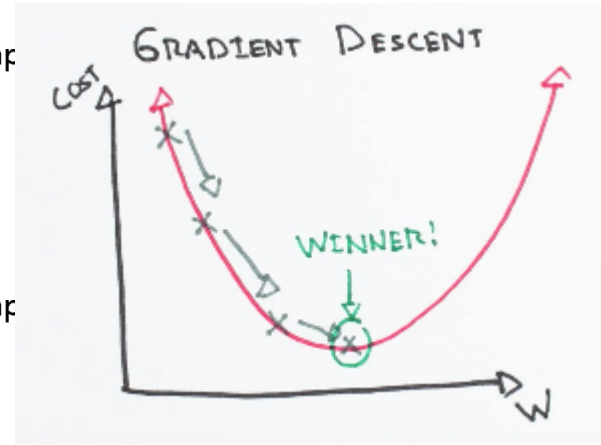
The following example uses *accuracy*, the fraction of the transactions that are correctly classified.

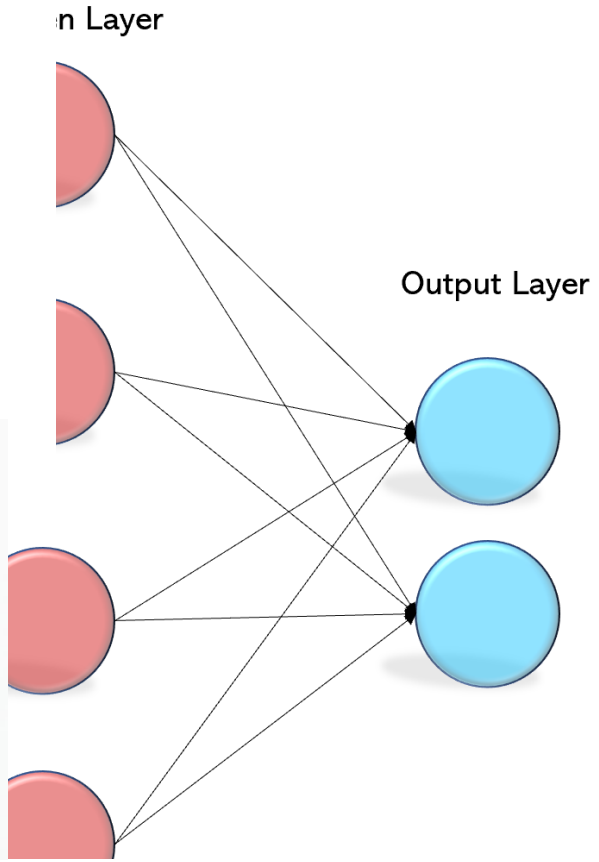
optimizers shape and mold your model into its most accurate possible form by futzing with the weights. The **loss** function is the guide to the terrain, telling the optimizer when it's moving in the right or wrong direction.

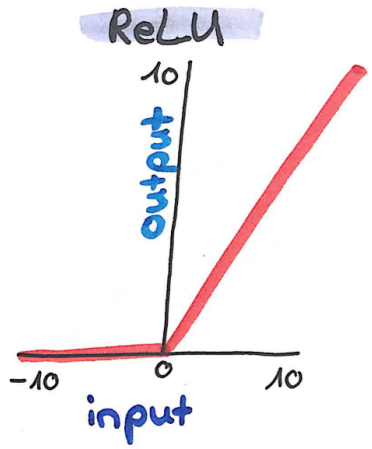
```
# import keras
from tensorflow.keras import backend as K
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Activation
from tensorflow.keras.layers import Dense
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.metrics import categorical_crossentropy

n_inputs = X_train.shape[1]

undersample_model = Sequential([
    Dense(n_inputs, input_shape=(n_inputs, ), activation='relu'),
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense (Dense)                (None, 30)                930
_________________________________________________________________
dense_1 (Dense)              (None, 32)                992
_________________________________________________________________
dense_2 (Dense)              (None, 2)                 66
=================================================================
Total params: 1,988
Trainable params: 1,988
```

```
ossentropy', metrics=['accuracy'])

ize=25, epochs=20, shuffle=True, verbose=2)

atch_size=200, verbose=0)

undersample_fraud_predictions = undersample_model.predict_classes(original_Xtest, batch_size=200, verbose=0)
```







