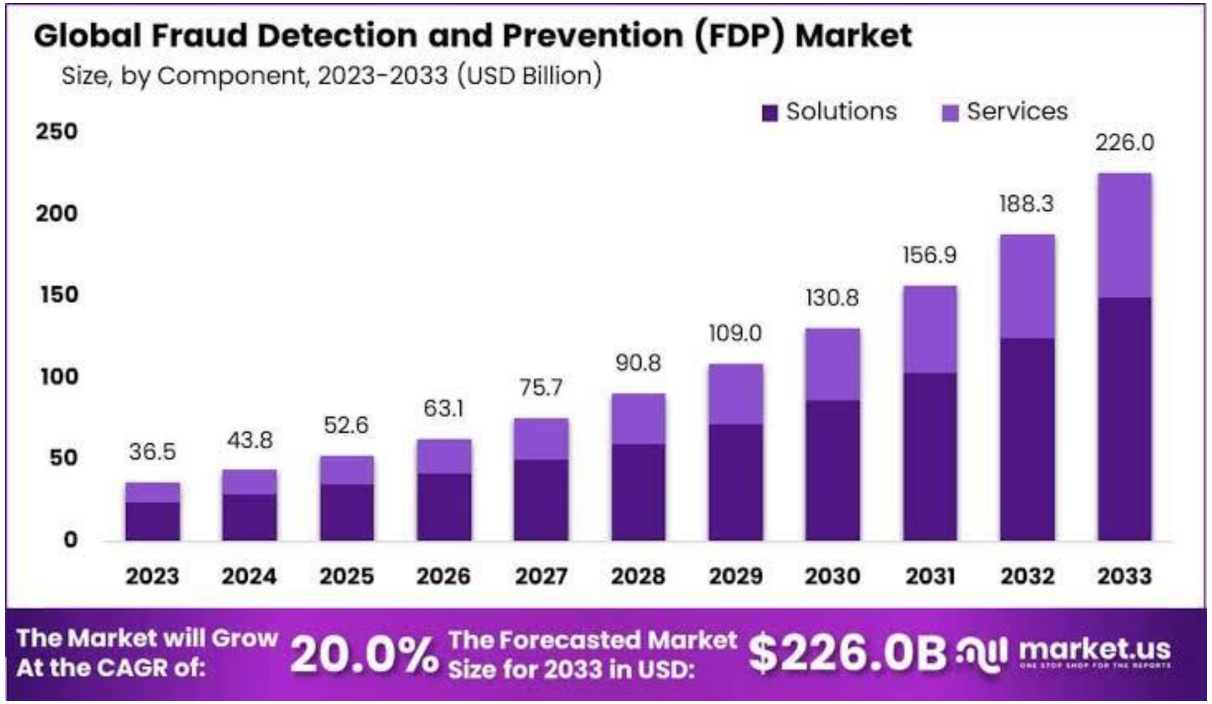
Fraud (anomalies) detection and prevention

- One area that machine learning finds important applications is fraud detection. With such large sums of money at stake, it is easy to see why.
- According to *Statista*, the global e-commerce fraud detection and prevention market was estimated at 36.7 billion U.S. dollars in 2021. Forecasts suggest that this figure will continue to grow steadily in the coming years, surpassing the 75 billion dollar mark by 2027.



Fraud detection and prevention

- Areas where fraud detection and prevention are applied include insurance claims, money laundering, electronic payments, and bank transactions, both online and offline.
- The challenge is to *quickly* identify and separate anomalous transactions from those that are legitimate, without impacting on customer experience.
- Note that companies are also suffering because of lost sales when genuine transactions are declined by fraud management systems.

Fraud detection and prevention

How can Machine learning help?

- Before Computer: Banks and financial institutions analyzed their customers’ behavioral patterns for any signs of abnormality, and designed checking rules to “flag” abnormal transactions manually.
- After Computer: Computers gave them the ability to identify and flag abnormal transactions in real-time.
- After ML: Machine learning tools ‘learn’ new patterns, without the need for human intervention. This allows models to adapt over time to uncover previously unknown patterns or identify new tactics that might be employed by fraudsters.

Fraud detection and prevention

Two keys objectives:

(1) High “accuracy”:

- high TPR (sensitivity, recall): TP/(TP+FN), equivalently, small number of false negatives
- high TFR (specificity): TN/(TN+FP), equivalently, small number of false positives
- high precision: TP/(TP+FP)

- F1-score $= \dfrac{2}{\frac{1}{\text{Precision}} + \frac{1}{\text{Recall}}} = \dfrac{2 \times \text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}$

Fraud detection and prevention

(2) Avoid overfitting

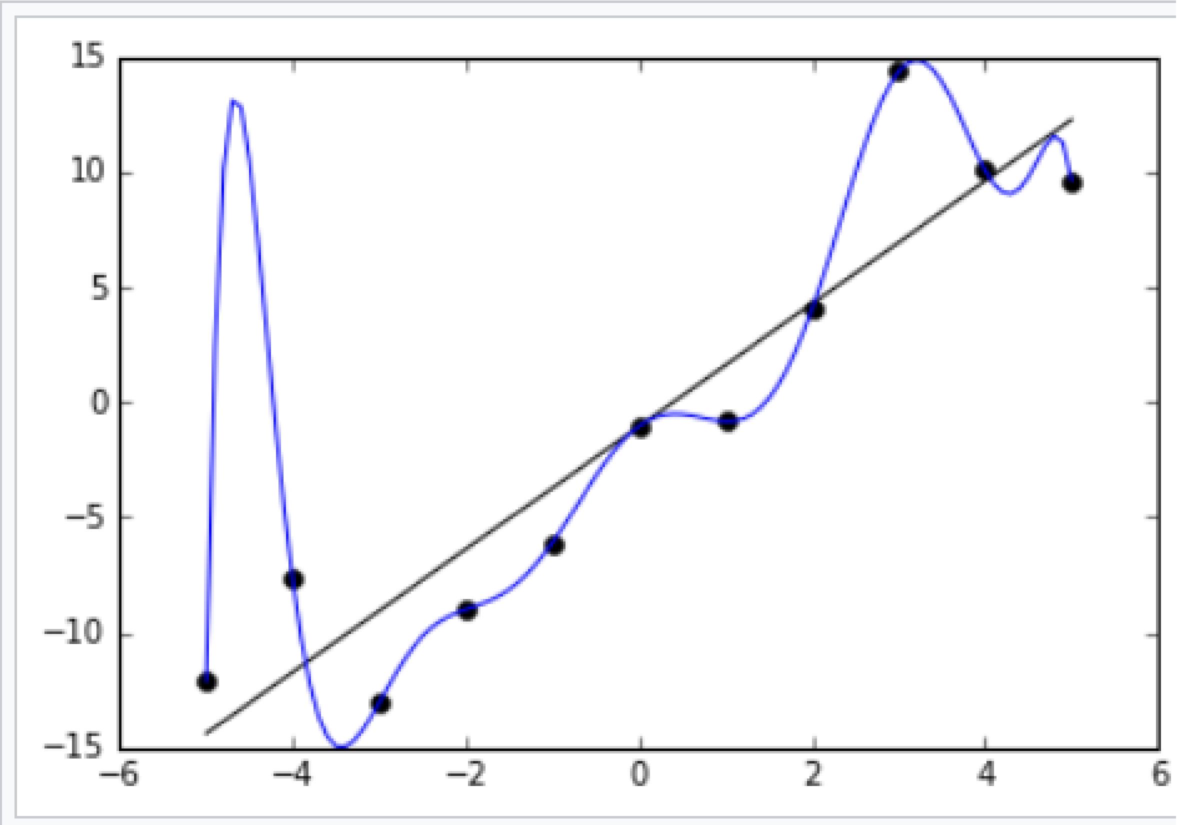
- We build the ML model based on some training dataset, which has many noises and outliers.
- An overly complex model may have very high accuracy for this training dataset, but very low accuracy in general.

Suppose the data points are all on the black straight line, but the training data points we used to build our model do not fall on the line because of noises. Our trained model (blue line) has serious overfitting problem:

(i) ‘fit’ extremely well for our training data, but

(ii) has very low accuracy in general (e.g., when x = -4.5, the correct label should be -13, but the model predicts 14..

A simple example of overfitting:



Fraud detection and prevention

- Choose the right model:
  - There are many good free ML learning tools available (e.g., from scikit-learn, TensorFlow)
  - These tools have their own strengths and weaknesses.
  - We need try a number of different model building algorithms, fine-tune their hyper-parameters so as to pick the right tool with good hyper-parameters for our application ➔ high “accuracy”, avoid overfitting (i.e., the complexity of the model is just right).

An example on ML fault prevention: Money Laundering

- The problem is difficult because
  - the offenders are legitimate users too,
  - just detect and give a ‘red flag’ for an abnormal transaction that has already happened may not help much; the user may have fled by the time the fraud is detected.
- We need to detect and prevent abnormal transactions from happening in real time.
  - We can use ML to learn those common characteristics (patterns) of the fraud transactions from money launderers and other hackers
  - For any new transaction, our machine learning tool detects these patterns automatically, and to alert us (e.g., tagging it with Redflag) immediately when it detects abnormal transactions.

The dataset

- Based on the following transaction file: `tranrecords.csv`
- The csv is for the “comma-separated-values” format

Amount	TranNo	Status	Departme	Fiscal Yea	Month	RedFlag

```
1599.5483533432368,C560169,Paid_UnReconciled,Finance,2015,Aug,1
1.161478786772198,C817105,Paid_UnReconciled,Purchasing,2018,Aug,0
11.885565040510526,C476755,Paid_Reconciled,Finance,2017,Apr,0
13.086409040014,E335726,Paid_Reconciled,R&D,2011,Aug,0
21.369041770353846,A818773,Paid_UnReconciled,R&D,2015,Jun,0
0.3106347611012756,C141146,Paid_UnReconciled,R&D,2018,Jan,0
3.355658400981397,E380954,Paid_Reconciled,Production,2012,Apr,0
2.14047021195408,E453358,Paid_Reconciled,Finance,2017,Aug,0
22.156267234847004,C641107,Paid_Reconciled,Finance,2013,Apr,0
0.79102969641633,:,Paid_UnReconciled,Purchasing,2016,Sep,0
151.3348287194061,C805852,Paid_UnReconciled,Marketing,2014,Aug,1
1.022413765092619,E977677,Paid_Reconciled,R&D,2015,Nov,0
0.04041556701330,C320147,Paid_Reconciled,Purchasing,2014,Sep,0
7.825946900244843,C156772,Paid_UnReconciled,Finance,2019,Apr,0
3.613603534637747,E908822,Paid_Reconciled,Finance,2018,Apr,0
6.50597223337363,C529606,Paid_Reconciled,Finance,:,Aug,0
2.96884571730533,E524327,Paid_Reconciled,Finance,2010,Apr,0
0.846074344629585,E761698,Paid_Reconciled,Finance,2010,Jun,0
```

The dataset

- Based on the following transaction file: `tranrecords.csv`
- The csv if for the “comma-separated-values” format

