Anomaly detection - Part 1

One should look for what is and not what he thinks should be. -Albert Einstein

Who we are

- Data Society's mission is to **integrate Big Data and machine learning best practices across entire teams** and empower professionals to identify new insights
- We provide:
  - High-quality data science training programs
  - Customized executive workshops
  - Custom software solutions and consulting services
- Since 2014, we've worked with thousands of professionals to make their data work for them

How we teach

- Interactive knowledge checks (with links)
- Exercises (we give you 2 files and one has the answers)
- Code files to help you work faster
- Breaks

Best practices for virtual classes

1. Find a quiet place that is free of distractions. Headphones are recommended.
2. Remove or silence alerts from cell phones, e-mails, etc.
3. Participate in discussions and ask questions.
4. Give your honest feedback so we can troubleshoot problems and improve the course.

Introduction

- Before we start, please answer the following questions in the chat:
  - Why are you taking this class?
  - What are you most excited to learn about?
  - What kinds of problems could anomaly detection solve in your work?

Module completion checklist

Objective	Complete
Define anomaly concepts and uses	
Differentiate between types of anomalies	
Define the concept of DBSCAN and its parameter estimation	
Run and visualize DBSCAN for an arbitrary distance	
Optimize parameters of DBSCAN	

What is an anomaly?

- **Anomalies** are objects with **behaviors that are very different from expectations**
- **Anomaly detection** is the process of finding anomalies
- Anomaly detection is also called outlier detection

A use for anomaly detection

- Imagine you are a part of the inspection team in an electric transmission company
- To minimize the losses in power grid transmission, you may use anomaly detection to identify energy usages that are rather different from typical cases
- For example, if the energy consumption changes drastically from the normal consumption, then the transmission is suspicious.
- The loss may have caused by short circuits, grid failures, or electricity burglars

Other uses

- Anomaly detection may also be useful for:
  - fraud detection (insurance, banking)
  - intrusion detection (computer networks, national surveillance)
  - medical informatics (diagnosis, disorder detection)
  - fault/damage detection (commerce, industry)

Outliers

- An **outlier** is the data object that deviates significantly from the rest of the objects
- Outliers are different from noisy data
- Outlier detection can be related to novelty detection in evolving datasets
- Novelty detection may identify new topics and trends in a timely manner
- Novelty detection is used when we are interested in detecting whether a new observation is an outlier

Types of outliers

- There are three types of outliers:
  - Global outliers
  - Contextual outliers
  - Collective outliers
- **We will concentrate on global outlier detection in this module, but let's understand each type**

Global outliers

- **Global outliers** are data points that deviate from the rest of the data
- They are called point anomalies and are the simplest type of outliers

Contextual outliers

- **Contextual outliers** are conditional on the selected context

- Is it an outlier to say “the temperature today is 84 degrees Fahrenheit”?
- It depends on the time and location
- If it is winter in Toronto, then yes, it is an outlier!

Collective outliers

- If a subset of data objects deviates significantly from the entire dataset, then they are called **collective outliers**

- Imagine you handle the shipments of goods in a supply chain organization
- If the shipment of an order is delayed, it is not an outlier because delays occur from time-to-time
- But if 100 orders are delayed on a single day it may be a deviation from the norm, making those 100 shipments outliers

Module completion checklist

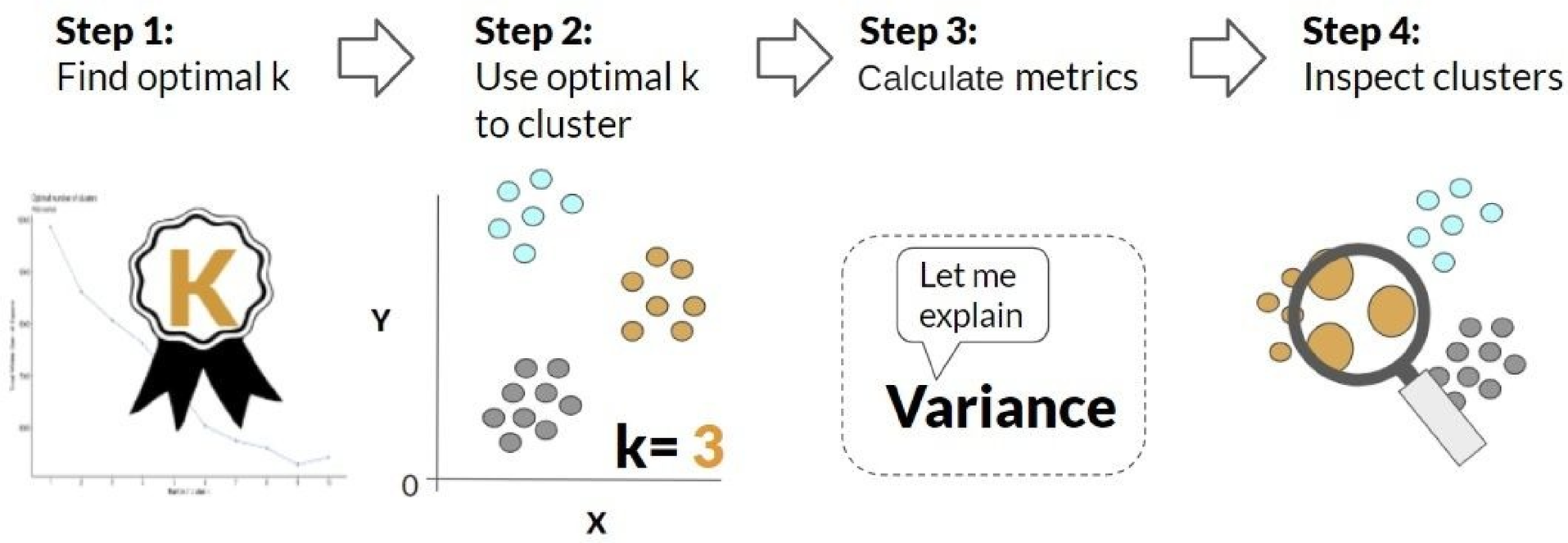
Objective	Complete
Define anomaly concepts and uses	✔
Differentiate between types of anomalies	✔
Define the concept of DBSCAN and its parameter estimation	
Run and visualize DBSCAN for an arbitrary distance	
Optimize parameters of DBSCAN	