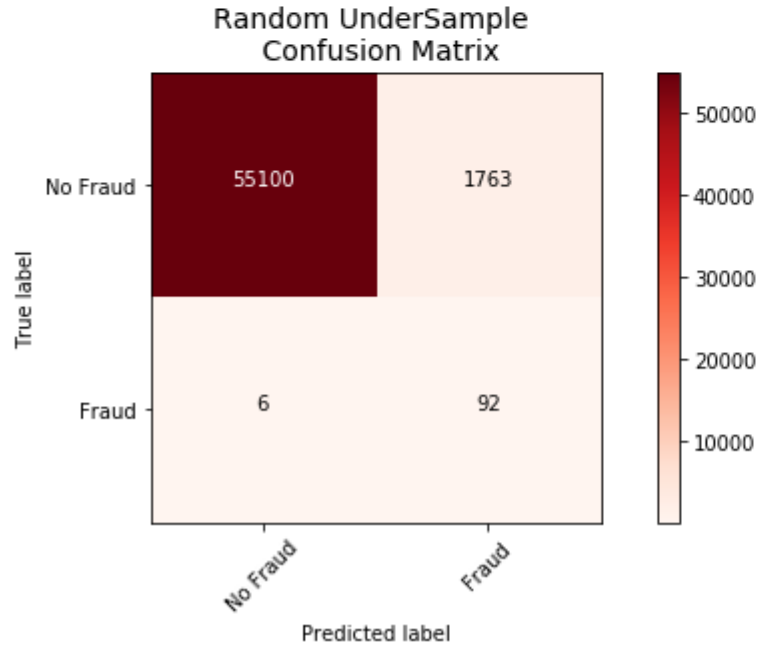
```
Epoch 1/20
604/604 - 0s - loss: 0.7081 - accuracy: 0.6358 - val_loss: 0.4522 - val_accuracy: 0.7947
Epoch 2/20
604/604 - 0s - loss: 0.3993 - accuracy: 0.8046 - val_loss: 0.3239 - val_accuracy: 0.8675
Epoch 3/20
604/604 - 0s - loss: 0.3101 - accuracy: 0.8825 - val_loss: 0.2623 - val_accuracy: 0.9073
Epoch 4/20
604/604 - 0s - loss: 0.2573 - accuracy: 0.9255 - val_loss: 0.2272 - val_accuracy: 0.9139
Epoch 5/20
604/604 - 0s - loss: 0.2201 - accuracy: 0.9387 - val_loss: 0.2035 - val_accuracy: 0.9139
Epoch 6/20
604/604 - 0s - loss: 0.1927 - accuracy: 0.9437 - val_loss: 0.1856 - val_accuracy: 0.9139
Epoch 7/20
604/604 - 0s - loss: 0.1707 - accuracy: 0.9487 - val_loss: 0.1728 - val_accuracy: 0.9205
Epoch 8/20
604/604 - 0s - loss: 0.1549 - accuracy: 0.9503 - val_loss: 0.1638 - val_accuracy: 0.9205
Epoch 9/20
604/604 - 0s - loss: 0.1437 - accuracy: 0.9503 - val_loss: 0.1615 - val_accuracy: 0.9205
Epoch 10/20
604/604 - 0s - loss: 0.1344 - accuracy: 0.9503 - val_loss: 0.1553 - val_accuracy: 0.9139
Epoch 11/20
604/604 - 0s - loss: 0.1258 - accuracy: 0.9553 - val_loss: 0.1522 - val_accuracy: 0.9139
Epoch 12/20
604/604 - 0s - loss: 0.1209 - accuracy: 0.9487 - val_loss: 0.1488 - val_accuracy: 0.9139
Epoch 13/20
604/604 - 0s - loss: 0.1135 - accuracy: 0.9570 - val_loss: 0.1470 - val_accuracy: 0.9139
Epoch 14/20
604/604 - 0s - loss: 0.1078 - accuracy: 0.9586 - val_loss: 0.1470 - val_accuracy: 0.9139
Epoch 15/20
604/604 - 0s - loss: 0.1023 - accuracy: 0.9636 - val_loss: 0.1463 - val_accuracy: 0.9139
Epoch 16/20
604/604 - 0s - loss: 0.0981 - accuracy: 0.9636 - val_loss: 0.1463 - val_accuracy: 0.9205
Epoch 17/20
604/604 - 0s - loss: 0.0934 - accuracy: 0.9685 - val_loss: 0.1489 - val_accuracy: 0.9205
Epoch 18/20
604/604 - 0s - loss: 0.0901 - accuracy: 0.9719 - val_loss: 0.1474 - val_accuracy: 0.9205
Epoch 19/20
604/604 - 0s - loss: 0.0854 - accuracy: 0.9685 - val_loss: 0.1511 - val_accuracy: 0.9205
Epoch 20/20
604/604 - 0s - loss: 0.0821 - accuracy: 0.9752 - val_loss: 0.1476 - val_accuracy: 0.9272
```