Amount	TranNo	Status	Departmei	Fiscal Yea	Month	RedFlag

```
1599.5483533432368,C560169,Paid_UnReconciled,Finance,2015,Aug,1
1.1614787867721998,C817105,Paid_UnReconciled,Purchasing,2018,Aug,0
11.885565040510526,C476755,Paid_Reconciled,Finance,2017,Apr,0
13.086409040014,E335726,Paid_Reconciled,R&D,2011,Aug,0
21.369041770353846,A818773,Paid_UnReconciled,R&D,2015,Jun,0
0.31063476110127564,C141146,Paid_UnReconciled,R&D,2018,Jan,0
3.3556584009813974,E380954,Paid_Reconciled,Production,2012,Apr,0
2.140470211954084,E453358,Paid_Reconciled,Finance,2017,Aug,0
22.156267234847004,C641107,Paid_Reconciled,Finance,2013,Apr,0
0.791029696416332,:,Paid_UnReconciled,Purchasing,2016,Sep,0
151.3348287194061,C805852,Paid_UnReconciled,Marketing,2014,Aug,1
1.0224137650926193,E977677,Paid_Reconciled,R&D,2015,Nov,0
0.040415567013301,C320147,Paid_Reconciled,Purchasing,2014,Sep,0
7.825946900244843,C156772,Paid_UnReconciled,Finance,2019,Apr,0
3.6136035346377477,E908822,Paid_Reconciled,Finance,2018,Apr,0
6.505972233373637,C529606,Paid_Reconciled,Finance,:,Aug,0
2.968845717305337,E524327,Paid_Reconciled,Finance,2010,Apr,0
0.8460743446295853,E761698,Paid_Reconciled,Finance,2010,Jun,0
```

Missing value

Libraries to be imported

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

from time import time
from sklearn.metrics import f1_score, accuracy_score
```

pandas has a simple method for reading cvs file

```
df = pd.read_csv("tranrecords.csv", na_values=':')
df
```

	Amount	TranNo	Status	Department	Fiscal Year	Month	RedFlag
0	12.467270	C023134	Paid_Reconciled	R&D	2014.0	Jan	0
1	0.455159	C486034	Paid_Reconciled	Purchasing	2015.0	Sep	0
2	1.058052	C432586	Paid_Reconciled	Marketing	2018.0	May	0
3	3.328995	C901026	Paid_UnReconciled	Finance	2012.0	May	0
4	1.777077	C328289	Paid_Reconciled	Finance	2012.0	Oct	0
...	...	...	...	...	...	...	...
2495	2.965170	A672539	Paid_Reconciled	Purchasing	2015.0	Aug	0
2496	3.053580	C860301	Paid_Reconciled	Finance	2016.0	Jun	0

pandas has a simple method for reading cvs file

```
df = pd.read_csv("tranrecords.csv", na_values=':')
df
```

	Amount	TranNo	Status	Department	Fiscal Year	Month	RedFlag
0	12.467270	C023134	Paid_Reconciled	R&D	2014.0	Jan	0
1	0.455159	C486034	Paid_Reconciled	Purchasing	2015.0	Sep	0
2	1.058052	C432586	Paid_Reconciled	Marketing	2018.0	May	0
3	3.328995	C901026	Paid_UnReconciled	Finance	2012.0	May	0
4	1.777077	C328289	Paid_Reconciled	Finance	2012.0	Oct	0
...	...	...	...	...	...	...	...
2495	2.965170	A672539	Paid_Reconciled	Purchasing	2015.0	Aug	0
2496	3.053580	C860301	Paid_Reconciled	Finance	2016.0	Jun	0

‘:’ represents missing values (not-available values)

What’s happened? Replace all ‘:’ by NaN

```
1599.5483533432368,C560169,Paid_UnReconciled,Finance,2015,Aug,1
1.1614787867721998,C817105,Paid_UnReconciled,Purchasing,2018,Aug,0
11.885565040510526,C476755,Paid_Reconciled,Finance,2017,Apr,0
13.086409040014,E335726,Paid_Reconciled,R&D,2011,Aug,0
21.369041770353846,A818773,Paid_UnReconciled,R&D,2015,Jun,0
0.31063476110127564,C141146,Paid_UnReconciled,R&D,2018,Jan,0
3.3556584009813974,E380954,Paid_Reconciled,Production,2012,Apr,0
2.140470211954084,E453358,Paid_Reconciled,Finance,2017,Aug,0
22.156267234847004,C641107,Paid_Reconciled,Finance,2013,Apr,0
0.791029696416382,:,Paid_UnReconciled,Purchasing,2016,Sep,0
151.3348287194061,C805852,Paid_UnReconciled,Marketing,2014,Aug,1
1.0224137650926193,E977677,Paid_Reconciled,R&D,2015,Nov,0
0.040415567013301,C320147,Paid_Reconciled,Purchasing,2014,Sep,0
7.825946900244843,C156772,Paid_UnReconciled,Finance,2019,Apr,0
3.6136035346377477,E908822,Paid_Reconciled,Finance,2018,Apr,0
6.505972233373637,C529606,Paid_Reconciled,Finance,:,Aug,0
2.968845717305337,E524327,Paid_Reconciled,Finance,2010,Apr,0
0.8460743446295853,E761698,Paid_Reconciled,Finance,2010,Jun,0
0.3767469630746701,C326197,Paid_UnReconciled,R&D,2015,:,0
```

19	0.310635	C141146	Paid_UnReconciled	R&D	2018.0	Jan	0
20	3.355658	E380954	Paid_Reconciled	Production	2012.0	Apr	0
21	2.140470	E453358	Paid_Reconciled	Finance	2017.0	Aug	0
22	22.156267	C641107	Paid_Reconciled	Finance	2013.0	Apr	0
23	0.791030	NaN	Paid_UnReconciled	Purchasing	2016.0	Sep	0
24	151.334829	C805852	Paid_UnReconciled	Marketing	2014.0	Aug	1
25	1.022414	E977677	Paid_Reconciled	R&D	2015.0	Nov	0
26	0.040416	C320147	Paid_Reconciled	Purchasing	2014.0	Sep	0
27	7.825947	C156772	Paid_UnReconciled	Finance	2019.0	Apr	0

What's happened? Replace all ':' by NaN

```
1599.5483533432368,C560169,Paid_UnReconciled,Finance,2015,Aug,1
1.1614787867721998,C817105,Paid_UnReconciled,Purchasing,2018,Aug,0
11.885565040510526,C476755,Paid_Reconciled,Finance,2017,Apr,0
13.086409040014,E335726,Paid_Reconciled,R&D,2011,Aug,0
21.369041770353846,A818773,Paid_UnReconciled,R&D,2015,Jun,0
0.31063476110127564,C141146,Paid_UnReconciled,R&D,2018,Jan,0
3.355658400981397 4,E380954,Paid_Reconciled,Production,2012,Apr,0
2.140470211954084,E453358,Paid_Reconciled,Finance,2017,Aug,0
22.156267234847004,C641107,Paid_Reconciled,Finance,2013,Apr,0
0.7910296964163 32,:,Paid_UnReconciled,Purchasing,2016,Sep,0
151.334828719406 1,C805852,Paid_UnReconciled,Marketing,2014,Aug,1
1.0224137650926193,E977677,Paid_Reconciled,R&D,2015,Nov,0
0.040415567013301,C320147,Paid_Reconciled,Purchasing,2014,Sep,0
7.825946900244843,C156772,Paid_UnReconciled,Finance,2019,Apr,0
3.6136035346377477,E908822,Paid_Reconciled,Finance,2018,Apr,0
6.505972233373637,C529606,Paid_Reconciled,Finance,:,Aug,0
2.968845717305337,E524327,Paid_Reconciled,Finance,2010,Apr,0
0.8460743446295853,E761698,Paid_Reconciled,Finance,2010,Jun,0
0.3767469630746701,C326197,Paid_UnReconciled,R&D,2015,:,0
```

19	0.310635	C141146	Paid_UnReconciled				
20	3.355658	E380954	Paid_Reconciled				
21	2.140470	E453358	Paid_Reconciled				
22	22.156267	C641107	Paid_Reconciled				
23	0.791030	NaN	Paid_UnReconciled				
24	151.334829	C805852	Paid_UnReconciled				
25	1.022414	E977677	Paid_Reconciled	R&D	2015.0	Nov	0
26	0.040416	C320147	Paid_Reconciled	Purchasing	2014.0	Sep	0
27	7.825947	C156772	Paid_UnReconciled	Finance	2019.0	Apr	0

Why do we need to convert ':' to Nan?
Because:
Pandas has many useful methods for us to detect and handle these NaN values.
More details later.

Know your data and clean your data

- Look at the “first” and “tail” of the data frame

```
df.head(10)
```

	Amount	TranNo	Status	Department	Fiscal Year	Month	RedFlag
0	12.467270	C023134	Paid_Reconciled	R&D	2014.0	Jan	0
1	0.455159	C486034	Paid_Reconciled	Purchasing	2015.0	Sep	0
2	1.058052	C432586	Paid_Reconciled	Marketing	2018.0	May	0
3	3.328995	C901026	Paid_UnReconciled	Finance	2012.0	May	0
4	1.777077	C328289	Paid_Reconciled	Finance	2012.0	Oct	0
5	0.113475	E876463	Paid_Reconciled	Marketing	2018.0	May	0
6	0.130126	C525492	Paid_Reconciled	Finance	2014.0	Jan	0
7	6.957715	C579803	Paid_Reconciled	Marketing	2013.0	Apr	0
8	4.222575	C422593	Paid_Reconciled	Marketing	2017.0	May	0
9	43.424134	C194584	Paid_UnReconciled	Finance	2019.0	Apr	0

…

```
df.tail(10)
```

	Amount	TranNo	Status	Department	Fiscal Year	Month	RedFlag
2490	0.666586	C992476	Paid_Reconciled	Finance	2018.0	Sep	0
2491	93.995399	C848300	Paid_Reconciled	Finance	2018.0	Sep	1
2492	75.463958	C408953	Paid_UnReconciled	Purchasing	2011.0	Apr	1
2493	282.203496	C988040	Paid_Reconciled	Finance	2014.0	Sep	1
2494	47.903434	C465992	Paid_UnReconciled	Finance	2018.0	Jul	0
2495	2.965170	A672539	Paid_Reconciled	Purchasing	2015.0	Aug	0
2496	3.053580	C860301	Paid_Reconciled	Finance	2016.0	Jun	0
2497	79.006232	E064806	Paid_Reconciled	Purchasing	2016.0	May	1
2498	262.263151	C741867	Paid_UnReconciled	Marketing	2011.0	Feb	1
2499	14.580766	C177150	Paid_Reconciled	Finance	2011.0	Sep	0

Know your data and clean your data

- Look at the “first” and “tail” of the data frame