Clustering

- Clustering is a type of unsupervised machine learning
- You find similarities between data points and create groups (clusters) based on those similarities
- It tries to find whether there is a relationship between the data points when the classes are unknown

K-means

- K-means is popular type of clustering because it is easy to understand
- In k-means, you define a target number k, which refers to the number of clusters that you need in the dataset
- All points are assigned to a cluster

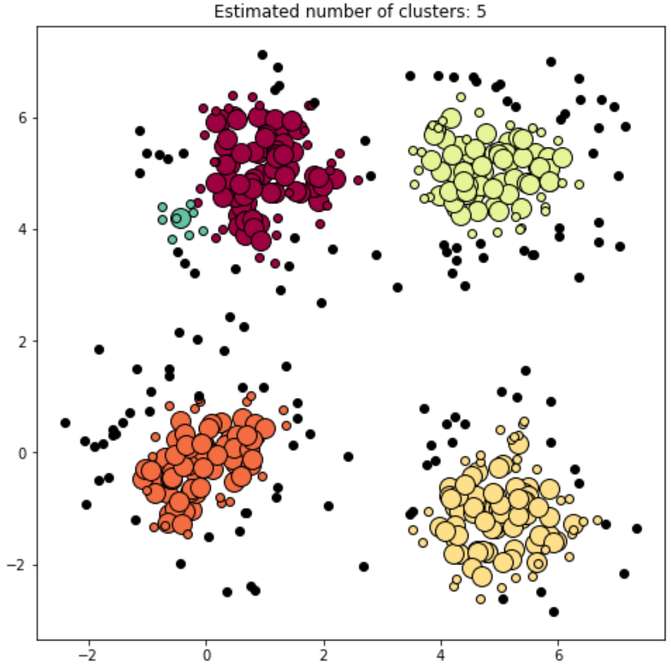


K-means

- While simple, k-means clustering does not take outliers into consideration
- All points are assigned to a cluster even if they may not belong
- Outliers assigned to clusters can lead to **misleading interpretations of collected data**
- We need another algorithm!

DBSCAN

- Density-Based Spatial Clustering of Applications with Noise (**DBSCAN**) is a popular density-based clustering algorithm
- Unlike k-means, DBSCAN doesn't have a requirement to set a number of clusters in advance
- Using density (the number of data points within a specified radius from a random data point) it can:
  - discover clusters with an **arbitrary shape**
  - **identify outliers** (points that are not part of any cluster)



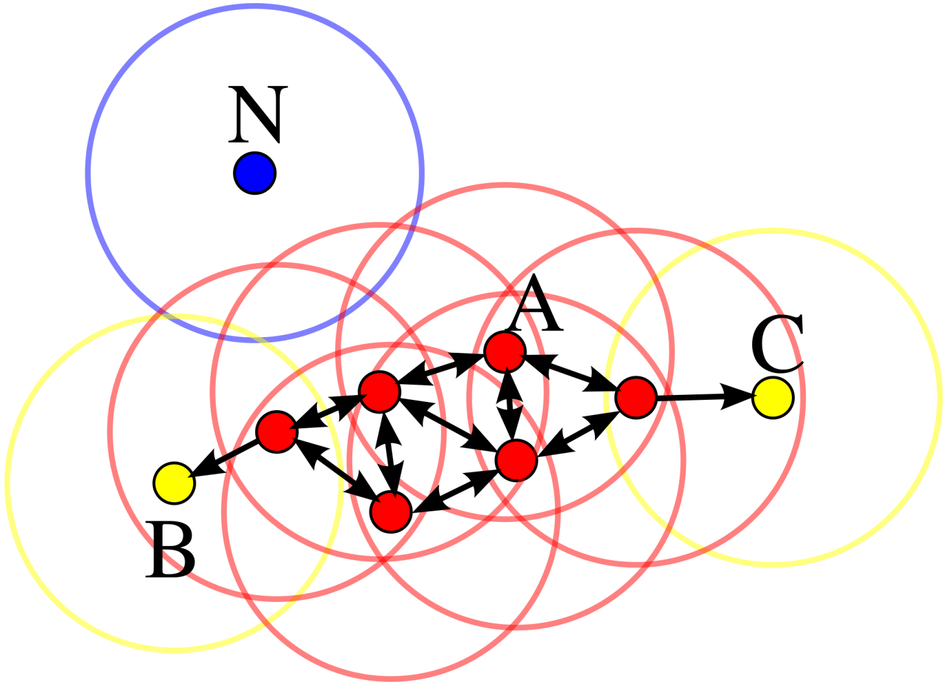
The DBSCAN algorithm

- The DBSCAN algorithm can be summarized into the following steps:

1. Pick a random point
2. Compute its neighborhood to determine if it is a core point or an outlier
3. If it’s a core point, expand the cluster by adding points directly in reach
4. Add all density-reachable points by jumping neighborhoods of the points assigned to the cluster
5. If an outlier is added to a cluster, reassign it as a border point
6. Repeat previous steps until all data points are assigned to a cluster or labeled an outlier

