

# Traveling Diabetes Clinic problem

- python data science stack
  - pandas, scikit-learn, ...
- how to train ML models

from: [How Machine Learning Works](#), Manning,

*“Essentially, all models are wrong, but some are useful”*

George E. P. Box, British Mathematician

# Traveling Diabetes Clinic problem

## - A first take at the problem -

Diabetes is a serious chronic disease in which glucose, the main source of energy for the human body, accumulates in the bloodstream without being consumed. It becomes toxic rather than energetic.

Ruim 1,2 miljoen Nederlanders hebben diabetes en elke week komen daar 1.200 bij

It's estimated that around 30 million people in the United States alone have diabetes, and about 24% of those people are undiagnosed. In order to identify and address those undiagnosed patients, a group of doctors decided to initiate the Traveling Diabetes Clinic project.

The **goal behind this project** is for doctors to travel the country and stay in a certain location for a couple of days to find people who are undiagnosed, and then initialise their treatment plans and coach them on how to live an easy and pleasant life with this chronic disease.

The problem is that they have **very limited resources** (on both the human and equipment levels) to perform the necessary medical examinations on all who visit the clinic, and they want to make sure that these resources are efficiently utilised with those who probably have the disease. What **they'd like to have is way to quickly prioritise the visitors who will probably have diabetes**, so that they can utilize their resources more efficiently on confirming their condition and providing them with the necessary coaching and treatment.

There are two resources they have plenty of information on because they test almost every patient to visit the clinic: the **instant blood glucose (BG)** monitoring finger-sticks and the digital scale that calculates the **Body Mass Index (BMI)** of the patient. So these are the only two resources that can be used in any prioritising system.

## problem description

# Traveling Diabetes Clinic problem

- A first take at the problem -

The doctors prefer an **automated system**, so they can cut the time needed to go through all the records themselves and prioritise the patients by hand. This seems like a job for a software engineer, and that's why they come to you for help!

**You need to create a software system that takes two inputs (the BG value of the patient and their BMI), then produces one output: whether this patient probably has diabetes or not.**

A possible way to solve this problem is for the doctor to **design some rules** that decide whether a patient has diabetes given the values of their BG and BMI values. These rules can then be programmed and used to predict whether the patient is diabetic or not. **To create a reliable set of rules though, the doctors will need to go through mounds of data by hand in order to figure out the relation between BG and BMI to the existing of diabetes.** This is not possible because they, as stated before, don't have the time for such manual labor.

However, the fact that there is a mound of data that they can go through is interesting, maybe there's a way that these data can be utilised automatically.

**This is what *Machine Learning (ML)* is all about: using mounds of data to automatically figure out the underlying relations and structures!**

# Traveling Diabetes Clinic problem

- A first take at the problem: dataset sample -

We first need a **sample of people** (both diabetes and non-diabetes) with their BG and BMI recorded: **Pima Indians Diabetes Dataset<sup>(1)</sup>**.

This dataset<sup>(2)</sup> contains a sample of 768 adult women of the native American Pima Indian heritage, each member with records of various medical information as well as whether or not she has diabetes.

The dataset was originally owned by the National Institute of Diabetes and Digestive and Kidney Diseases, and was donated by The Johns Hopkins University's school of medicine (which conducted research on the same data) to the [UCI machine learning repository](https://archive.ics.uci.edu/), a source of datasets that is often used.

Dataset can be found in a csv<sup>(2)</sup> format in the dataset directory of module MCL.



(1) Think of **Dataset** as a dumped content of a database table.

(2) Format **csv** = 'comma-separated values'



# Traveling Diabetes Clinic problem

- A first take at the problem: reading the data with pandas -

To start working with this information, we need to load this file into our Python program. This can be achieved using the **pandas** library, which is one the most commonly used libraries in machine learning and data analysis, and is provided by default with the Anaconda distribution.

**pandas** is a very fast and powerful library providing set of **data structures** designed specifically to be used in the analysis of data stored in different formats, supporting a set of **advanced indexing and querying operations**.

One of the its operations that we're going to be using a lot is the **read\_csv** method, which allows us to read a **csv** format (or any variant of it like tsv) into our code.