```
df.head(10)
```

	Amount	TranNo	Status	Department	Fiscal Year	Month	RedFlag
0	12.467270	C023134	Paid_Reconciled	R&D	2014.0	Jan	0
1	0.455159	C486034	Paid_Reconciled	Purchasing	2015.0	Sep	0
2	1.058052	C432589	Paid_Reconciled	Marketing	2018.0	May	0
3	3.328995	C901026	Paid_UnReconciled	Finance	2012.0	May	0
4	1.777077	C328289	Paid_Reconciled	Finance	2012.0	Oct	0
5	0.113475	E876463	Paid_Reconciled	Marketing	2018.0	May	0
6	0.130126	C525492	Paid_Recon				
7	6.957715	C579803	Paid_Recon				
8	4.222575	C422593	Paid_Recon				
9	43.424134	C194584	Paid_UnRecon				

```
df.tail(10)
```

	Amount	TranNo	Status	Department	Fiscal Year	Month	RedFlag
2490	0.666586	C992476	Paid_Reconciled	Finance	2018.0	Sep	0
2491	93.995399	C848300	Paid_Reconciled	Finance	2018.0	Sep	1
2492	75.463958	C408953	Paid_UnReconciled	Purchasing	2011.0	Apr	1
2493	282.203496	C988040	Paid_Reconciled	Finance	2014.0	Sep	1
2494	47.903434	C465992	Paid_UnReconciled	Finance	2018.0	Jul	0
2495	2.965170	A672539	Paid_Reconciled	Purchasing	2015.0	Aug	0
							0
							1
							1
							0

Payment reconciliation is a financial process that involves matching and comparing transaction records to ensure that the payments made or received are accurate and consistent with what is recorded in the business's accounting books or financial statements. This process is essential for verifying the accuracy of financial transactions, avoiding errors or discrepancies, and maintaining the integrity of financial records.

Know your data and clean your data

- Have some summaries of your data:

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2500 entries, 0 to 2499
Data columns (total 7 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   Amount       2500 non-null   float64
 1   TranNo       2456 non-null   object
 2   Status       2459 non-null   object
 3   Department   2486 non-null   object
 4   Fiscal Year  2445 non-null   float64
 5   Month        2478 non-null   object
 6   RedFlag      2500 non-null   int64
dtypes: float64(2), int64(1), object(4)
memory usage: 136.8+ KB
```

i.e., not NaN values

```
df.describe()
```

	Amount	Fiscal Year	RedFlag
count	2500.000000	2445.000000	2500.000000
mean	33.218551	2014.549284	0.130800
std	160.001246	2.925515	0.337249
min	0.001359	2010.000000	0.000000
25%	1.176832	2012.000000	0.000000
50%	4.672201	2014.000000	0.000000
75%	19.919945	2017.000000	0.000000
max	4060.364437	2019.000000	1.000000

Know your data and clean your data

Find the number of NaN values in every column:

```
df.isnull()
```

	Amount	TranNo	Status	Department	Fiscal Year	Month	RedFlag
0	False	False	False	False	False	False	False
1	False	False	False	False	False	False	False
2	False	False	False	False	False	False	False
3	False	False	False	False	False	False	False
4	False	False	False	False	False	False	False
...	...	...	...	...	...	...	...
2495	False	False	False	False	False	False	False
2496	False	False	False	False	False	False	False
2497	False	False	False	False	False	False	False
2498	False	False	False	False	False	False	False
2499	False	False	False	False	False	False	False

2500 rows × 7 columns

```
df.isnull().sum(axis=0)
```

```
Amount          0
TranNo         44
Status         41
Department     14
Fiscal Year    55
Month          22
RedFlag         0
dtype: int64
```

sum along the row indices

Know your data and clean your data

- Before further exploring your data, we need to handle these NaN values first.
- For numerical data, we can replace the Nan values by the mean of “normal” values in that column

Know your data and clean your data

- To find the mean of each (numeric) columns: df.mean()
- To replace the Nan values by this mean: df.fillna(df.mean())

```
df = df.fillna(df.mean())
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2500 entries, 0 to 2499
Data columns (total 7 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   Amount       2500 non-null   float64
 1   TranNo       2456 non-null   object
 2   Status       2459 non-null   object
 3   Department   2486 non-null   object
 4   Fiscal Year  2500 non-null   float64
 5   Month        2478 non-null   object
 6   RedFlag      2500 non-null   int64
dtypes: float64(2), int64(1), object(4)
memory usage: 136.8+ KB
```

object values have no mean, thus there are still NaN values in these columns

Know your data and clean your data

- To drop (i.e., delete) all the rows with Nan values in df:

```
df = df.dropna()
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 2381 entries, 0 to 2499
Data columns (total 7 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   Amount       2381 non-null   float64
 1   TranNo       2381 non-null   object
 2   Status       2381 non-null   object
 3   Department   2381 non-null   object
 4   Fiscal Year  2381 non-null   float64
 5   Month        2381 non-null   object
 6   RedFlag      2381 non-null   int64
dtypes: float64(2), int64(1), object(4)
memory usage: 148.8+ KB
```

Know your data and clean your data

Some better method for handling Nan_values:

Using sklearn’s class SimpleImputer

Know your data and clean your data

Using sklearn's class SimpleImputer

For numeric columns: We find from df.info() that only the column 'Fiscal Year' has Nan-value.

```
from sklearn.impute import SimpleImputer
```

```
imr = SimpleImputer(missing_values=np.nan, strategy='mean')
df['Fiscal Year'] =imr.fit_transform(df['Fiscal Year'].values.reshape(-1,1)).reshape(-1,)
```

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2500 entries, 0 to 2499
Data columns (total 7 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   Amount       2500 non-null   float64
 1   TranNo       2456 non-null   object
 2   Status       2459 non-null   object
 3   Department   2486 non-null   object
 4   Fiscal Year  2500 non-null   float64
 5   Month        2478 non-null   object
 6   RedFlag      2500 non-null   int64
dtypes: float64(2), int64(1), object(4)
memory usage: 136.8+ KB
```

Know your data and clean your data

Using sklearn's class SimpleImputer

For numeric columns: We find from df.info() that only the column 'Fiscal Year' has Nan-value.

```
from sklearn.impute import SimpleImputer
```

```
imr = SimpleImputer(missing_values=np.nan, strategy='mean')
df['Fiscal Year'] =imr.fit_transform(df['Fiscal Year'].values.reshape(-1,1)).reshape(-1,)
```

Note that this df is the original one, not df we got after calling fillna() & dropna() in the previous two slides.

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2500 entries, 0 to 2499
Data columns (total 7 columns):
 #   Column       Non-Null Count  Dtype  
---  ------       --------------  -----  
 0   Amount       2500 non-null   float64
 1   TranNo       2456 non-null   object 
 2   Status       2459 non-null   object 
 3   Department   2486 non-null   object 
 4   Fiscal Year  2500 non-null   float64
 5   Month        2478 non-null   object 
 6   RedFlag      2500 non-null   int64  
dtypes: float64(2), int64(1), object(4)
memory usage: 136.8+ KB
```

Know your data and clean your data

- For “object” columns

```
imr = SimpleImputer(missing_values=np.nan, strategy='most_frequent')
df[['TranNo', 'Status', 'Department', 'Month']] = \
    imr.fit_transform(df[['TranNo', 'Status', 'Department', 'Month']])
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2500 entries, 0 to 2499
Data columns (total 7 columns):
 #   Column       Non-Null Count  Dtype  
---  ------       --------------  -----  
 0   Amount       2500 non-null   float64
 1   TranNo       2500 non-null   object 
 2   Status       2500 non-null   object 
 3   Department   2500 non-null   object 
 4   Fiscal Year  2500 non-null   float64
 5   Month        2500 non-null   object 
 6   RedFlag      2500 non-null   int64  
dtypes: float64(2), int64(1), object(4)
memory usage: 136.8+ KB
```

Know your data and clean your data

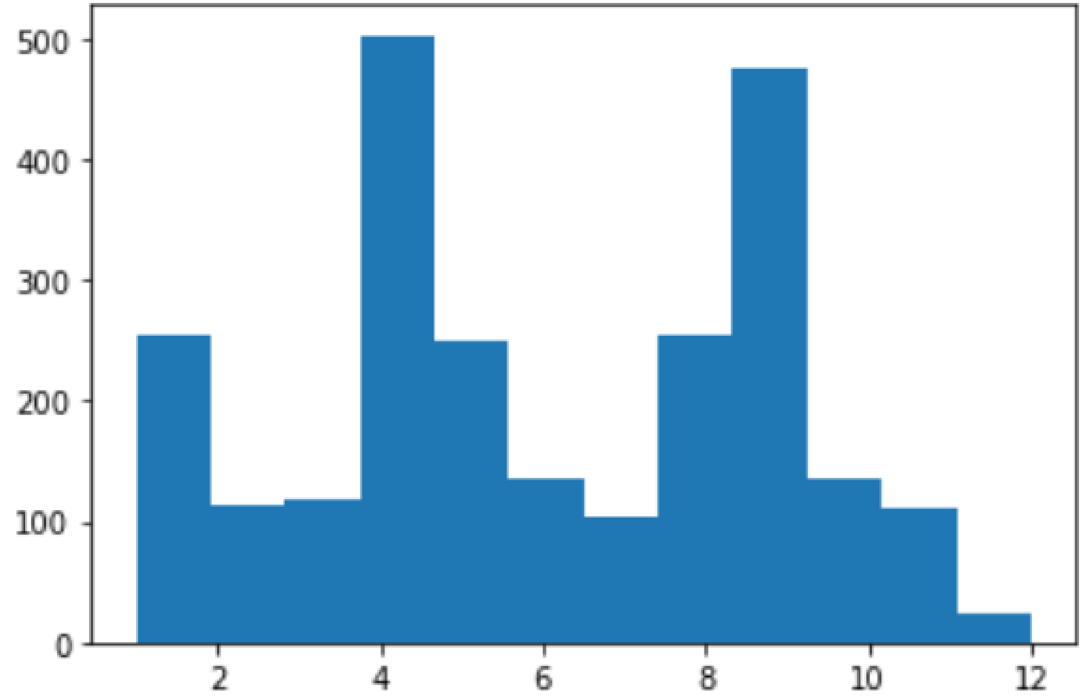
SimpleImputer() has other strategies:

The imputation strategy.