The DBSCAN algorithm

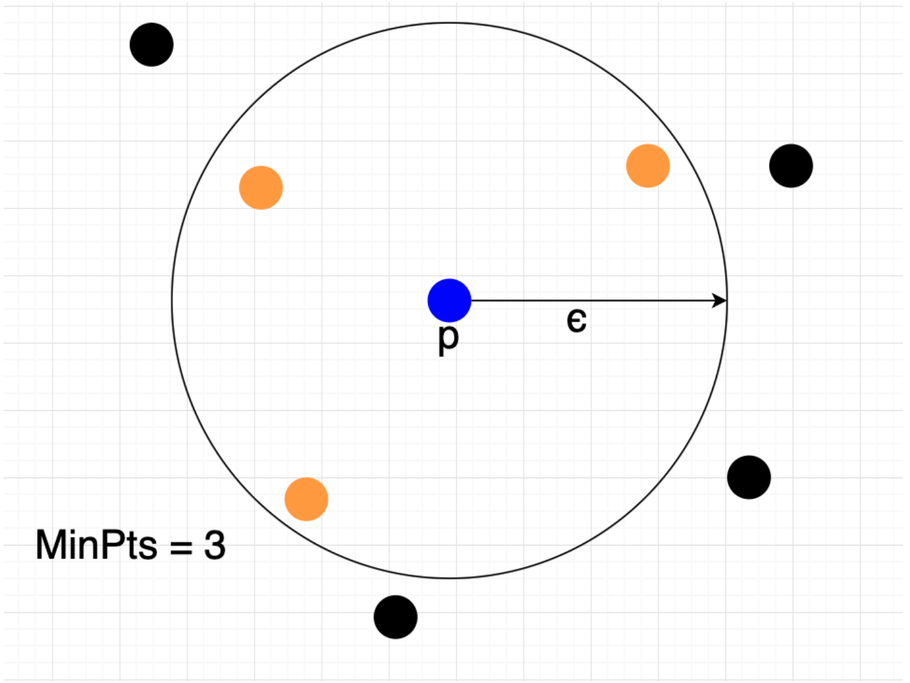
- `MinPts` is set to `4` in the diagram
- The red points are **core points which form a single cluster** as they are all **reachable from one another**
- Yellow points `B` and `C` are not core points, but are reachable from them
- Hence, they belong to the cluster
- Blue point `N` is an outlier as it is neither a core point, nor reachable from them



Parameters

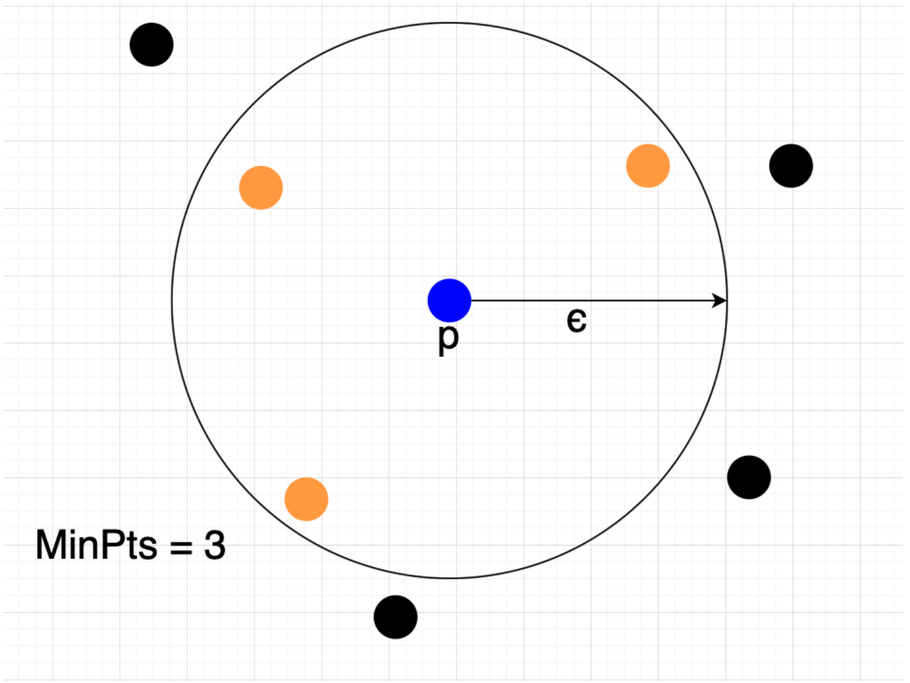
There are 2 parameters in DBSCAN:

- ϵ:
  - the **maximum distance between two points** for them to be in the **same neighborhood**
  - it can also be defined as the radius of our neighborhood around a data point p
  - if the distance between a data point and p is this value or lower, then that point is a neighbor of p



Parameters

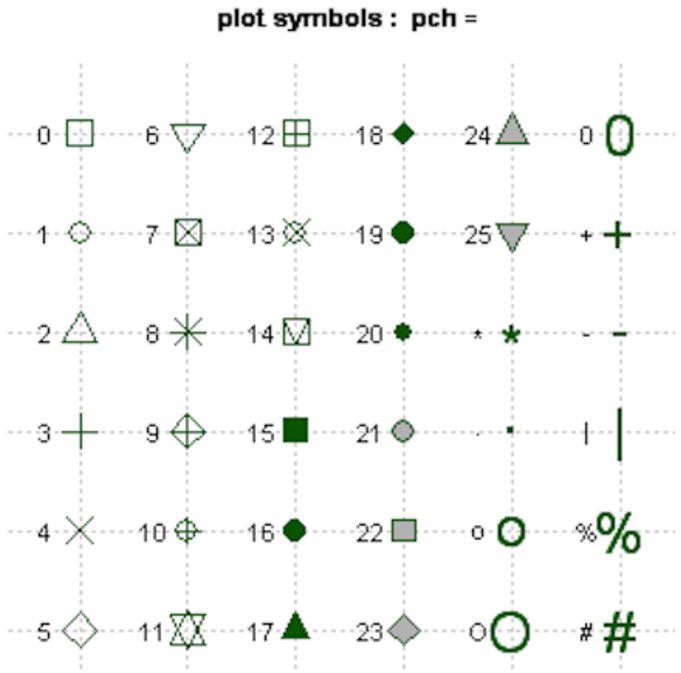
- `MinPts`:
  - the **minimum number of data points** required to form a dense region
  - if we set `MinPts` as 10, then we need 10 points in the neighborhood region to form a dense region