We here use the **read\_csv** method with a single parameter that specifies the location of the file to read, but the function contains much more parameters that we can learn about in [pandas official docs](#)

```
import pandas as pd
data = pd.read_csv("./datasets/PimaIndiansDiabetes.csv")
```



# Traveling Diabetes Clinic problem

- A first take at the problem: reading the data -

To start working with this information, we need to load this file into our Python program. `read_csv` works by reading our csv data into a data structure called **DataFrame**.

```
[15]: import pandas as pd
```

```
data = pd.read_csv("../datasets/PimaIndiansDiabetes.csv")  
data.head(10) # show the first 10 rows  
# data.shape # show the dimesion (#rows, # columns)
```

```
[15]:
```

	Pregnancy Count	Blood Glucose	Diastolic BP	Triceps Skin Fold Thickness	Serum Insulin	BMI	Diabetes Pedigree Function	Age	Class
0	6	148	72	35	0	33.6	0.627	50	1
1	1	85	66	29	0	26.6	0.351	31	0
2	8	183	64	0	0	23.3	0.672	32	1
3	1	89	66	23	94	28.1	0.167	21	0
4	0	137	40	35	168	43.1	2.288	33	1
5	5	116	74	0	0	25.6	0.201	30	0
6	3	78	50	32	88	31.0	0.248	26	1
7	10	115	0	0	0	35.3	0.134	29	0
8	2	197	70	45	543	30.5	0.158	53	1
9	8	125	96	0	0	0.0	0.232	54	1

A **DataFrame** is basically a table, with **indexed rows** and **labeled columns**; so it follows intuitively that if we want to select some data within that table, we'll do that by providing info about in which rows and columns they are -> **.loc** method

see Jupiter notebook: Casus\_Traveling-Diabetes\_Clinic\_1



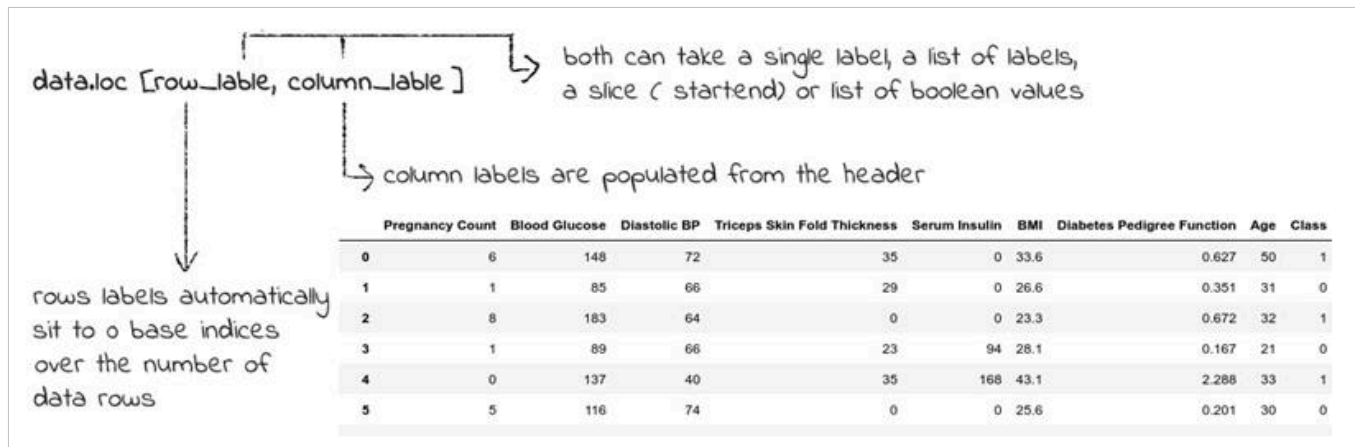
# Traveling Diabetes Clinic problem

- A first take at the problem: data of interest -

`data.loc[row_table, column_label]`: take the part of data we're interested in from amongst the other data.