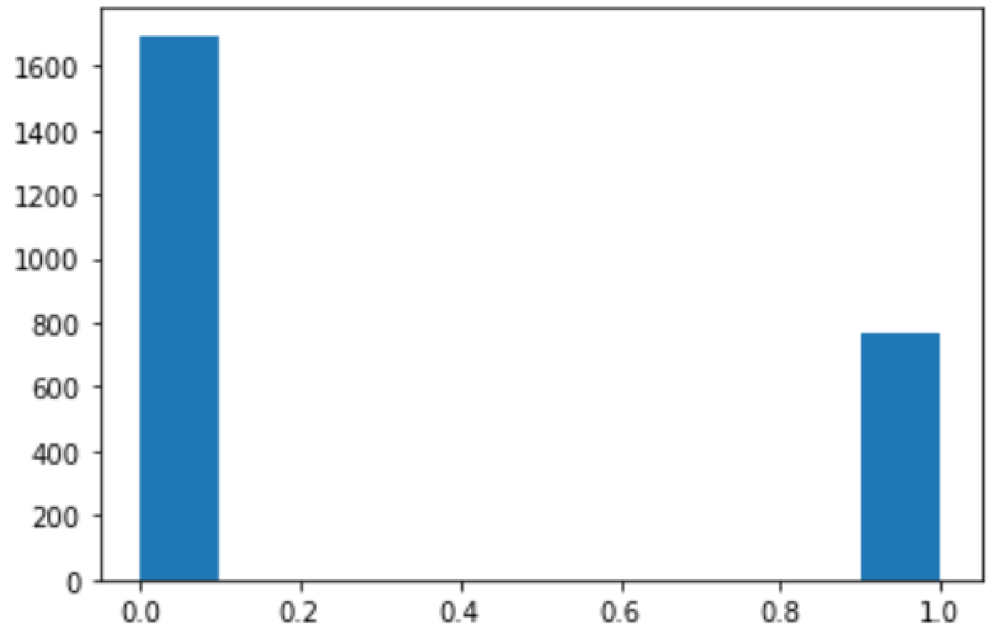
- If “mean”, then replace missing values using the mean along each column. Can only be used with numeric data.
- If “median”, then replace missing values using the median along each column. Can only be used with numeric data.
- If “most_frequent”, then replace missing using the most frequent value along each column. Can be used with strings or numeric data. If there is more than one such value, only the smallest is returned.
- If “constant”, then replace missing values with fill_value. Can be used with strings or numeric data.
- If an instance of Callable, then replace missing values using the scalar statistic returned by running the callable over a dense 1d array containing non-missing values of each column.

Know your data: distributions of data

```
pic = plt.hist(df['Month'],bins=12)
```

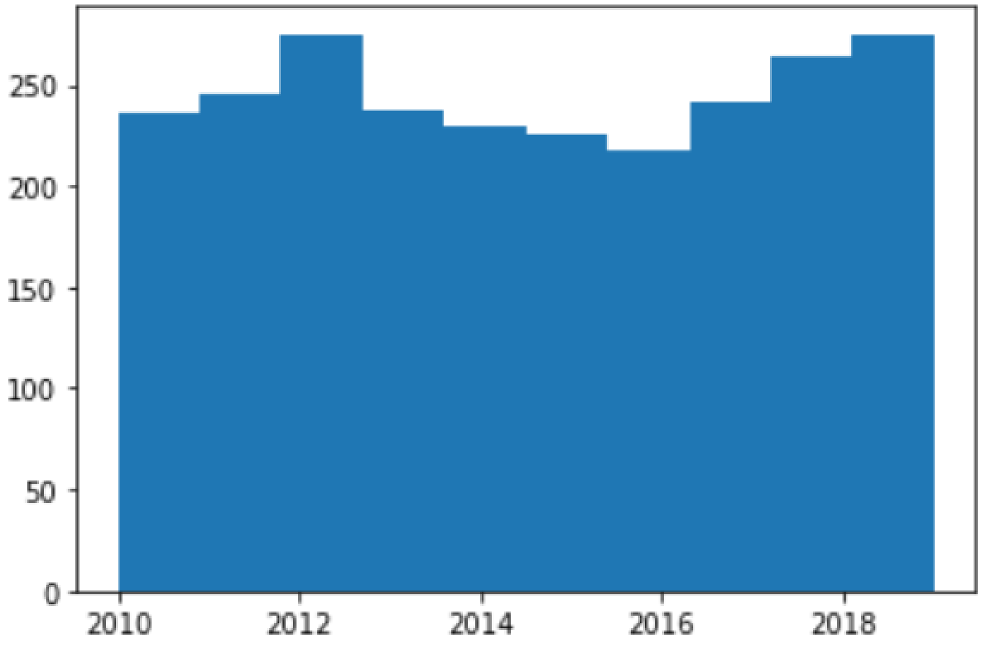



```
pic = plt.hist(df['Status'])
```

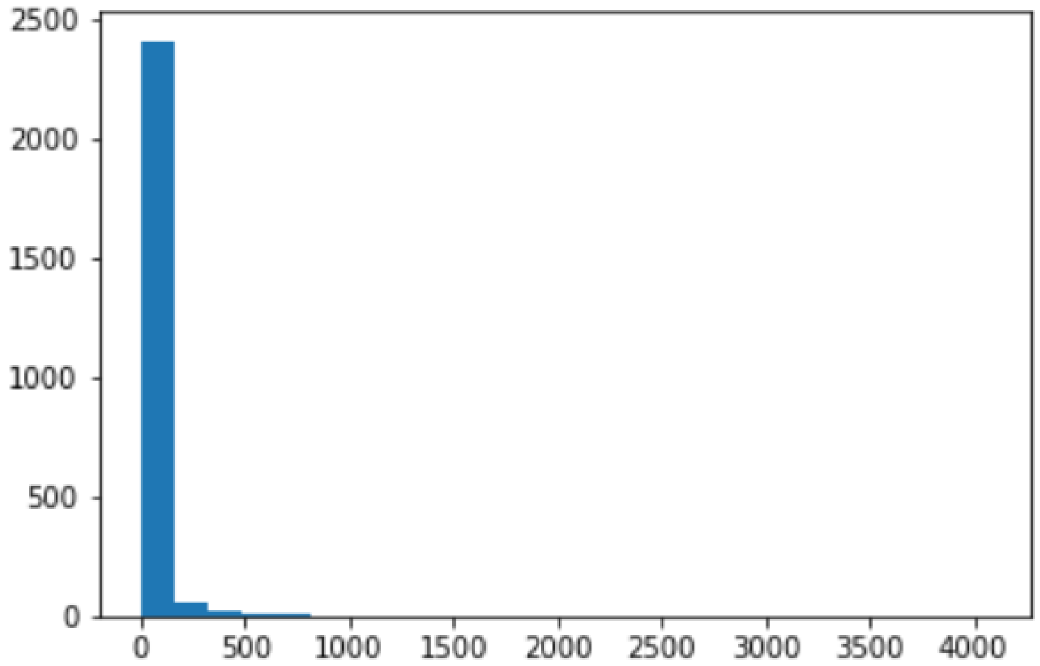


Know your data set: distributions of data

```
pic = plt.hist(df['Fiscal Year'], bins=10)
```

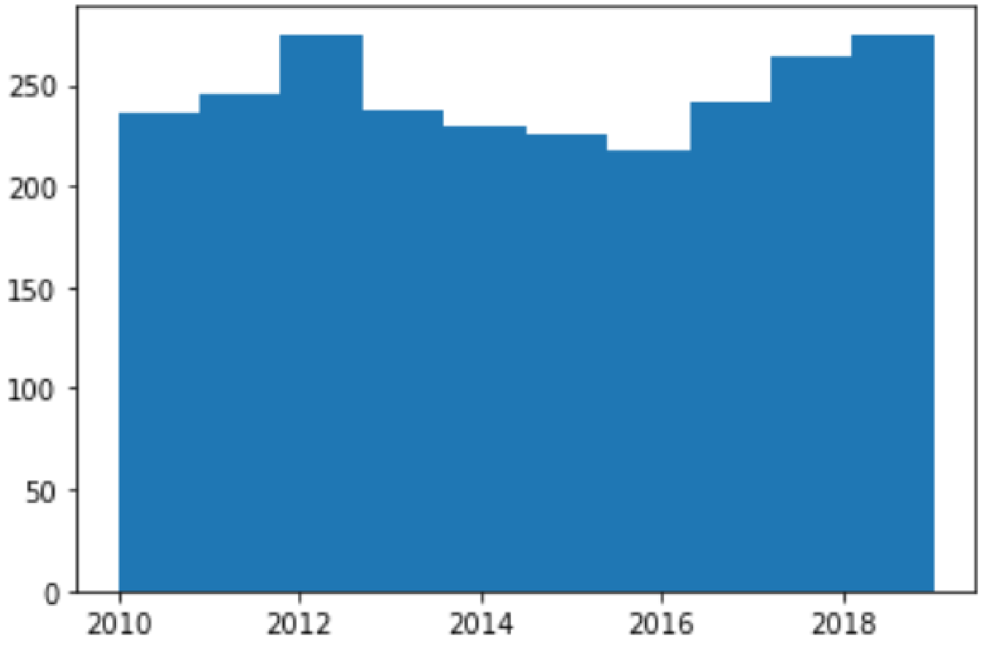



```
pic = plt.hist(df['Amount'],bins=25)
```

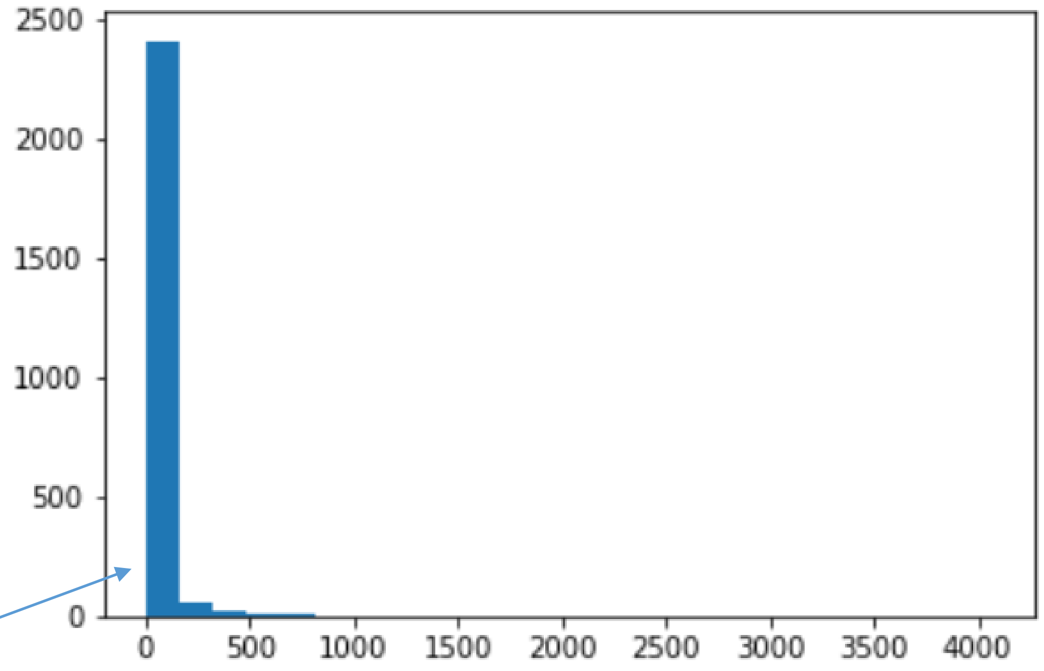


Know your data set: distributions of data

```
pic = plt.hist(df['Fiscal Year'], bins=10)
```




```
pic = plt.hist(df['Amount'],bins=25)
```

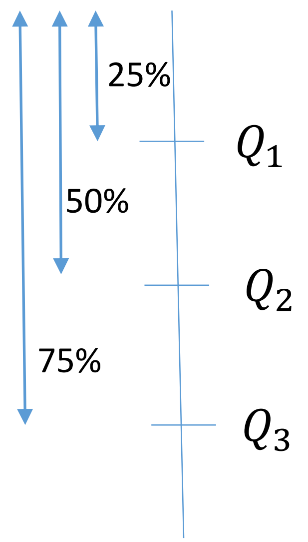


Not normal.
Why?