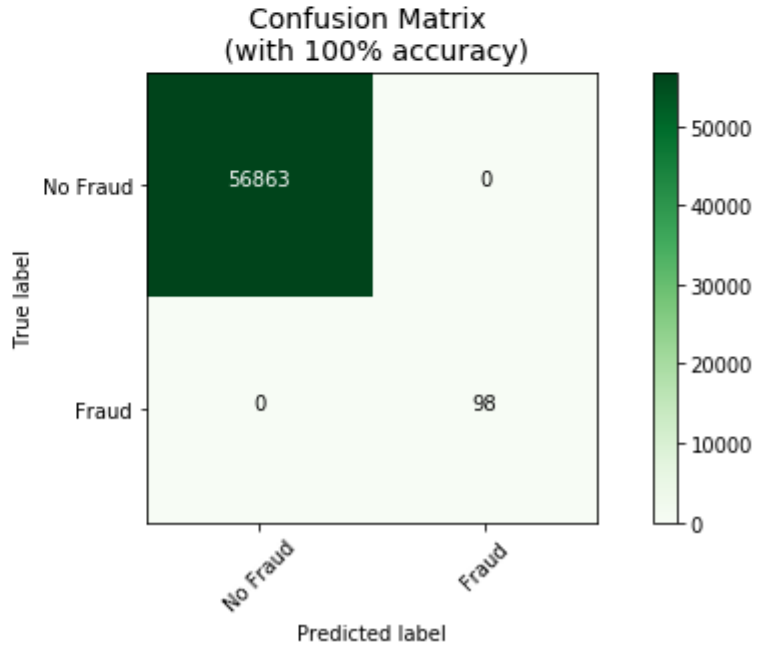
Confusion Matrix

	Predicted: no	Predicted: Yes
Actual: no	True negative	False positive
Actual: yes	False negative	True positive

```
undersample_cm = confusion_matrix(original_ytest, undersample_fraud_predictions)
actual_cm = confusion_matrix(original_ytest, original_ytest)
labels = ['No Fraud', 'Fraud']

fig = plt.figure(figsize=(16,8))

fig.add_subplot(221)
plot_confusion_matrix(undersample_cm, labels, title="Random UnderSample \n Confusion Matrix", cmap=plt.cm.Reds)

fig.add_subplot(222)
plot_confusion_matrix(actual_cm, labels, title="Confusion Matrix \n (with 100% accuracy)", cmap=plt.cm.Greens)
```

	Predicted: no	Predicted: Yes
Actual: no	True negative	False positive
Actual: yes	False negative	True positive





Using SMOTE

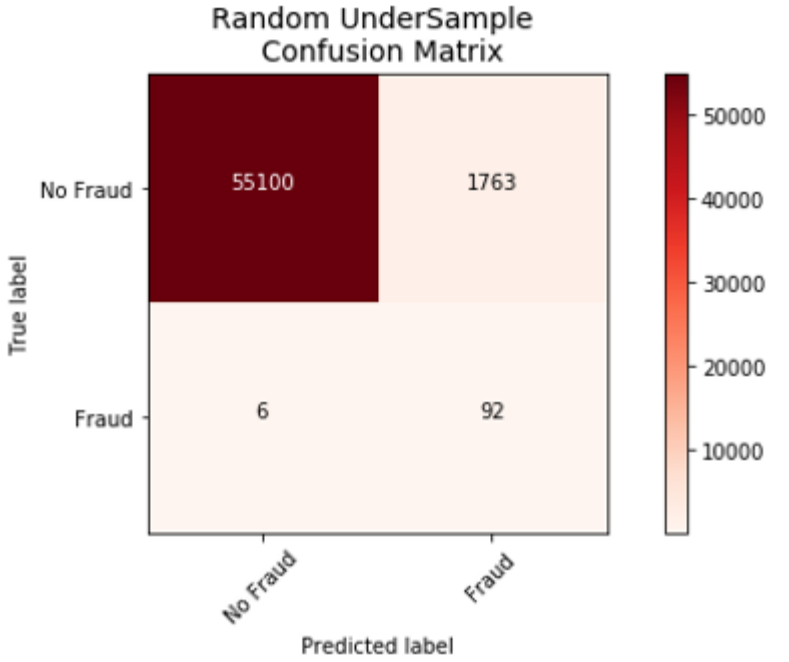
Keras || OverSampling (SMOTE):

```
n_inputs = Xsm_train.shape[1]

oversample_model = Sequential([
    Dense(n_inputs, input_shape=(n_inputs, ), activation='relu'),
    Dense(32, activation='relu'),
    Dense(2, activation='softmax')
])

oversample_model.compile(Adam(lr=0.001), loss='sparse_categorical_crossentropy', metrics=['accuracy'])

oversample_model.fit(Xsm_train, ysm_train, validation_split=0.2, batch_size=300, epochs=20, shuffle=True, verbose=2)

oversample_predictions = oversample_model.predict(original_Xtest, batch_size=200, verbose=0)

oversample_fraud_predictions = oversample_model.predict_classes(original_Xtest, batch_size=200, verbose=0)
```