Using the slice operator ':' as a `row_table`, selects all the rows in the DataFrame

`data_of_interest = data.loc[:, ["Blood Glucose", "BMI", "Class"]]`  
will get the data of interest



data

	Blood Glucose	BMI	Class
0	148	33.6	1
1	85	26.6	0
2	183	23.3	1
3	89	28.1	0
4	137	43.1	1
...	...	...	...
763	101	32.9	0
764	122	36.8	0
765	121	26.2	0
766	126	30.1	1
767	93	30.4	0

768 rows x 3 columns

data\_of\_interest

see Jupiter notebook: Casus\_Traveling-Diabetes\_Clinic\_1

# Traveling Diabetes Clinic problem

- A first take at the problem: need for a model -

- We're starting simple; we need a simple *model* to use and we also want a simple and efficient way to use it. But before we can choose any model, we first need to understand **what is a model**.
- Looking at a diabetic person, we implicitly know that there's **some process going on inside their bodies that links their measurements to their disease**.
  - For example, for someone who has diabetes, we know that there is something going on in their body that causes their BG values to be above normal; we might not know the exact details of that process, but we know that there's a process.
  - This is generally the way we look at the world: between the pieces of data we gather from any phenomenon, we expect that **there is a process that leads one piece to the other**.
  - However, these processes can be very complicated to the extent that we can't capture all their aspects and their details, and that's why we create a [model](https://nl.wikipedia.org/wiki/Model) for them.

<https://nl.wikipedia.org/wiki/Model>



# Traveling Diabetes Clinic problem

## - A first take at the problem: a model -

A **model** is an assumption we make about how the process under investigation works in order to approximate it.

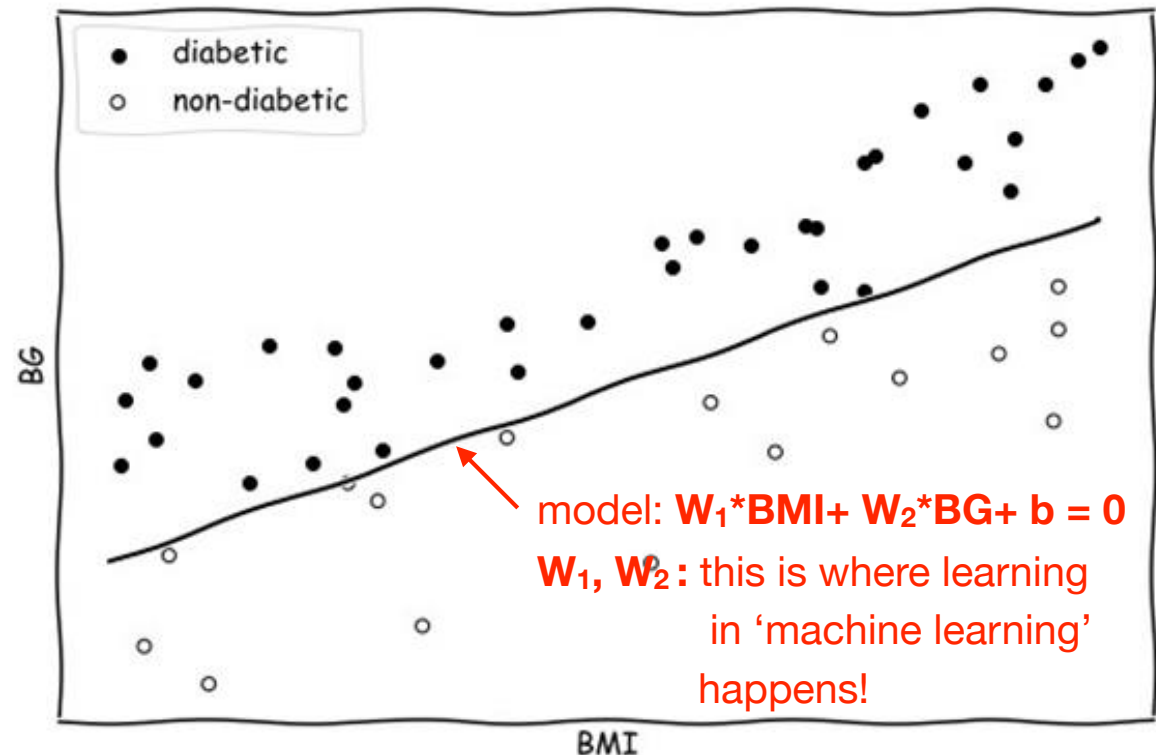
One of the most important characteristics of a model is that we can work with it that is, we have the tools (both mathematical and computational) that allow us to get some useful information out of it.