Know your data: detecting outliers

- Percentiles: For any 0 ≤p≤100, the pth-percentile of a set S of values is the value x_p in S such that
  - a fraction of p of the data values in S are smaller than or equal to x_p, and
  - the remaining fraction (1-p) is greater than or equal to x_p.

  For example, the median of S is the 50th percentile of S.



- Quantiles:
  - The first quantile of S (Q_1)= the 25th percentile of S
  - The second quantile of S (Q_2)= the median of S = the 50th percentile of S
  - The third quantile of S (Q_3)= the 75th percentile of S
- 5-number summary: x_{min} , Q_1 , Q_2 , Q_3 , x_{max}

Know your data: detecting outliers

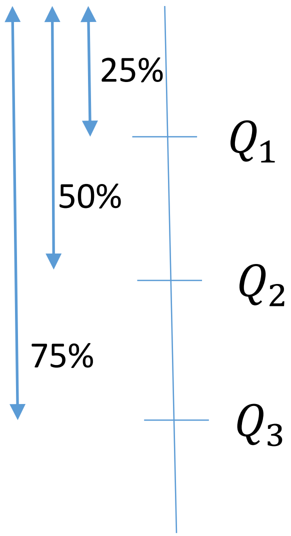
```
dfa = df['Amount']
dfa.min(), dfa.quantile(q=0.25), dfa.quantile(q=0.5), \
    dfa.quantile(q=0.75), dfa.max()
```

```
(0.0013588035896111,
 1.176832215065768 8,
 4.67220094725914 1,
 19.91994521858257 6,
 4060.364436505363)
```

e of a set S of values is

n or equal to x_p, and

tile of S.



- The second quantile of S (Q_2)= the median of S = the 50th percentile of S
- The third quantile of S (Q_3)= the 75th percentile of S

• 5-number summary: x_{min} , Q_1 , Q_2 , Q_3 , x_{max}

Know your data: detecting outliers

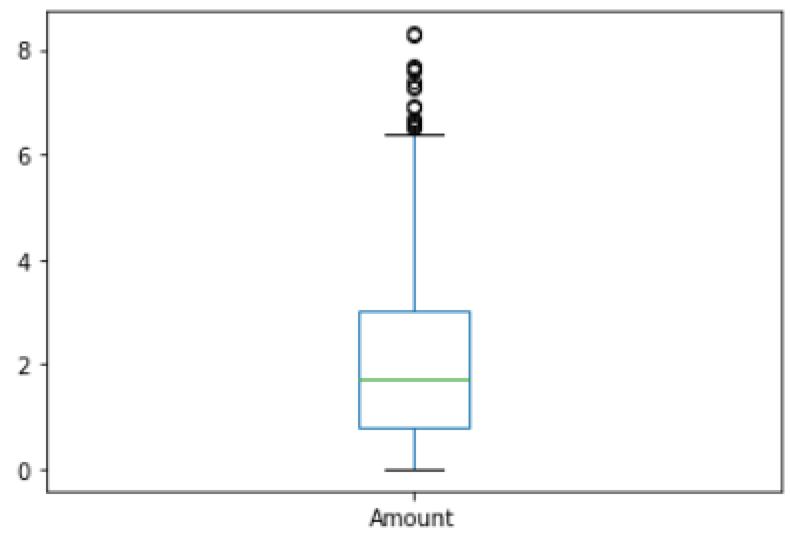
`DataFrame.plot.box(by=None, **kwargs)` [source]

Make a box plot of the DataFrame columns.

A box plot is a method for graphically depicting groups of numerical data through their quartiles. The box extends from the Q1 to Q3 quartile values of the data, with a line at the median (Q2). The whiskers extend from the edges of box to show the range of the data. The position of the whiskers is set by default to 1.5*IQR (IQR = Q3 - Q1) from the edges of the box. Outlier points are those past the end of the whiskers.

```
dfa.plot.box()
```

<AxesSubplot:>



Cleaning your data: detecting outliers

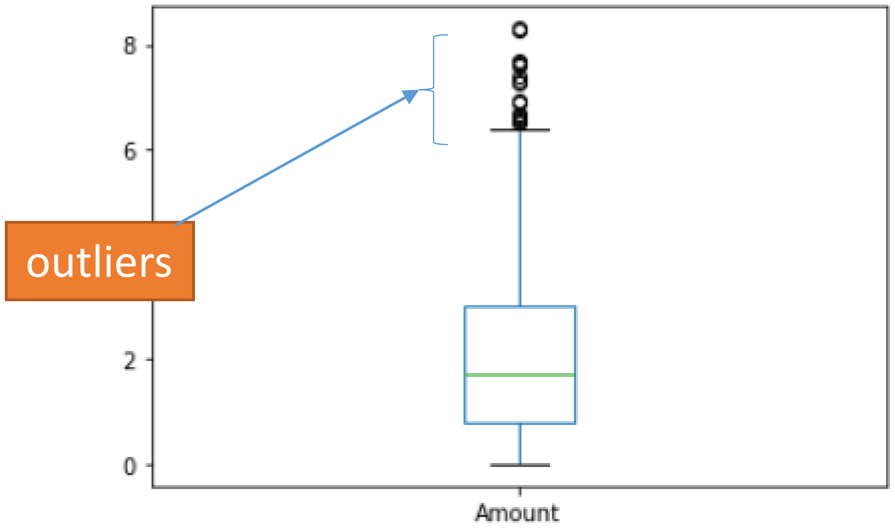
DataFrame.plot.box(*by=None*, ***kwargs*) [source]

Make a box plot of the DataFrame columns.

A box plot is a method for graphically depicting groups of numerical data through their quartiles. The box extends from the Q1 to Q3 quartile values of the data, with a line at the median (Q2). The whiskers extend from the edges of box to show the range of the data. The position of the whiskers is set by default to 1.5*IQR (IQR = Q3 - Q1) from the edges of the box. Outlier points are those past the end of the whiskers.

```
dfa.plot.box()
```

<AxesSubplot:>

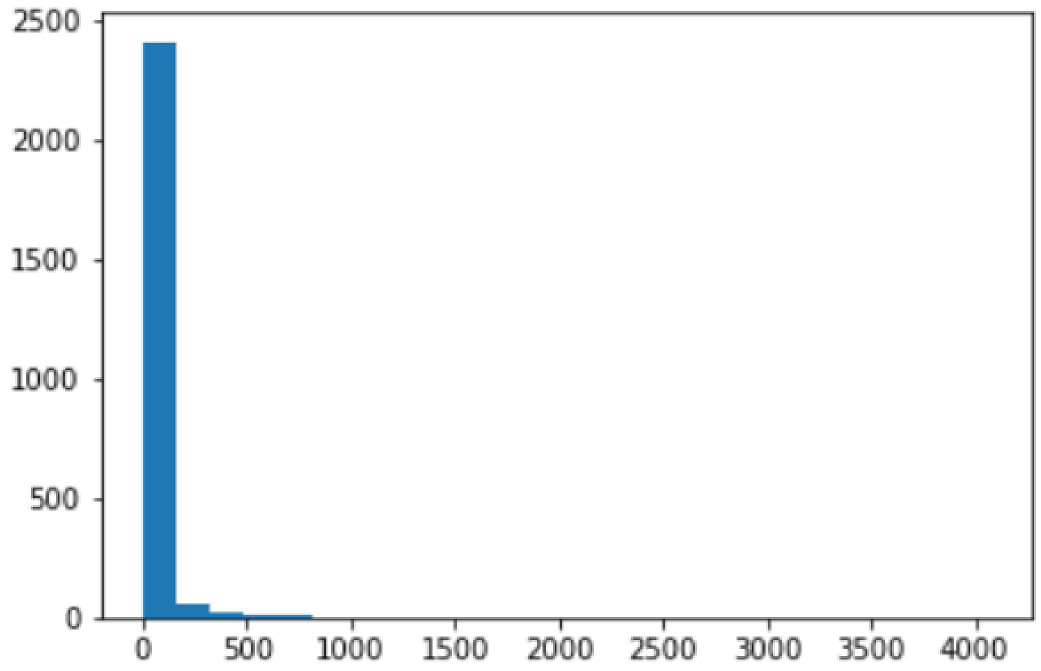


Clean your data: How to deal with outliers

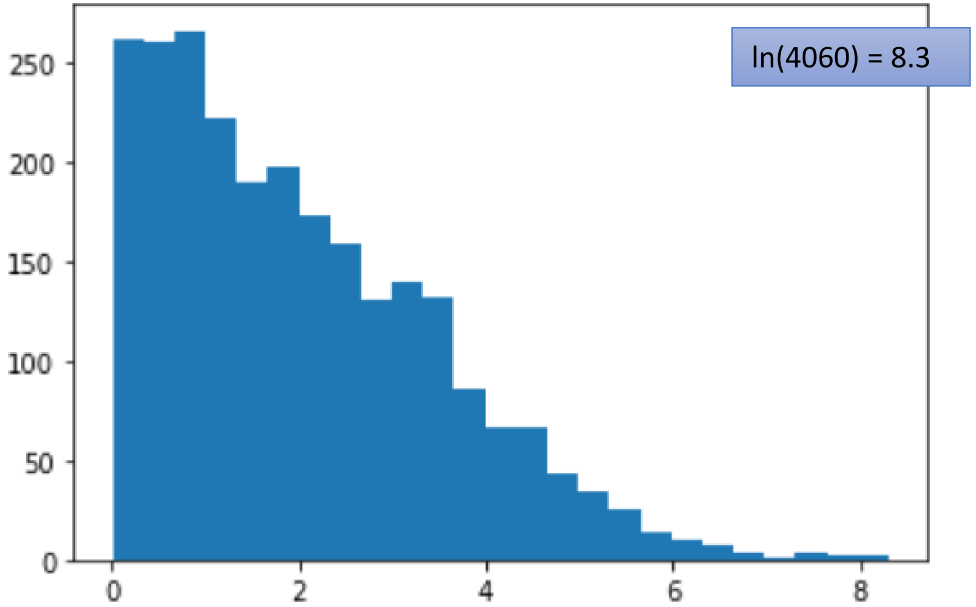
Option 1: Throw them away.