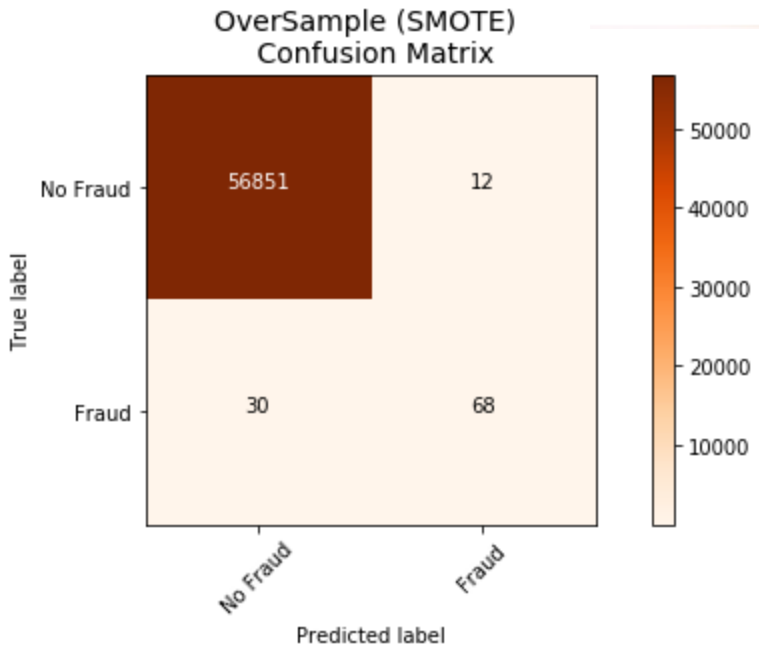
```
Train on 363923 samples, validate on 90981 samples
Epoch 1/20
363923/363923 - 2s - loss: 0.0898 - accuracy: 0.9685 - val_loss: 0.0341 - val_accuracy: 0.9880
Epoch 2/20
363923/363923 - 2s - loss: 0.0187 - accuracy: 0.9953 - val_loss: 0.0118 - val_accuracy: 0.9997
Epoch 3/20
363923/363923 - 2s - loss: 0.0099 - accuracy: 0.9978 - val_loss: 0.0078 - val_accuracy: 0.9996
Epoch 4/20
363923/363923 - 2s - loss: 0.0065 - accuracy: 0.9985 - val_loss: 0.0038 - val_accuracy: 0.9999
Epoch 5/20
363923/363923 - 2s - loss: 0.0050 - accuracy: 0.9990 - val_loss: 0.0055 - val_accuracy: 1.0000
Epoch 6/20
363923/363923 - 2s - loss: 0.0039 - accuracy: 0.9992 - val_loss: 0.0019 - val_accuracy: 1.0000
Epoch 7/20
363923/363923 - 2s - loss: 0.0035 - accuracy: 0.9993 - val_loss: 0.0031 - val_accuracy: 0.9997
Epoch 8/20
363923/363923 - 2s - loss: 0.0028 - accuracy: 0.9994 - val_loss: 0.0015 - val_accuracy: 1.0000
Epoch 9/20
363923/363923 - 2s - loss: 0.0028 - accuracy: 0.9994 - val_loss: 0.0018 - val_accuracy: 1.0000
Epoch 10/20
363923/363923 - 2s - loss: 0.0022 - accuracy: 0.9995 - val_loss: 0.0015 - val_accuracy: 1.0000
Epoch 11/20
363923/363923 - 2s - loss: 0.0020 - accuracy: 0.9996 - val_loss: 0.0011 - val_accuracy: 1.0000
Epoch 12/20
363923/363923 - 2s - loss: 0.0021 - accuracy: 0.9996 - val_loss: 0.0033 - val_accuracy: 0.9993
Epoch 13/20
363923/363923 - 2s - loss: 0.0017 - accuracy: 0.9997 - val_loss: 7.2115e-04 - val_accuracy: 1.0000
Epoch 14/20
363923/363923 - 2s - loss: 0.0015 - accuracy: 0.9997 - val_loss: 6.4973e-04 - val_accuracy: 1.0000
Epoch 15/20
363923/363923 - 2s - loss: 0.0014 - accuracy: 0.9997 - val_loss: 4.9727e-04 - val_accuracy: 1.0000
Epoch 16/20
363923/363923 - 2s - loss: 0.0014 - accuracy: 0.9997 - val_loss: 3.8057e-04 - val_accuracy: 1.0000
Epoch 17/20
363923/363923 - 2s - loss: 0.0013 - accuracy: 0.9997 - val_loss: 0.0018 - val_accuracy: 0.9998
Epoch 18/20
363923/363923 - 3s - loss: 0.0013 - accuracy: 0.9997 - val_loss: 0.0011 - val_accuracy: 1.0000
Epoch 19/20
363923/363923 - 2s - loss: 9.5826e-04 - accuracy: 0.9998 - val_loss: 1.4088e-04 - val_accuracy: 1.0000
Epoch 20/20
363923/363923 - 2s - loss: 8.8616e-04 - accuracy: 0.9998 - val_loss: 4.3943e-04 - val_accuracy: 1.0000
```

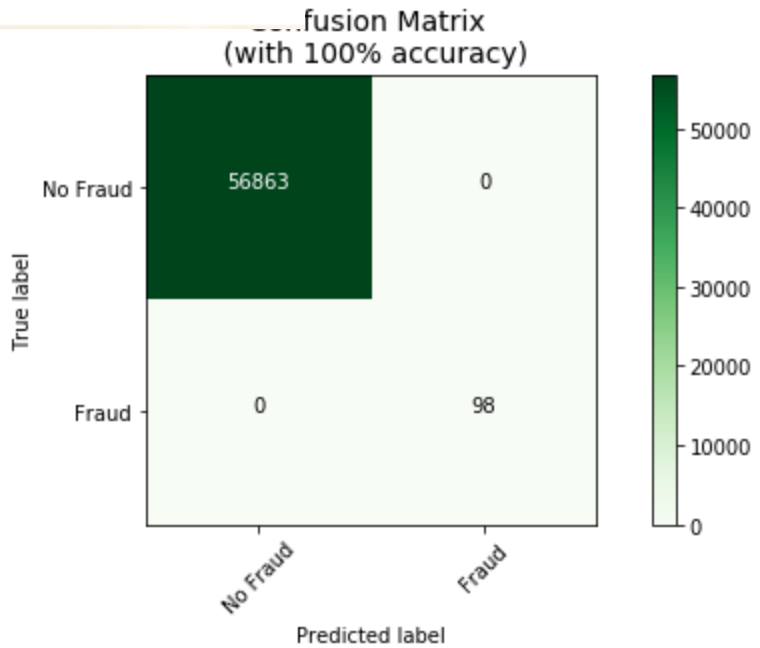
```
oversample_smote = confusion_matrix(or
actual_cm = confusion_matrix(original_
labels = ['No Fraud', 'Fraud']

fig = plt.figure(figsize=(16,8))

fig.add_subplot(221)
plot_confusion_matrix(oversample_smote

fig.add_subplot(222)
plot_confusion_matrix(actual_cm, label
```

```
Confusion matrix, without normalizatio
[[56851    12]
 [   30    68]]
Confusion matrix, without normalizatio
[[56863     0]
 [    0    98]]
```




```
, cmap=plt.cm.Oranges)

p=plt.cm.Greens)
```