This characteristic usually means that the model will lose details from the actual phenomenon; but **if the model is good enough we can capture a lot of the important details.**

For example, one of the simplest models is the **linear model**.

In a linear model, we assume the internal process works through a line that is, we can differentiate between a diabetic and a non-diabetic person by determining on which side of a straight line this person appears.

$$y = \begin{cases} 1 & \text{if } w_1 x'_1 + w_2 x'_2 + b \geq 0 \\ 0 & \text{otherwise} \end{cases}$$



# Traveling Diabetes Clinic problem

- A first take at the problem: implementing a model -

- `scikit-learn` is *the* machine learning library in Python!
  - With a huge community of users, maintainers, and contributors, it provides a comprehensive, consistent and easy-to-use machine learning framework covering a wide range of models and algorithms across the machine learning world. Among the diversity of models that `scikit-learn` provides, our simple linear model we chose resides there under the name *Perceptron*.
  - Models in `scikit-learn` are organised by their category; because the Perceptron model is essentially a linear model, we'll find it under the `linear_model` category.

```
from sklearn.linear_model import Perceptron
from sklearn.model_selection import train_test_split

X = data_of_interest.loc[:, ["Blood Glucose", "BMI"]]
y = data_of_interest.loc[:, "Class"]

X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=42)

classifier = Perceptron()
classifier.fit(X_train, y_train)
```

you're teaching a machine learning model with the training data and to test their performance, you need to try the model on data it hasn't seen during training.



# Traveling Diabetes Clinic problem

- A first take at the problem: accuracy of a model: score -

Now that we've fitted the model to the training data, testing time comes. Will the model be able to predict diabetic people correctly and with enough accuracy so that we can use it for the prioritising task?

`scikit-learn`'s model provide us with the `score` method to do that task; it takes the test data and return the portion correctly predicted samples among the test set.

```
accuracy = classifier.score(X_test, y_test)
print("Prediction Accuracy: {:.2f}%".format(accuracy * 100))
```

```
Prediction Accuracy: 35.94%
```

Our linear model is only able to predict about 36% of the data patients in the test data correctly. **That's not very good.**

Even more, if we tried to get the model's score on the training data itself (which the model is supposed to ace completely), we'll find that it can only predict about 35% correctly!

```
train_accuracy = classifier.score(X_train, y_train)
print("Training Prediction Accuracy: {:.2f}%".format(train_accuracy * 100))
```

```
Training Prediction Accuracy: 34.72%
```



# Traveling Diabetes Clinic problem

- A first take at the problem: accuracy of a model: baseline -

- Previous score suggests that the linear model we chose is not suitable for the problem at all.
  - We can further prove that by seeing that it's performing much worse than a dummy *baseline*.
  - One type of model that we can use as a baseline is called a **dummy model**. Dummy models do not learn anything from the data, they just generate their decision by following a rule that may or may not be related to the data.
    - For example, a dummy model for our problem here is **one that outputs 0 or 1 at random with a 50% chance for each**; this is an example of a dummy rule that is not related to the data.
    - Another dummy model is one that **always outputs the most frequent class in the training data**; this dummy model is related to the data, but it does not learn from the data.
  - These kinds of dummy models are provided in the `sklearn.dummy` module. All of them are implemented in the `DummyClassifier` class. We'll use the `MostFrequentClassifier` for our baseline. This strategy is called **the most frequent class**. We're going to use the `MostFrequentClassifier` in the training data.
- You can learn more about dummy models in the [scikit-learn docs](#)

**DummyClassifier is a classifier that makes predictions using simple rules. This classifier is useful as a simple baseline to compare with other (real) classifiers. Do not use it for real solutions.**

```
from sklearn.dummy import DummyClassifier

dummy_baseline = DummyClassifier(strategy='most_frequent')
dummy_baseline.fit(X_train, y_train)

baseline_accuracy = dummy_baseline.score(X_test, y_test)
print("Dummy Prediction Accuracy: {:.2f}%".format(baseline_accuracy * 100))
```