Option 2: Reduce their impact (scaling their values down. Eg. use log-transformation.

```
pic = plt.hist(df['Amount'],bins=25)
```




```
df['Amount'] = df['Amount'].apply(lambda x: np.log(x+1))
pic = plt.hist(df['Amount'], bins=25)
```

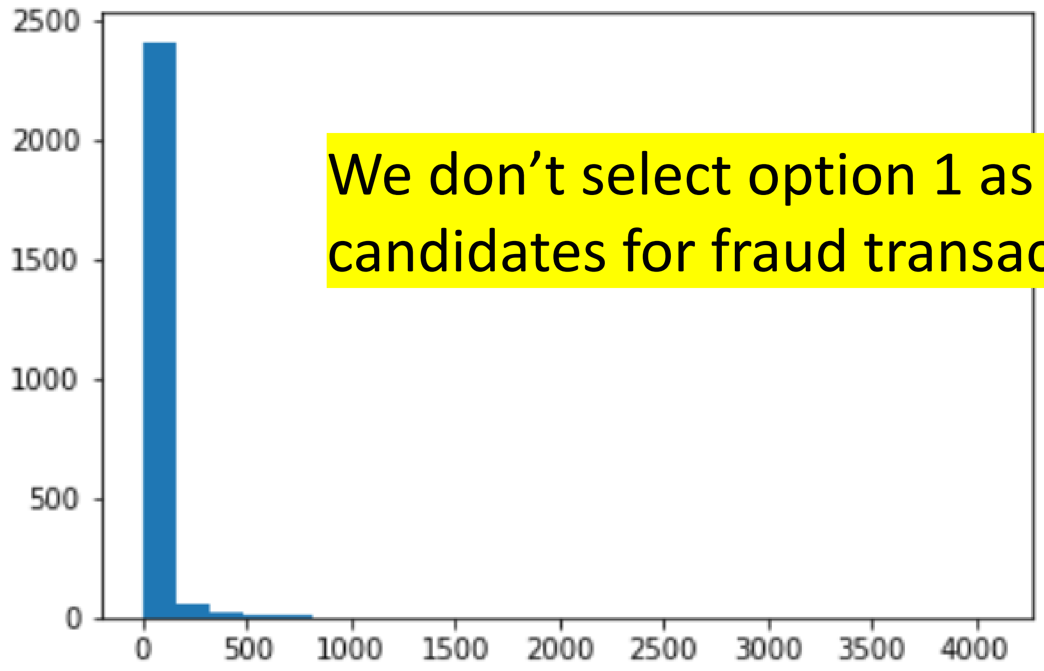


Clean your data: How to deal with outliers

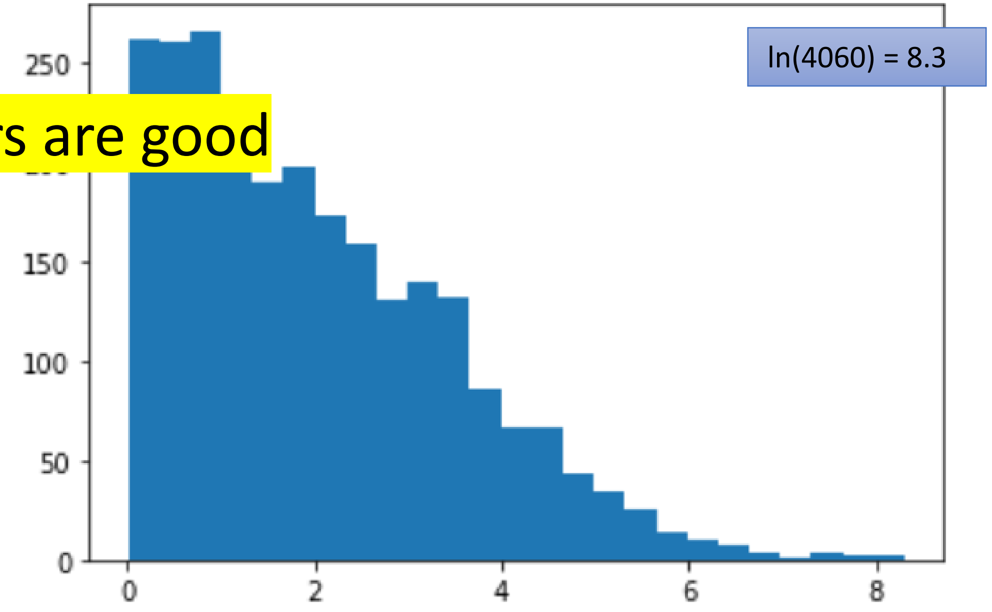
Option 1: Throw them away.

Option 2: Reduce their impact (scaling their values down). Eg. use log-transformation.

```python
pic = plt.hist(df['Amount'],bins=25)
```




```python
df['Amount'] = df['Amount'].apply(lambda x: np.log(x+1))
pic = plt.hist(df['Amount'], bins=25)
```



We don’t select option 1 as outliers are good candidates for fraud transactions.

Features encoding

- In our data set, there are many columns with type "object". In fact they are character strings. We say that these columns containing categorical data.
- Categorical data is a type of data that represents categories or distinct groups rather than numerical values. It is used to classify items or classes based on qualitative characteristics. These categories are often mutually exclusive and do not have a natural order or numerical value associated with them.
- In supervised learning, the labels are categorial data.
- But ML model learning methods require numeric inputs. Thus, in prepare our dataset, we need to convert (encode) categorical features into a small set of integers (e.g., 0, 1, 2, …, k).

Feature encoding

20	3.355658	E380954	Paid_Reconciled	Production	2012.0	Apr	0
21	2.140470	E453358	Paid_Reconciled	Finance	2017.0	Aug	0
22	22.156267	C641107	Paid_Reconciled	Finance	2013.0	Apr	0
23	0.791030	NaN	Paid_UnReconciled	Purchasing	2016.0	Sep	0
24	151.334829	C805852	Paid_UnReconciled	Marketing	2014.0	Aug	1
25	1.022414	E977677	Paid_Reconciled	R&D	2015.0	Nov	0
26	0.040416	C320147	Paid_Reconciled	Purchasing	2014.0	Sep	0
27	7.825947	C156772	Paid_UnReconciled	Finance	2019.0	Apr	0
28	3.613604	E908822	Paid_Reconciled	Finance	2018.0	Apr	0
29	6.505972	C529606	Paid_Reconciled	Finance	NaN	Aug	0
30	2.968846	E524327	Paid_Reconciled	Finance	2010.0	Apr	0
31	0.846074	E761698	Paid_Reconciled	Finance	2010.0	Jun	0
32	0.376747	C326197	Paid_UnReconciled	R&D	2015.0	NaN	0
33	16.220062	C350403	Paid_Reconciled	Purchasing	2012.0	Apr	0
34	33.547451	E505968	Paid_Reconciled	Purchasing	2016.0	Apr	0
35	32.696441	C893015	Paid_Reconciled	Finance	2014.0	Apr	0
36	0.202472	A348537	Paid_Reconciled	Finance	2018.0	Sep	0
37	100.950884	C294574	Paid_Reconciled	Marketing	NaN	Sep	1
38	0.283136	C551366	Paid_Reconciled	Finance	2011.0	Apr	0

```
d = {"Jan":1, "Feb":2, "Mar":3, "Apr":4, "May":5,
     "Jun":6, "Jul":7, "Aug":8,
    "Sep":9, "Oct":10, "Nov":11, "Dec":12}
df['Month'] = df['Month'].map(d)
df['Month'].head(10)
```

	Month
0	1
1	9
2	5
3	5
4	10
5	5
6	1
7	4
8	5
9	4

dtype: int64

Feature encoding

20	3.355658	E380954	Paid_Reconciled	Production	2012.0	Apr	0
21	2.140470	E453358	Paid_Reconciled	Finance	2017.0	Aug	0
22	22.156267	C641107	Paid_Reconciled	Finance	2013.0	Apr	0
23	0.791030	NaN	Paid_UnReconciled	Purchasing	2016.0	Sep	0
24	151.334829	C805852	Paid_UnReconciled	Marketing	2014.0	Aug	1
25	1.022414	E977677	Paid_Reconciled	R&D	2015.0	Nov	0
26	0.040416	C320147	Paid_Reconciled	Purchasing	2014.0	Sep	0
27	7.825947	C156772	Paid_UnReconciled	Finance	2019.0	Apr	0
28	3.613604	E908822	Paid_Reconciled	Finance	2018.0	Apr	0
29	6.505972	C529606	Paid_Reconciled	Finance	NaN	Aug	0
30	2.968846	E524327	Paid_Reconciled	Finance	2010.0	Apr	0
31	0.846074	E761698	Paid_Reconciled	Finance	2010.0	Jun	0
32	0.376747	C326197	Paid_UnReconciled	R&D	2015.0	NaN	0
33	16.220062	C350403	Paid_Reconciled	Purchasing	2012.0	Apr	0
34	33.547451	E505968	Paid_Reconciled	Purchasing	2016.0	Apr	0
35	32.696441	C893015	Paid_Reconciled	Finance	2014.0	Apr	0
36	0.202472	A348537	Paid_Reconciled	Finance	2018.0	Sep	0
37	100.950884	C294574	Paid_Reconciled	Marketing	NaN	Sep	1
38	0.283136	C551366	Paid_Reconciled	Finance	2011.0	Apr	0

```python
d = {"Paid_Reconciled":0, "Paid_UnReconciled":1}
df['Status'] = df['Status'].map(d)
df['Status'].head(10)
```

	Status
0	0
1	0
2	0
3	1
4	0
5	0
6	0
7	0
8	0
9	1

dtype: int64

Feature scaling

- We also need to do something to the numerical columns.
- The majority of ML learning algorithm (e.g. linear regression, SVM, or neural network) behave much better if features are of the same scale.
- There are two common approaches to bringing different features onto the same scale:
  - standardization: $x_{std}^{(i)} = \frac{x^{(i)} - \mu_x}{\sigma_x}$ where μ_x and σ_x are the mean and the

    standard deviation of of the values in that feature column.
  - normalization (min-max scaling): $x_{norm}^{(i)} = \frac{x^{(i)} - x_{min}}{x_{max} - x_{min}}$

Feature scaling

- For standardization:

```
from sklearn.preprocessing import StandardScaler
stdsc = StandardScaler()
X_train_std = stdsc.fit_transform(X_train)
X_test_std = stdsc.transform(X_test)
```

- For normalization: just replace “StandardScaler” by “MinMaxScaler” in the above example.