Assigning data points

Based on ϵ, the data points are assigned to three categories:

- **Core point** `p`:
  - Point `p` is a core point if at least `minPts` points are within distance ϵ of it
- **Border point** `q`:
  - Does not have `minPts` points within distance ϵ of it
  - Still part of the cluster as it's reachable from a core point `p`,
- **Outlier** `o`:
  - Neither a core point or a border point
  - The **"other"** class



Parameter tuning

- ϵ:
  - Setting ϵ to be very small will mean that a **large part of data will not be clustered**
  - Setting ϵ to be very large will result in **all the points forming a single cluster**
  - We can use a **k-distance graph** to find the optimal value for ϵ
- `MinPts`:
  - Can be derived from a **number of dimensions (D)** in the dataset, such as

$$MinPts \geq D + 1$$

- Larger datasets should generally have larger `MinPts` value

Knowledge check 1

Module completion checklist

Objective	Complete
Define anomaly concepts and uses	✔
Differentiate between types of anomalies	✔
Define the concept of DBSCAN and its parameter estimation	✔
Run and visualize DBSCAN for an arbitrary distance	
Optimize parameters of DBSCAN	

Our objectives

- In this module we will try to accurately determine if transactions in the **Paysim** dataset are fraudulent
  - This is a fraud detection dataset, which has rows of credit transactions and the target specifying if the particular transaction is a fraud transaction or not
  - You can read more about the dataset ***here***
- In the exercises for this module we will try to detect whether locations in the **Seismic hazards** dataset are prone to seismic hazard
  - This dataset contains data on the mining activity in different locations
  - Since earthquakes happen infrequently, the rows that are prone to hazards are anomalous from rest of the observations
  - Read more about the dataset ***here***

Loading packages

Let's load the packages we will be using:

```
import os
from pathlib import Path
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import pickle

from sklearn import metrics
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix, roc_auc_score
from imblearn.over_sampling import SMOTE

from sklearn.preprocessing import StandardScaler
from sklearn.neighbors import NearestNeighbors
from sklearn.cluster import DBSCAN
from sklearn.tree import DecisionTreeClassifier
```

Directory settings

- In order to maximize the efficiency of your workflow, you should encode your directory structure into `variables`
- We will use the `pathlib` library
- Let the `main_dir` be the variable corresponding to your course materials folder
- `data_dir` be the variable corresponding to your `data` folder

```
# Set 'main_dir' to location of the project folder
from pathlib import Path
home_dir = Path(".").resolve()
main_dir = home_dir.parent.parent
print(main_dir)
```

```
data_dir = str(main_dir) + "/data"
print(data_dir)
```

Load the dataset

- Load `paysim_transactions.csv` dataset and print the head

```
paysim =
pd.read_csv(str(data_dir)+"/paysim_transactions.cs

paysim.head()
```

```
   step      type      amount   ...
newbalanceDest  isFraud  isFlaggedFraud
0   308  CASH_OUT    94270.99  ...
486682.07         0                 0
1   215  TRANSFER  1068883.00  ...
5165788.35        0                 0
2   326  TRANSFER  2485281.21  ...
2663110.80        0                 0
3   371   PAYMENT     2243.36  ...
0.00          0                 0
4   283   PAYMENT     5845.82  ...
0.00          0                 0

[5 rows x 11 columns]
```

```
paysim.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 20220 entries, 0 to 20219
Data columns (total 11 columns):
 #   Column           Non-Null Count  Dtype
---  ------           --------------  -----
 0   step             20220 non-null  int64
 1   type             20220 non-null  object
 2   amount           20220 non-null  float64
 3   nameOrig         20220 non-null  object
 4   oldbalanceOrg    20220 non-null  float64
 5   newbalanceOrig   20220 non-null  float64
 6   nameDest         20220 non-null  object
 7   oldbalanceDest   20220 non-null  float64
 8   newbalanceDest   20220 non-null  float64
 9   isFraud          20220 non-null  int64
 10  isFlaggedFraud   20220 non-null  int64
dtypes: float64(5), int64(3), object(3)
memory usage: 1.7+ MB
```

Understand the dataset

```
paysim.columns
```

```
Index(['step', 'type', 'amount', 'nameOrig',
'oldbalanceOrg', 'newbalanceOrig',
       'nameDest', 'oldbalanceDest',
'newbalanceDest', 'isFraud',
       'isFlaggedFraud'],
      dtype='object')
```

- `step` - the unit of time
- `type` - type of transaction
- `amount` - amount of transaction in local currency
- `nameOrig` - customer who started the transaction
- `oldbalanceOrg` - initial balance before the transaction
- `newbalanceOrig` - new balance after the transaction
- `nameDest` - customer who is the recipient of the transaction
- `oldbalanceDest` - initial balance recipient before the transaction
- `newbalanceDest` - new balance recipient after the transaction
- `isFraud` - our target if it's a fraud or not
- `isFlaggedFraud` - tells whether the transaction is flagged for fraudulent activity

Target of the dataset

- `isFraud` is our target which tells if a particular transaction is fraud (`1`) or not (`0`)

```
paysim['isFraud'].value_counts()
```