Unsupervised Learning

Unsupervised Learning

Iris Data Set

Iris setosa

Iris versicolor

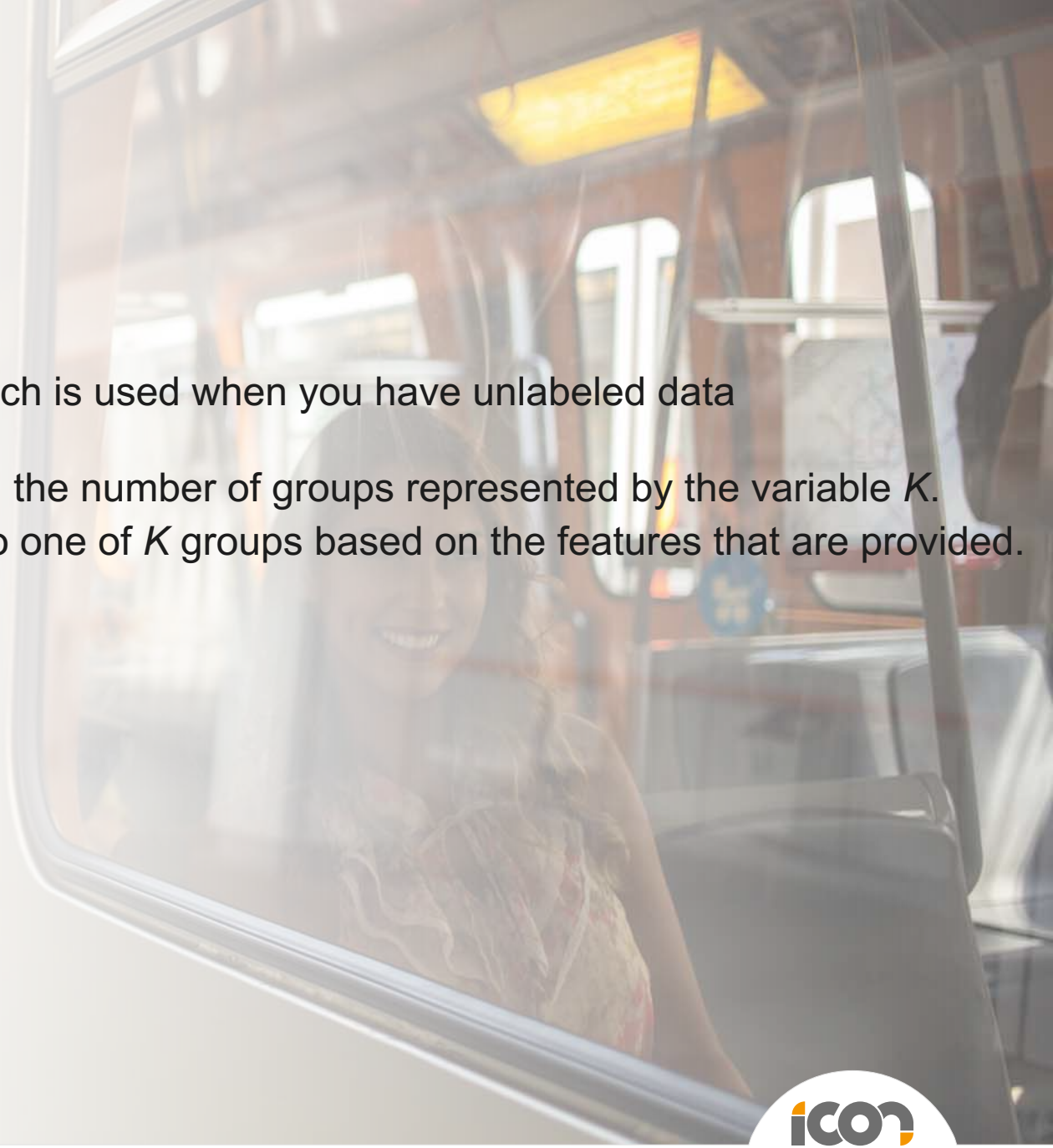
Iris virginica

- 50 samples from each of three species of *Iris*.
- Four features were measured from each sample:
- the length and the width of the sepals and petals, in centimeters.

- the objective of K-means is simple:
- group similar data points together and discover underlying patterns.

- To achieve this objective, K-means looks for a fixed number (*k*) of clusters in a dataset.