Do Feature encoding and scaling together

- Using sklearn's ColumnTransformer

```
data = df.copy()
```

```
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import OrdinalEncoder, MinMaxScaler

ct = ColumnTransformer([
    ('minmax', MinMaxScaler(), [0]),
    ('ordinal', OrdinalEncoder(), [1, 2, 3 ,4, 5]),
    ('nothing', 'passthrough', [6])
])
data.iloc[:,:] = ct.fit_transform(data.values)
data
```

Do Feature encoding and scaling together

- Using sklearn's ColumnTransformer

```
data = df.copy()
```

```
from sklearn.compose import C
from sklearn.preprocessing im

ct = ColumnTransformer([
    ('minmax', MinMaxScaler()
    ('ordinal', OrdinalEncode
    ('nothing', 'passthrough'
])
data.iloc[:,:] = ct.fit_trans
data
```

	Amount	TranNo	Status	Department	Fiscal Year	Month	RedFlag
0	0.003070	267.0	0.0	4.0	4.0	4.0	0
1	0.000112	935.0	0.0	3.0	6.0	11.0	0
2	0.000260	863.0	0.0	1.0	9.0	8.0	0
3	0.000820	1595.0	1.0	0.0	2.0	8.0	0
4	0.000437	700.0	0.0	0.0	2.0	10.0	0
...	...	...	...	...	...	...	...
2495	0.000730	145.0	0.0	3.0	6.0	1.0	0
2496	0.000752	1532.0	0.0	0.0	7.0	6.0	0
2497	0.019458	1782.0	0.0	3.0	7.0	8.0	1
2498	0.064591	1340.0	1.0	1.0	1.0	3.0	1
2499	0.003591	484.0	0.0	0.0	1.0	11.0	0

Feature selection

- We compute, for each column, the compute the *covariance* of this column and the output column (i.e., the RedFlag column).

```
np.cov(df['Amount'].fillna(df['Amount'].mean()), df['RedFlag'])
```

```
array([[2.22031009, 0.31095493],
       [0.31095493, 0.11373685]])
```

X, Y
Cov(X, X) (aka Variance(X)
Cov(X, Y)

Cov(Y, X)
Cov(Y, Y)

Note: Cov(X,Y) = Cov(Y,X)

- x_i = a given x value in the data set
- x_m = the mean, or average, of the x values
- y_i = the y value in the data set that corresponds with x_i
- y_m = the mean, or average, of the y values
- n = the number of data points

Given this information, the formula for covariance is: **Cov(x,y) = SUM [(x_i - x_m) * (y_i - y_m)] / (n - 1)**

Feature selection

```
np.cov(df['Amount'].fillna(df['Amount'].mean()), df['RedFlag'])
```

```
array([[2.22031009, 0.31095493],
       [0.31095493, 0.11373685]])
```

```
np.cov(df['Month'].fillna(df['Amount'].mean()), df['RedFlag'])
```

```
array([[9.23587379, 0.02772836],
       [0.02772836, 0.11373685]])
```

```
np.cov(df['Status'].fillna(df['Status'].mean()), df['RedFlag'])
```

```
array([[0.21073578, 0.01974177],
       [0.01974177, 0.11373685]])
```

```
np.cov(df['Fiscal Year'].fillna(df['Fiscal Year'].mean()), df['RedFlag'])
```

```
array([[ 8.37027261, -0.02253711],
       [-0.02253711,  0.11373685]])
```

Handle rare classes

- It is expected that in our DS, there are only a small fraction of redflag transactions. If we sample the whole DS normally, a major of the training data are non-redflag, and this makes the ML model favor prediction of non-redflag. Thus, in our training dataset, the number of redflag and non-redflag inputs should be more or less equal.
- How to do it? By oversamples. E.g. duplicate the redflag inputs in the DS to make the size non-redflag and redflag inputs more or less equal
- But we have to do it carefully. There are many good methods.
  - Navie Random over-sample
  - ROSE: Random Over-Sample Examples
  - SMOTE: Synthetic Minority Oversample Technique
  - ADASYN: Adaptive Synthetic Method

How to handle rar

- It is expected that in our DS, transactions. If we sample t data are non-redflag, and thi redflag. Thus, in our training redflag inputs should be mo
- How to do it? By oversampl to make the size non-redflag
- But we have to do it carefull
  - Navie Random over-sample
  - ROSE: Random Over-Sample E
  - SMOTE: Synthetic Minority O
  - ADASYN: Adaptive Synthetic

For example,

Package 'ROSE'

January 20, 2025

Type Package

Title Random Over-Sampling Examples

Version 0.0-4

Date 2021-06-14

Author Nicola Lunardon, Giovanna Menardi, Nicola Torelli

Maintainer Nicola Lunardon <nicola.lunardon@unimib.it>

Suggests MASS, nnet, rpart, tree

Description Functions to deal with binary classification problems in the presence of imbalanced classes. Synthetic balanced samples are generated according to ROSE (Menardi and Torelli, 2013). Functions that implement more traditional remedies to the class imbalance are also provided, as well as different metrics to evaluate a learner accuracy. These are estimated by holdout, bootstrap or cross-validation methods.

License GPL-2

NeedsCompilation no

Repository CRAN

Date/Publication 2021-06-14 08:10:09 UTC

We are now ready to apply the ML methods in SciKit-Learn to construct classifiers for detecting fraud transactions

We use the following ML methods from scikit-learn to construct classifiers

- GuassianNB: To learn and construct a Guassian Naive Bayes classifier
- LogisticRegression: To learn and construct Logistic Regression classifier
- DecisionTreeClassifier: To learn and construct a Decision Tree classifier
- RandomForestClassifier: To learn and construct a Random Forest classifier
- SGDClassifier: To learn and construct a Stochastic Gradient Descent (SGD) classifier

We use the following ML methods from scikit-learn to construct classifiers

- GuassianNB: To learn and construct a Guassian Naive Bayes classifier
- LogisticRegression: To learn and construct Logistic Regression classifier
- DecisionTreeClassifier: To learn and construct a Decision Tree classifier
- RandomForestClassifier: To learn and construct a Random Forest classifier
- SGDClassifier: To learn and construct a Stochastic Gradient Descent (SGD) classifier

To prepare a test data set for your boss

```
from sklearn.model_selection import train_test_split

data = data.dropna()
X_train, X_test, y_train, y_test = train_test_split(data[['Amount','Month','Status']],
                                   data['RedFlag'], test_size=0.2, random_state=0)
print(f"Training set has {X_train.shape[0]} samples")
print(f"Testing set has {X_test.shape[0]} samples")
```

```
Training set has 1904 samples
Testing set has 477 samples
```

Preparing an empty table for showing the result

```
summary = pd.DataFrame(columns=['Learner', 'Train Time', 'Pred Time', 'Acc score', 'F1 score',
                                'Precision', 'Recall'])
```

```python
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import accuracy_score, f1_score, precision_score, \
        recall_score, classification_report, confusion_matrix

NB = GaussianNB()
start = time()
NB.fit(X_train, y_train)
mid = time()
pred = NB.predict(X_test)
end = time()

pred_res = pd.DataFrame(pred)
pred_res = pred_res.set_index(y_test.index)
summary = pd.concat([summary,
                     pd.DataFrame({'Learner':['GaussianNB'],
                         'Train_Time':[mid-start],
                         'Pred_Time':[end-mid],
                         'Acc score':[accuracy_score(y_test,pred)],
                         'F1 score':[f1_score(y_test, pred, average='macro')],
                         'Precision':[precision_score(y_test, pred, average='macro')],
                         'Recall':[recall_score(y_test, pred, average='macro')]})],
                    ignore_index=True)


summary
```

	Learner	Train Time	Pred Time	Acc score	F1 score	Precision	Recall
0	GaussianNB	0.003991	0.001993	0.959227	0.915621	0.935524	0.89824

```python
from sklearn.linear_model import LogisticRegression

LR = LogisticRegression(random_state=0)
start = time()
LR.fit(X_train, y_train)
mid = time()
pred = LR.predict(X_test)
end = time()

summary = pd.concat([summary,
                     pd.DataFrame({'Learner':['LogisticRegression'],
                         'Train_Time':[mid-start],
                         'Pred_Time':[end-mid],
                         'Acc score':[accuracy_score(y_test,pred)],
                         'F1 score':[f1_score(y_test, pred, average='macro')],
                         'Precision':[precision_score(y_test, pred, average='macro')],
                         'Recall':[recall_score(y_test, pred, average='macro')]})],
                    ignore_index=True)
summary
```

	Learner	Train_Time	Pred_Time	Acc score	F1 score	Precision	Recall
0	GaussianNB	0.004751	0.003016	0.961373	0.921558	0.932438	0.911474
1	LogisticRegression	0.016485	0.002179	0.858369	0.503295	0.928726	0.521739

```python
from sklearn.tree import DecisionTreeClassifier

DT = DecisionTreeClassifier(max_features=0.2, max_depth=2,
                            min_samples_split=2, random_state=0)
start = time()
DT.fit(X_train, y_train)
mid = time()
pred = DT.predict(X_test)
end = time()
summary = pd.concat([summary,
                     pd.DataFrame({'Learner':['DecisionTree'],
                         'Train_Time':[mid-start],
                         'Pred_Time':[end-mid],
                         'Acc score':[accuracy_score(y_test,pred)],
                         'F1 score':[f1_score(y_test, pred, average='macro')],
                         'Precision':[precision_score(y_test, pred, average='macro')],
                         'Recall':[recall_score(y_test, pred, average='macro')]})],
                    ignore_index=True)
summary
```

	Learner	Train_Time	Pred_Time	Acc score	F1 score	Precision	Recall
0	GaussianNB	0.004751	0.003016	0.961373	0.921558	0.932438	0.911474
1	LogisticRegression	0.016485	0.002179	0.858369	0.503295	0.928726	0.521739
2	DecisionTree	0.007393	0.002051	0.869099	0.568224	0.933406	0.557971

```python
from sklearn.ensemble import RandomForestClassifier

RF = RandomForestClassifier(max_depth=2)
start = time()
RF.fit(X_train, y_train)
mid = time()
pred = RF.predict(X_test)
end = time()

summary = pd.concat([summary,
                     pd.DataFrame({'Learner':['Random Forest'],
                         'Train_Time':[mid-start],
                         'Pred_Time':[end-mid],
                         'Acc score':[accuracy_score(y_test,pred)],
                         'F1 score':[f1_score(y_test, pred, average='macro')],
                         'Precision':[precision_score(y_test, pred, average='macro')],
                         'Recall':[recall_score(y_test, pred, average='macro')]})],
                    ignore_index=True)
summary
```

	Learner	Train_Time	Pred_Time	Acc score	F1 score	Precision	Recall
0	GaussianNB	0.004751	0.003016	0.961373	0.921558	0.932438	0.911474
1	LogisticRegression	0.016485	0.002179	0.858369	0.503295	0.928726	0.521739
2	DecisionTree	0.007393	0.002051	0.869099	0.568224	0.933406	0.557971
3	Random Forest	0.309314	0.013900	0.976395	0.949861	0.986520	0.920290