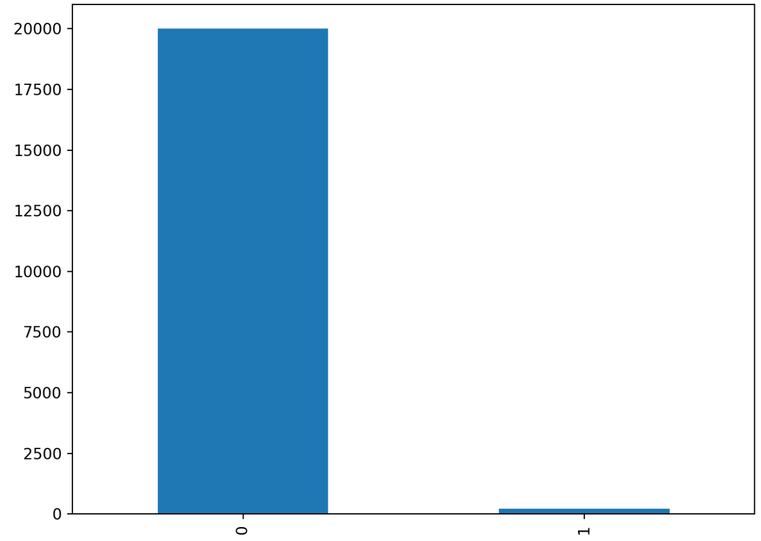
```
0    20000
1      220
Name: isFraud, dtype: int64
```

```
print(paysim['isFraud'].value_counts() / len(paysim))
```

```
0    0.98912
1    0.01088
Name: isFraud, dtype: float64
```

- Approximately 99% of the transactions are not fraud and only 1% of the transactions are fraud
- Anomalous behavior is rare and that is why we describe our dataset as highly imbalanced

```
paysim['isFraud'].value_counts().plot(ki
= 'bar')
```



- Our target is highly imbalanced between two classes

EDA of the dataset

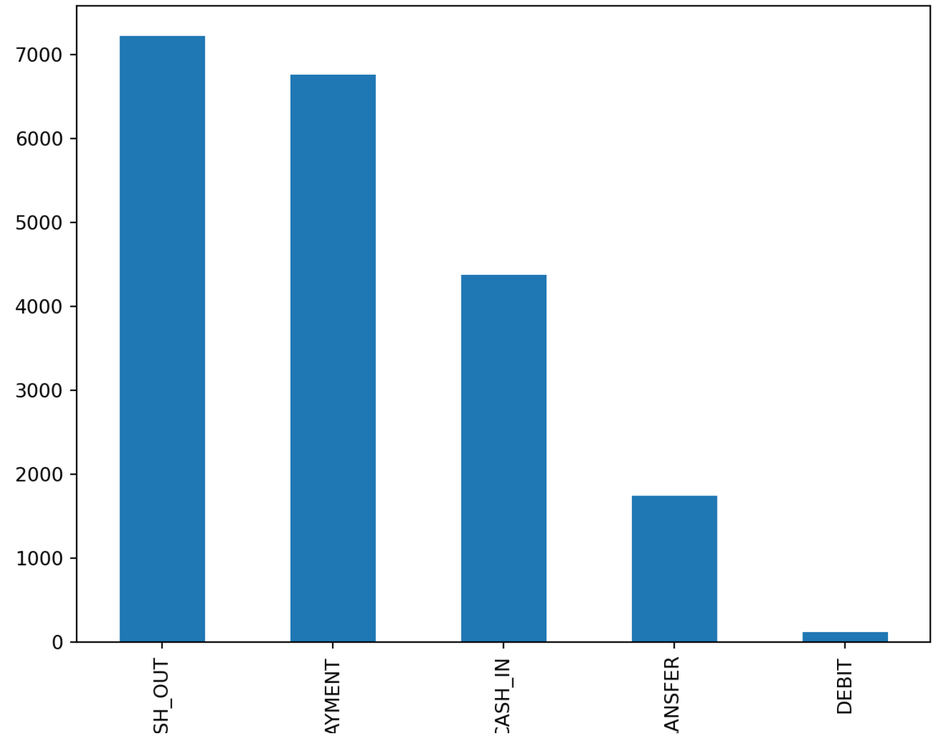
- Check for null values

```
paysim.isnull().sum()
```

```
step              0
type              0
amount            0
nameOrig          0
oldbalanceOrg     0
newbalanceOrig    0
nameDest          0
oldbalanceDest    0
newbalanceDest    0
isFraud           0
isFlaggedFraud    0
dtype: int64
```

- View the frequency of type of transactions

```
paysim['type'].value_counts().plot(kind = 'bar')
```



Prepare data for DBSCAN

For unsupervised clustering, we need to:

- Prepare:
- Subset the dataset to include only the numeric variables
- Remove the labels so that we can use unsupervised learning
- Clean:
  - Address **NAs**, missing values
  - Scale data

Subset the data

- We will subset the dataset to include just the numeric variables and remove the target variable `isFraud` as well from our data

```
# Subset variables from fraud dataset
paysim_dbscan = paysim.drop(['step', 'type','nameOrig', 'nameDest', 'isFlaggedFraud','isFraud'],
axis = 1)
print(paysim_dbscan.head())
```

```
       amount  oldbalanceOrg  newbalanceOrig  oldbalanceDest  newbalanceDest
0    94270.99            0.0             0.0       392411.08       486682.07
1  1068883.00          227.3             0.0      4096905.34      5165788.35
2  2485281.21        54940.0             0.0       177829.59      2663110.80
3     2243.36            0.0             0.0            0.00            0.00
4     5845.82            0.0             0.0            0.00            0.00
```

Data cleaning: NAs

- First, we check how many NAs there are in each column

```
print(paysim_dbscan.isnull().sum())
```

```
amount              0
oldbalanceOrg       0
newbalanceOrig      0
oldbalanceDest      0
newbalanceDest      0
dtype: int64
```