KMeans

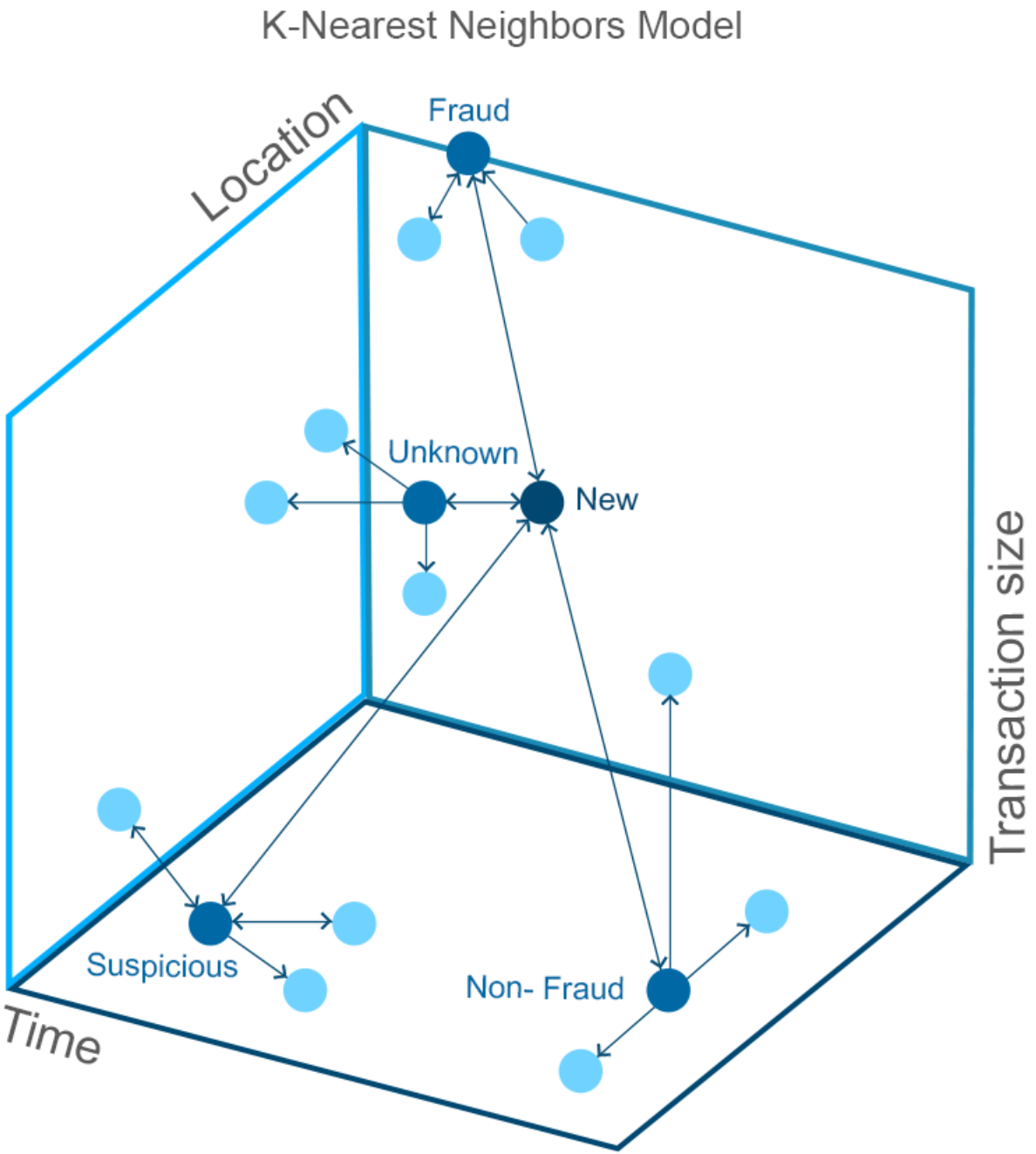


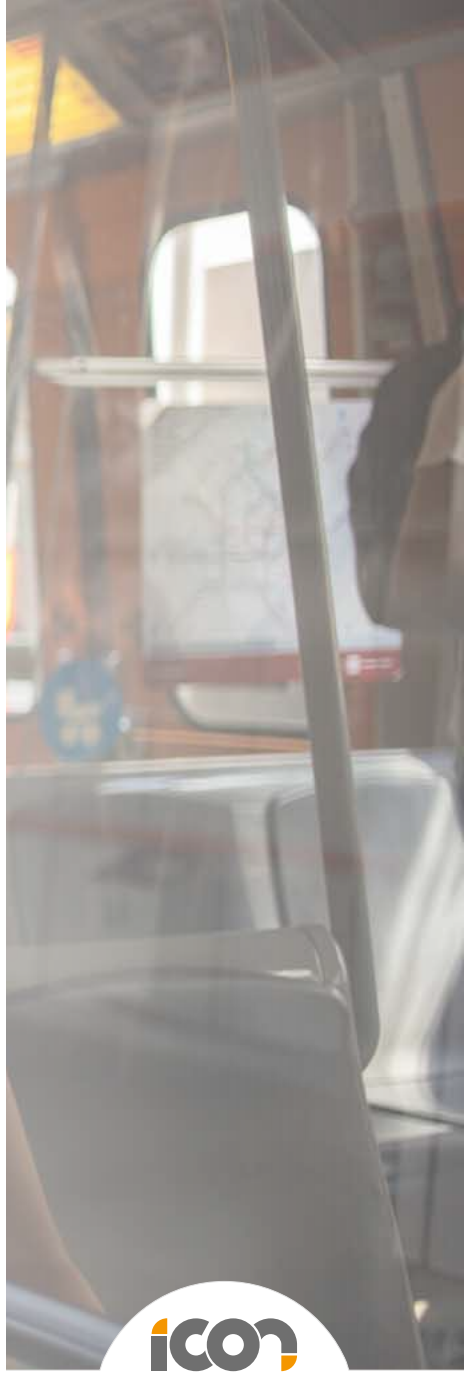
K-means clustering is a type of unsupervised learning, which is used when you have unlabeled data (i.e., data without defined categories or groups).

The goal of this algorithm is to find groups in the data, with the number of groups represented by the variable *K*.