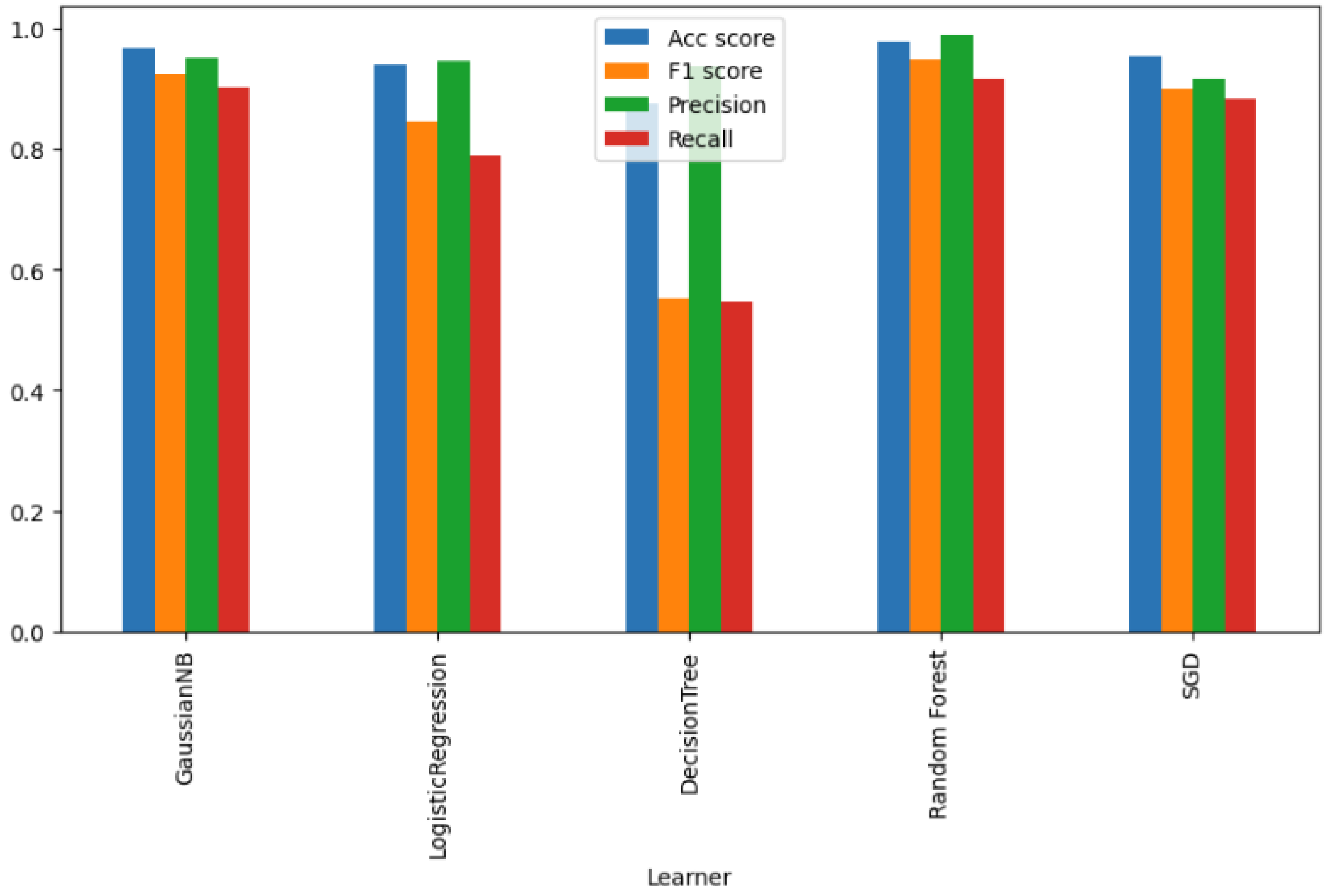
```
SGD = SGDClassifier(loss='hinge', penalty="l2")
start = time()
SGD.fit(X_train, y_train)
mid = time()
pred = SGD.predict(X_test)
end = time()

summary = pd.concat([summary,
                     pd.DataFrame({'Learner':['SGD'],
                         'Train_Time':[mid-start],
                         'Pred_Time':[end-mid],
                         'Acc score':[accuracy_score(y_test,pred)],
                         'F1 score':[f1_score(y_test, pred, average='macro')],
                         'Precision':[precision_score(y_test, pred, average='macro')],
                         'Recall':[recall_score(y_test, pred, average='macro')]})],
                    ignore_index=True)
summary
```

	Learner	Train_Time	Pred_Time	Acc score	F1 score	Precision	Recall
0	GaussianNB	0.004751	0.003016	0.961373	0.921558	0.932438	0.911474
1	LogisticRegression	0.016485	0.002179	0.858369	0.503295	0.928726	0.521739
2	DecisionTree	0.007393	0.002051	0.869099	0.568224	0.933406	0.557971
3	Random Forest	0.309314	0.013900	0.976395	0.949861	0.986520	0.920290
4	SGD	0.008706	0.001603	0.890558	0.676719	0.943080	0.630435

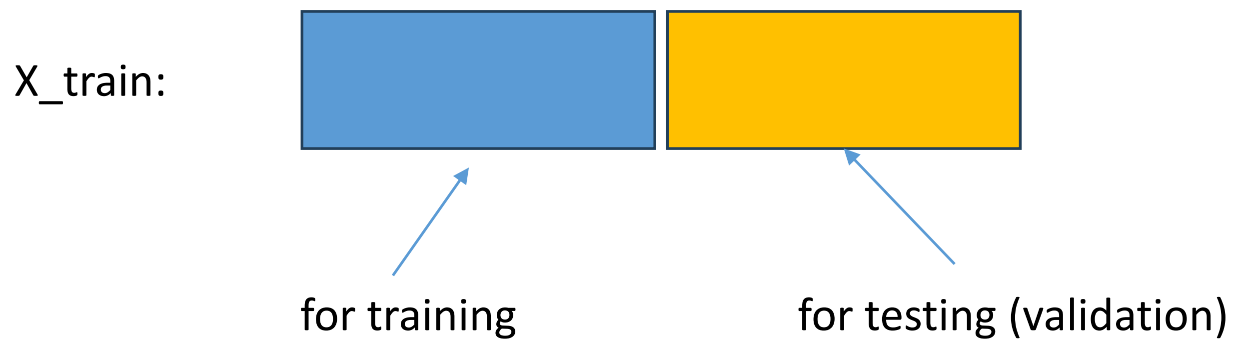
```
summary[['Learner', 'Acc score', 'F1 score', 'Precision', 'Recall']].plot(kind='bar', x = 'Learner', figsize=(10,5))
```

```
<Axes: xlabel='Learner'>
```



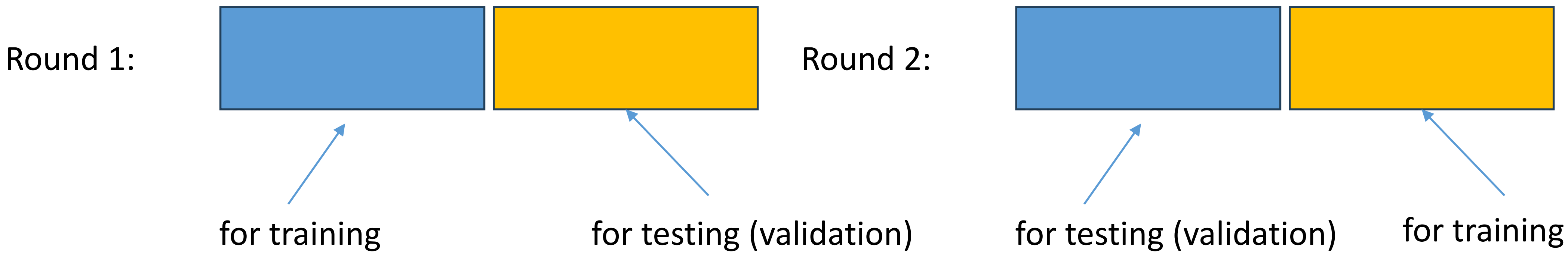
Validation

- After training a model using X_train, y_train, we test the model using your boss’ X_test, y_test
- But you want to be find out whether your model is not complex (e.g. too many layers in a NN, too many trees in a Random forest), and have good performance for unseen data set, you can divide further divide X_train into two:



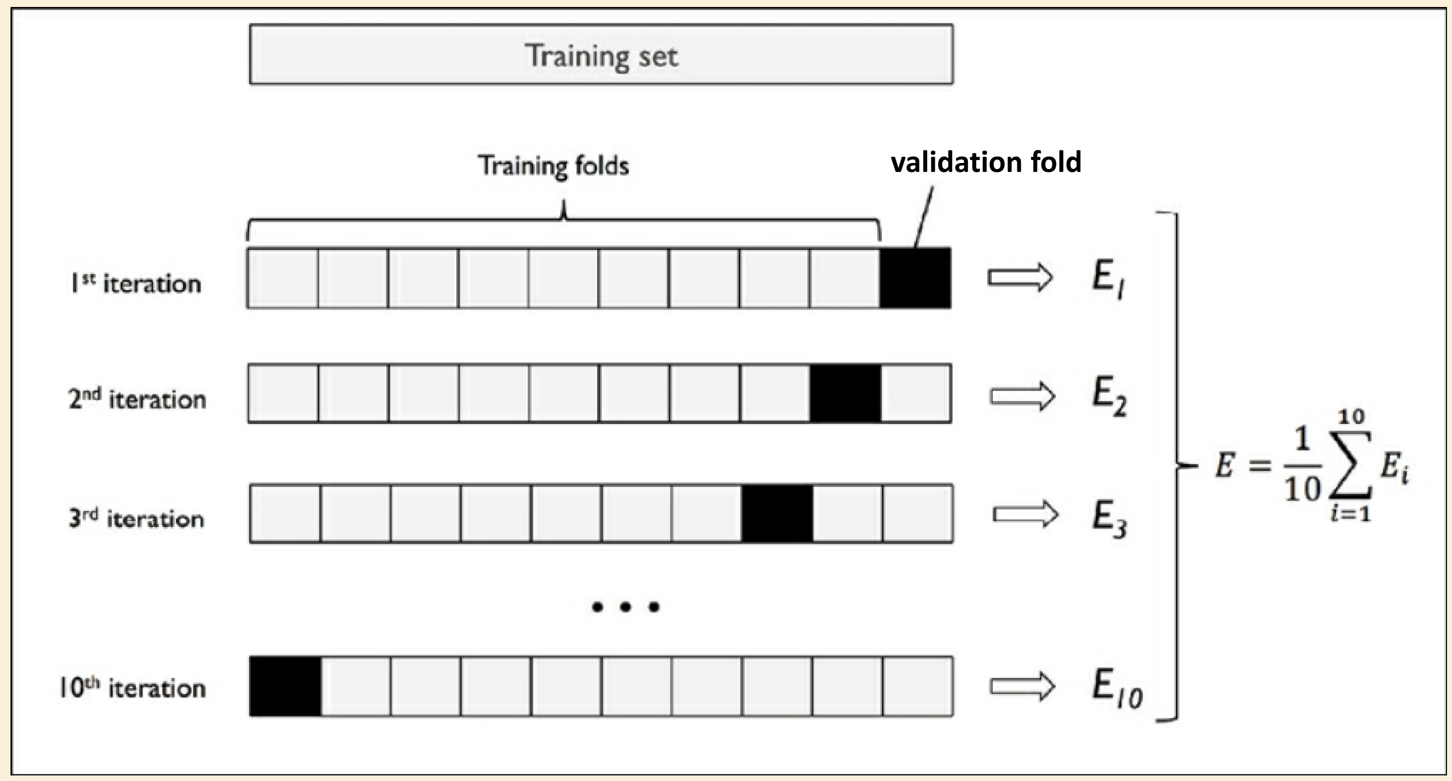
Cross validation

- After training a model using X_train, y_train, we test the model using your boss' X_test, y_test
- And to fully use X_train, we may switch the roles of the two data sets:



k-fold cross validation

- Example: 10-fold validation



k-fold cross validation

- Example: 10-fold validation

If the average performance of the 10 training sets is poor → model not powerful enough

If the average performance of the 10 training sets is good, but that of the 10 validation sets is poor → the model has been overfit

If the average performance of both the training sets and validation sets are good → the model is good