StandardScalar

- Let's instantiate the scaler and transform our `paysim_dbscan` dataset
- We create a new object `paysim_dbscan_scaled`

```
# Instantiate MinMaxScaler.
scaler = StandardScaler()

# Scale the dataframe.
paysim_dbscan_scaled = scaler.fit_transform(paysim_dbscan)
```

```
# Convert back to dataframe, making sure to name the columns again.
paysim_dbscan_scaled = pd.DataFrame(paysim_dbscan_scaled, columns = paysim_dbscan.columns)
print(paysim_dbscan_scaled.head())
```

```
     amount  oldbalanceOrg  newbalanceOrig  oldbalanceDest  newbalanceDest
0 -0.158339      -0.291955       -0.291554       -0.210158       -0.205427
1  1.360976      -0.291878       -0.291554        0.924753        1.108908
2  3.568988      -0.273385       -0.291554       -0.275897        0.405920
3 -0.301800      -0.291955       -0.291554       -0.330377       -0.342133
4 -0.296184      -0.291955       -0.291554       -0.330377       -0.342133
```

sklearn.cluster.DBSCAN

```
DBSCAN(eps = ..,
       min_samples = ..,
       ..)
```

- The main parameters are:
  - `eps`: maximum distance between two samples for them to be in the same neighborhood
  - `min_samples`: number of samples in a neighborhood for a point to be considered as a core point

sklearn.cluster.DBSCAN

class sklearn.cluster. **DBSCAN** (*eps=0.5*, *min_samples=5*, *metric='euclidean'*, *metric_params=None*, *algorithm='auto'*, *leaf_size=30*, *p=None*, *n_jobs=None*) [source]

Perform DBSCAN clustering from vector array or distance matrix.

DBSCAN - Density-Based Spatial Clustering of Applications with Noise. Finds core samples of high density and expands clusters from them. Good for data which contains clusters of similar density.

Read more in the User Guide.

Parameters:

eps : *float, optional*
The maximum distance between two samples for them to be considered as in the same neighborhood.

min_samples : *int, optional*
The number of samples (or total weight) in a neighborhood for a point to be considered as a core point. This includes the point itself.

metric : *string, or callable*
The metric to use when calculating distance between instances in a feature array. If metric is a string or callable, it must be one of the options allowed by `sklearn.metrics.pairwise_distances` for its metric parameter. If metric is "precomputed", X is assumed to be a distance matrix and must be square. X may be a sparse matrix, in which case only "nonzero" elements may be considered neighbors for DBSCAN.

New in version 0.17: metric *precomputed* to accept precomputed sparse matrix.

metric_params : *dict, optional*
Additional keyword arguments for the metric function.

New in version 0.19

DBSCAN: model

- We now run the model on `paysim_dbscan_scaled`
- We can tweak the parameters ϵ and `MinPts` to get a plot which makes sense
- For now, we will set ϵ to have a radius of `0.2`
- We set `MinPts` (`min_samples` in the function) to `5`
- That means that we want 5 samples in a neighborhood with a radius of 0.2

```
# Let's run DBSCAN.
dbscan = DBSCAN(eps=0.2, min_samples = 5)
clusters = dbscan.fit_predict(paysim_dbscan_scaled)

# Check the number of clusters
unique, counts = np.unique(clusters, return_counts=True)
print(np.asarray((unique, counts)).T)
```

```
[[   -1   901]
 [    0 19139]
 [    1     7]
 [    2     7]
 [    3     9]
 [    4    21]
 [    5    21]
 [    6     5]
 [    7     7]
 [    8    11]
 [    9     5]
```

Exercise 1

Module completion checklist

Objective	Complete
Define anomaly concepts and uses	✔
Differentiate between types of anomalies	✔
Define the concept of DBSCAN and its parameter estimation	✔
Run and visualize DBSCAN for an arbitrary distance	✔
Optimize parameters of DBSCAN	

DBSCAN: optimizing eps

- We can find the optimal value for ϵ using a **k-distance** approach
- It consists of computing the average distances of every point to its k-nearest neighbors
- k-value refers to `MinPts`, which will be specified by the user
- The optimal `eps` parameter will be determined by the “elbow” plot

Nearest Neighbors

- We will be using sklearn's **NearestNeighbors()** function to find the k-number of samples closest in distance to a point
- Read more about the function ***here***

Optimal Eps determination

Steps to determine the optimal ϵ

- Instantiate **NearestNeighbors()** with `MinPts`. Let's set this value to be 10 (twice the dimensions in our data)
- Calculate the average distance between each point in the data set and its 5 nearest neighbors
- Sort distance values by ascending value
- Plot the sorted distances to find the optimal eps using the elbow plot

Optimal Eps determination

- Instantiate **NearestNeighbors()** with 10 neighbors and fit the estimator on `paysim_dbscan_scaled`
- Retrieve the distances of every point with it's 10 closest neighbors

```
nn_model = NearestNeighbors(n_neighbors=10)
nbrs = nn_model.fit(paysim_dbscan_scaled)
distances, indices = nbrs.kneighbors(paysim_dbscan_scaled)
```