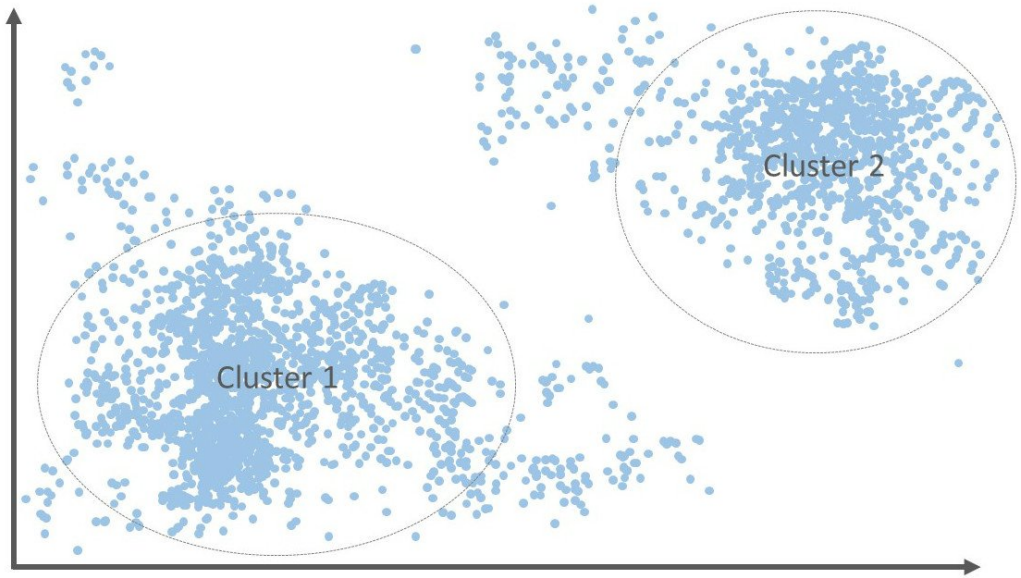
The algorithm works iteratively to assign each data point to one of *K* groups based on the features that are provided. Data points are clustered based on feature similarity.

Demonstration 3


K-Nearest Neighbors Model
Location
Fraud
Unknown
New
Transaction size
Suspicious
Non- Fraud
Time



Unsupervised Machine Learning



Supervised Machine Learning

Machine Learning and Fraud Detection

- Payments
- Demonstration
- Thoughts

Two Questions Every Machine Learning Project Should Ask

Is the purpose of the project ethical?

Is the implementation of the project ethical?

Two Questions Every Machine Learning Project Should Ask

Is the purpose of the project ethical?

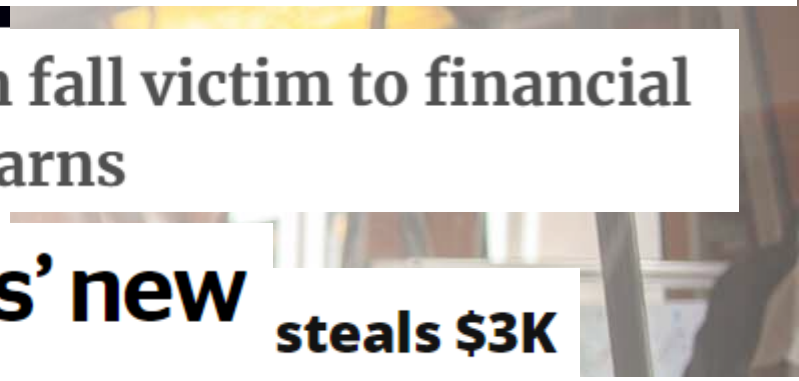
what are the additional benefits of the project?
who does it benefit?

THE TIMES

As more over-55s are targeted, regulators say average amount swindled in 2017 was £91,000

Action Fraud in
lost £18,000 but

More than five million fall victim to financial scams, Lloyds Bank warns



Fraud complaints hit record high as banks' new anti-scam measures delaye

steals $3K

The New York Times

Scam victims left unprotected

More people tricked into transferring money to

Is the purpose of the project ethical?

BBC NEWS

Home | UK | World | Business | Politics | Tech | Science | Health | Family & Edu

Business | Your Money | Market Data | Companies | Economy

herability,
and Find It in Older People

Swindlers prey on people over 70, who lose millions each year to their shifting tactics.