Dummy Prediction Accuracy: 64.06%

**This shows  
stronger evidence  
against the perceptron  
model**

see Jupiter notebook: Casus\_Traveling-Diabetes\_Clinic\_1



# Traveling Diabetes Clinic problem

- A first take at the problem: conclusion first approach with Perceptron model -
- **Previous score suggests that the linear model we chose is not suitable for the problem at all.**
- It's performing even much worse than a dummy *baseline*.

We need to take a deeper look inside the data we used to start to understand its characteristics and see how it looks like - that is, ***describe it***.

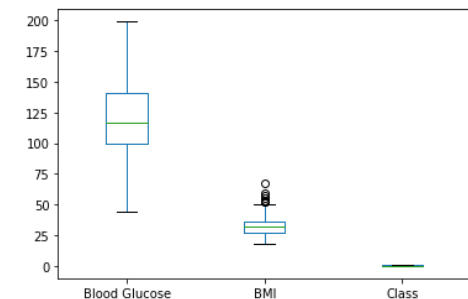
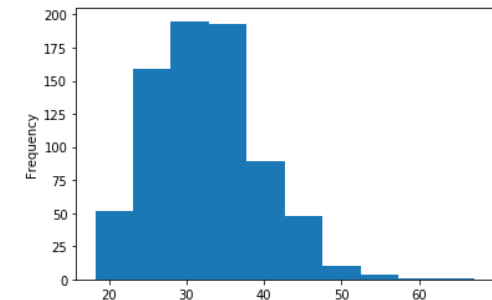
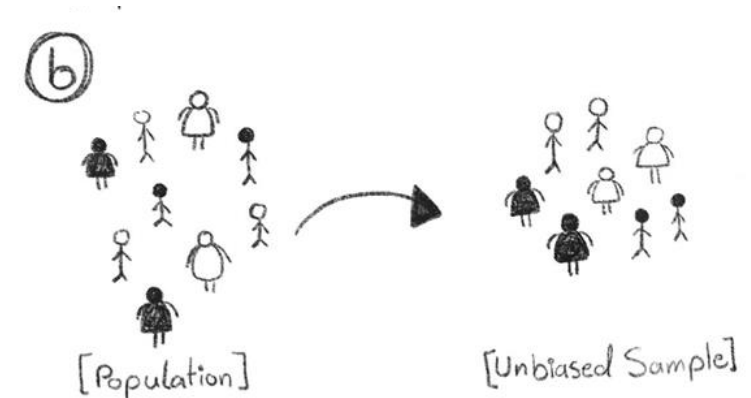
By describing the data, we'll be able to make more convenient models that fit the data better than blindly choosing a model.

First, we need to understand what the data represents and how it relates to the real-world phenomenon that we're trying to study.

# Traveling Diabetes Clinic problem

- grokking the problem: what does the data look like? -

- We'll be taking a deeper look inside the data we used to start to understand its characteristics and see how it looks like - that is - ***describe the data.***
- By describing the data, we'll be able **to make more convenient models** that fit the data better than blindly choosing a model.



*“Statistics may be defined as a body of methods for making wise decisions in the face of uncertainty”*

W.A. Wallis, American economist and statistician

# Traveling Diabetes Clinic problem

- grokking the problem: what does the data look like? -

## ◆ statistics

- statistic can be simply defined as a single piece of info or data digested from a large set of numerical values,
- the field of mathematical statistics is concerned with analysing, extracting, and finally using these pieces of information in a principled, rigorous fashion

## ◆ descriptive statistics

- concerned with describing various aspects of the sample in order to get insights into the data, its validity, characteristics, and shape
- it leads us finally to a principled process for choosing a model