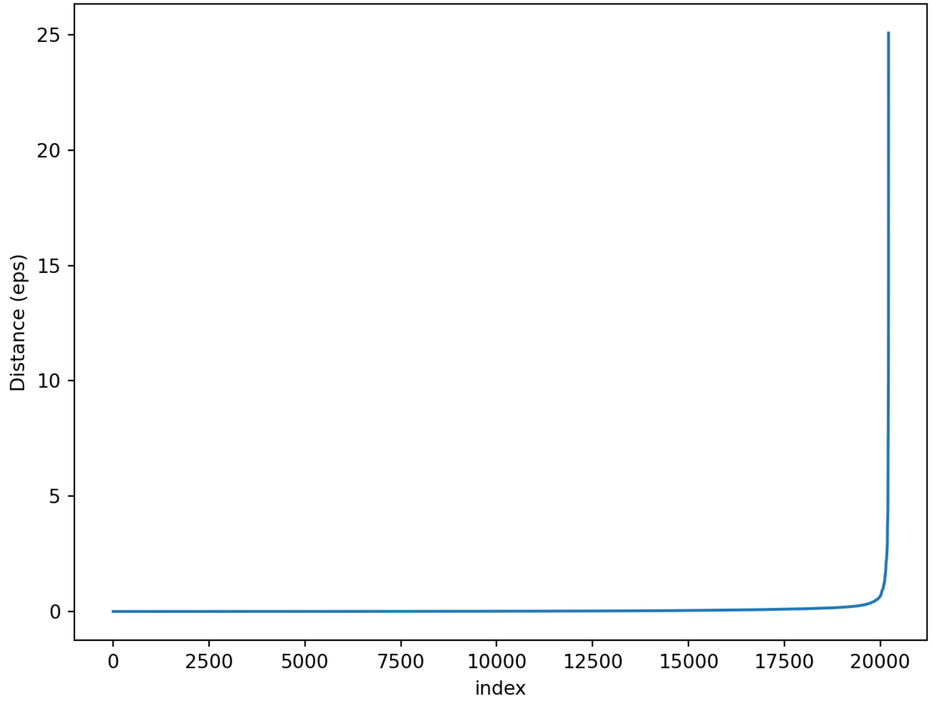
- Average distance calculation for each point and sorting the distances

```
distances = np.mean(distances,axis=1)
distances = np.sort(distances, axis=0)
```

Elbow plot to find optimal Eps

- Plot the sorted distances

```
fig, ax = plt.subplots()
elbow = ax.plot(distances)
plt.xlabel('index')
plt.ylabel('Distance (eps)')
plt.show()
```



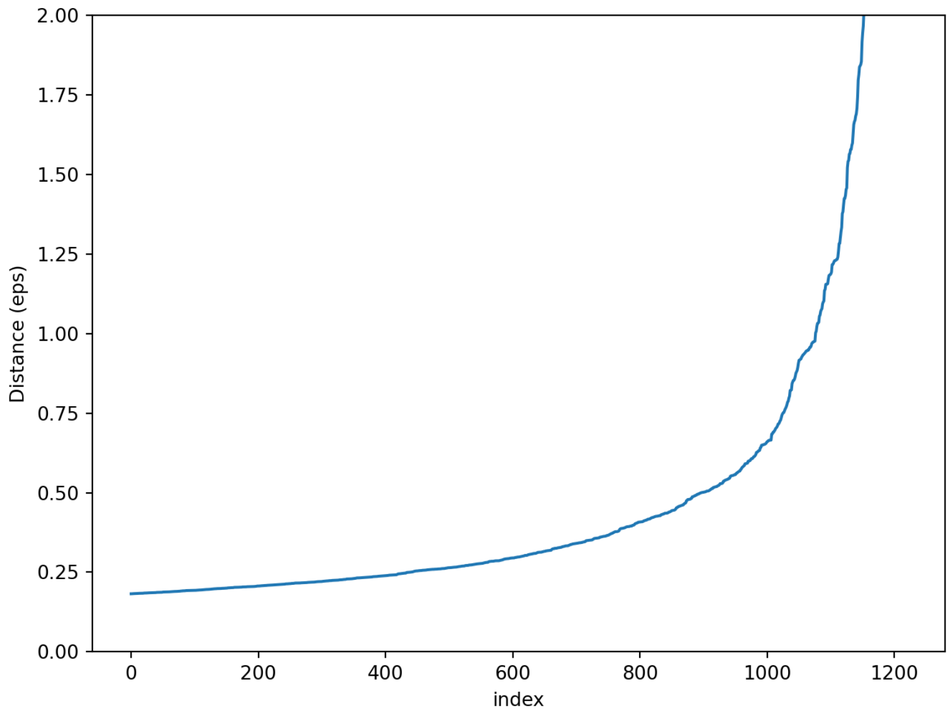
- The optimal value for **eps** is the distance value at the **crook of the elbow**, or the point of maximum curvature

Elbow plot to find optimal Eps

- To find the optimal value of eps, let's zoom the plot by filtering the distances after index 19000 and specifying the `y` limit range between 0 and 2

```
fig, ax = plt.subplots()
elbow_zoom = ax.plot(distances[19000:])
plt.xlabel('index')
plt.ylabel('Distance (eps)')
plt.ylim(0, 2)
plt.show()
```

```
(0.0, 2.0)
```



- It looks like the optimal eps is between **0.5** and **0.75**

DBSCAN: optimized model

- Rerun the model with optimized `eps` value
- You can also set different `MinPts` value based on your domain knowledge

```
# DBSCAN
dbscan = DBSCAN(eps=0.5, min_samples = 10)
optimized_clusters = dbscan.fit_predict(paysim_dbscan_scaled)

# Check the number of clusters
unique, counts = np.unique(optimized_clusters, return_counts=True)
print(np.asarray((unique, counts)).T)
```

```
[[   -1   387]
 [    0 19833]]
```

- Assign the cluster results to `paysim_dbscan`
- Anomaly points have cluster value of -1 and the rest of the cluster values represent inlier points
- We will replace the cluster values to **1** for anomalies and **0** for inliers