Bank payment scams claim 84,000 victims

ITV REPORT 12 September 2019 at 10:29am

nmers lived 'like Premier League

Fraud: The Public Threat ' after conning v

ITV REPORT 23 June 2019 at 11:59am

Scammers targeting flooding vic

Woman loses £300,000

Two Questions Every Machine Learning Project Should Ask

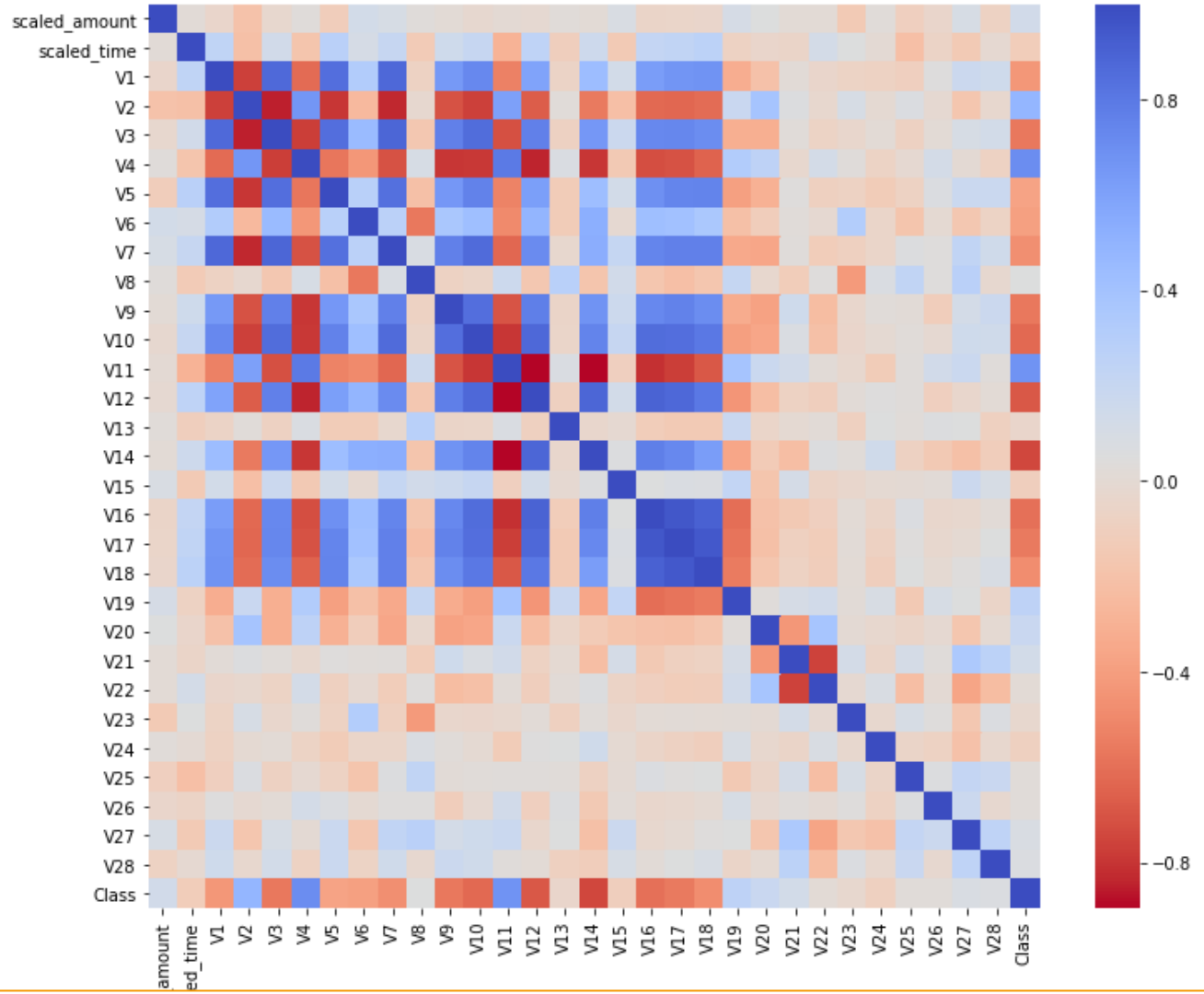
Is the implementation of the project ethical?

Does it implement unfair bias?

Disclose to stakeholders about their interactions with an AI

Governance:

- secure,
- reliable and robust, and
- appropriate processes are in place to ensure responsibility and accountability for those AI systems



Is the implementation of the project ethical?

- **Positive Correlations:** V2, V4, V11, and V19 are positively correlated. Notice how the higher these values are, the more likely the end result will be a fraud transaction.
- **Negative Correlations:** V17, V14, V12 and V10 are negatively correlated. Notice how the lower these values are, the more likely the end result will be a fraud transaction.

One last thing

Is it Intelligent?

THE TIMES

As more over-55s are targeted, regulators say average amount swindled in 2017 was £91,000

Action Fraud investigation: Victim fall victim to financial

Frau
anti-

More

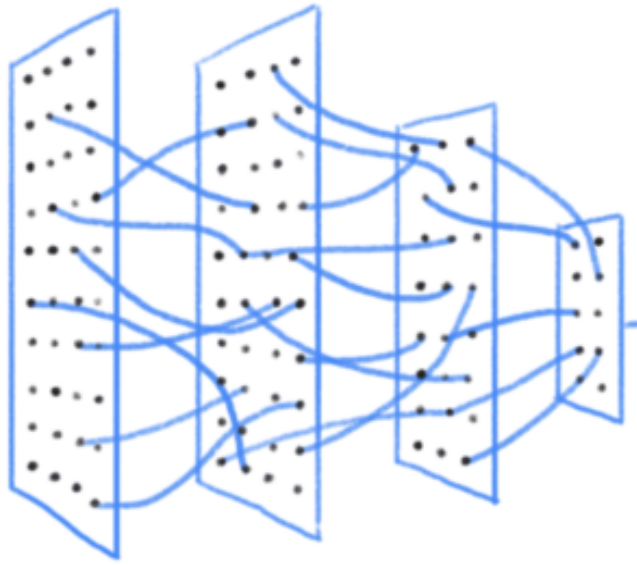
BBC

NEWS

Birmingh

Fraudulent Transaction

Non-Fraudulent Transaction



Phone scammers lived like Premier League footballers' after conning v

ITV REPORT 23 June 2019 at 11:59am

@CrosslandTamsin

targeting flooding vic

Woman loses £300,000