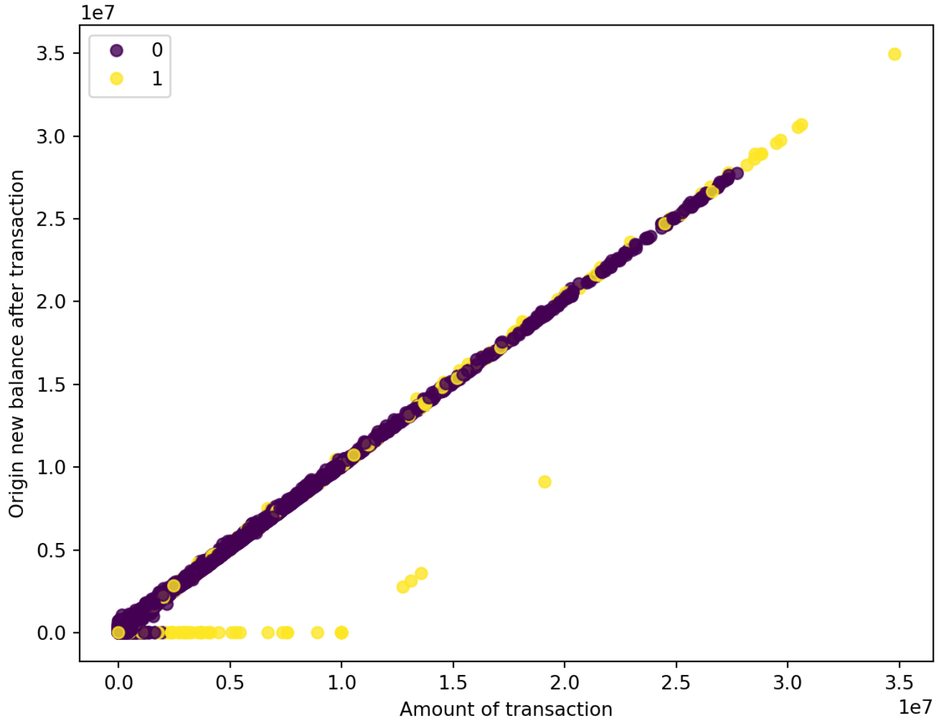
```
paysim_dbscan['cluster'] = optimized_clusters

paysim_dbscan.loc[paysim_dbscan['cluster'] >= 0,'cluster'] = 0
paysim_dbscan.loc[paysim_dbscan['cluster'] == -1,'cluster'] = 1
```

Visualize the Anomalies

- Let’s visualize the anomalies detected by the DBScan algorithm
- The scatterplot shows the variables **oldbalanceOrg** and **newbalanceOrig**
- Points represented in **yellow** are anomaly points

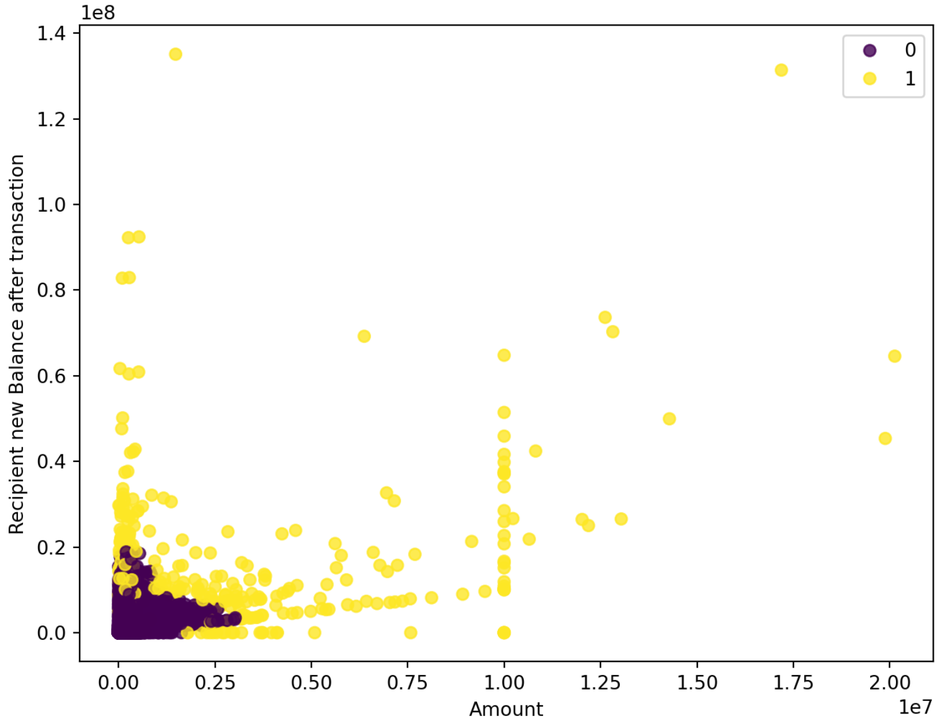
```
fig, ax = plt.subplots()
sc = ax.scatter(paysim_dbscan['oldbalanceOrg'],
paysim_dbscan['newbalanceOrig'], c =
paysim_dbscan['cluster'], alpha = 0.8)
ax.legend(*sc.legend_elements())
plt.xlabel('Amount of transaction')
plt.ylabel('Origin new balance after
transaction')
```



Visualize the Anomalies

- Here, we show a scatterplot between variables **amount** and **newbalanceDest**

```
fig, ax = plt.subplots()
sc = ax.scatter(paysim_dbscan['amount'],
paysim_dbscan['newbalanceDest'], c =
paysim_dbscan['cluster'], alpha = 0.8)
ax.legend(*sc.legend_elements())
plt.xlabel('Amount')
plt.ylabel('Recipient new Balance after
transaction')
plt.show()
```



What can be interpreted from the plots? Did DBSCAN perform well in identifying the anomalies?

Module completion checklist

Objective	Complete
Define anomaly concepts and uses	✔
Differentiate between types of anomalies	✔
Define the concept of DBSCAN and its parameter estimation	✔
Run and visualize DBSCAN for an arbitrary distance	✔
Optimize parameters of DBSCAN	✔

Knowledge check 2

Exercise 2

What’s next?

- In this module we:
  - talked about use cases for anomaly detection
  - learned about the DBSCAN algorithm
- In the next module, we will:
  - implement DBSCAN on time series data
  - examine why classification techniques are not useful for anomaly detection
  - learn about SMOTE analysis

Congratulations on completing this module!