## ◆ inferential statistics

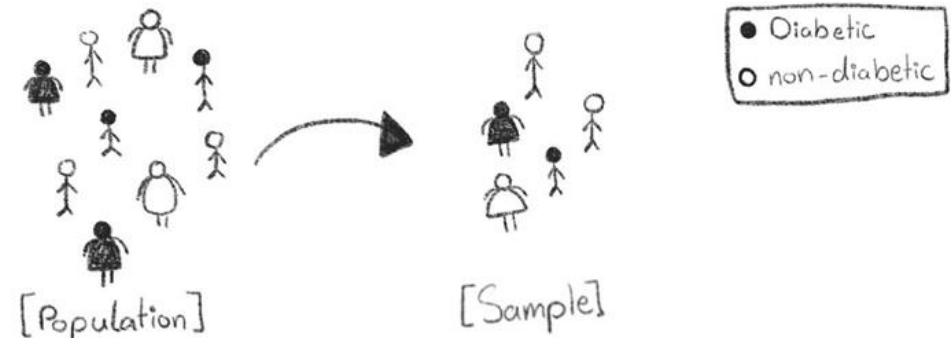
- the process of using these pieces of data and information to infer facts and rules that govern the original population

# Traveling Diabetes Clinic problem

- grokking the problem: descriptive statistics -

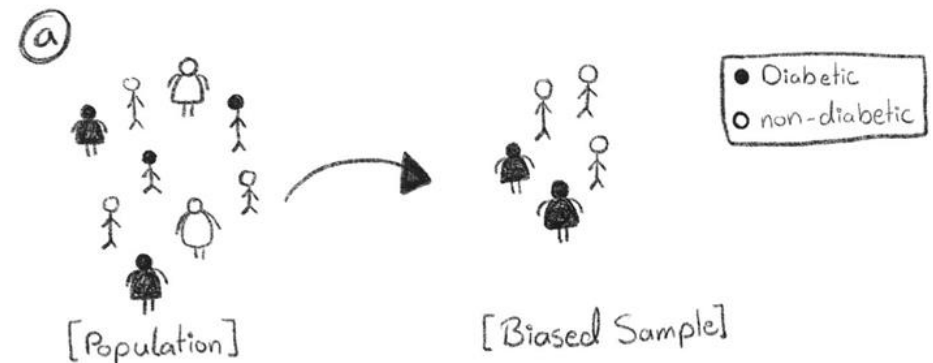
## ◆ populations and samples

- World Health Organization (WHO): the number of adults suffering from diabetes around the world has blown from 108 million in 1980 to 422 million adults in 2014
- our sample: 768 records Including non-diabetic



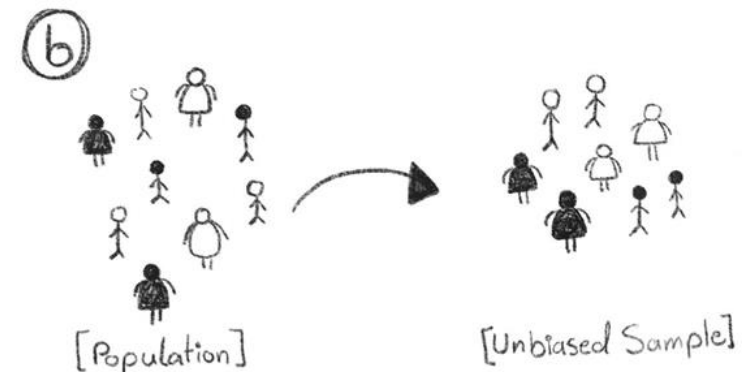
## ◆ biased vs. unbiased sample

- a) **biased sample**, it contains only overweight diabetics and normal-weight non-diabetics
  - any model we use will predict “diabetic” for anyone whose BMI value is high.
- b) **unbiased sample**: we don't see any favoring toward specific features in the data.



## ◆ descriptive statistics

- start building a better model for the traveling diabetes clinic, we need to investigate the PIMA Indians data, we have and see if it's an unbalanced sample or not and **what would be the best model to fit it. That's what descriptive statistics allows us to do.**





# Traveling Diabetes Clinic problem

- grokking the problem: what does the data look like? -

## ◆ descriptive statistics

- types of data

It is important to know what kind of data you're dealing with because different techniques might have different nuances, depending on what kind of data you're you're handling.

◆ numerical

◆ ordinal

◆ categorical

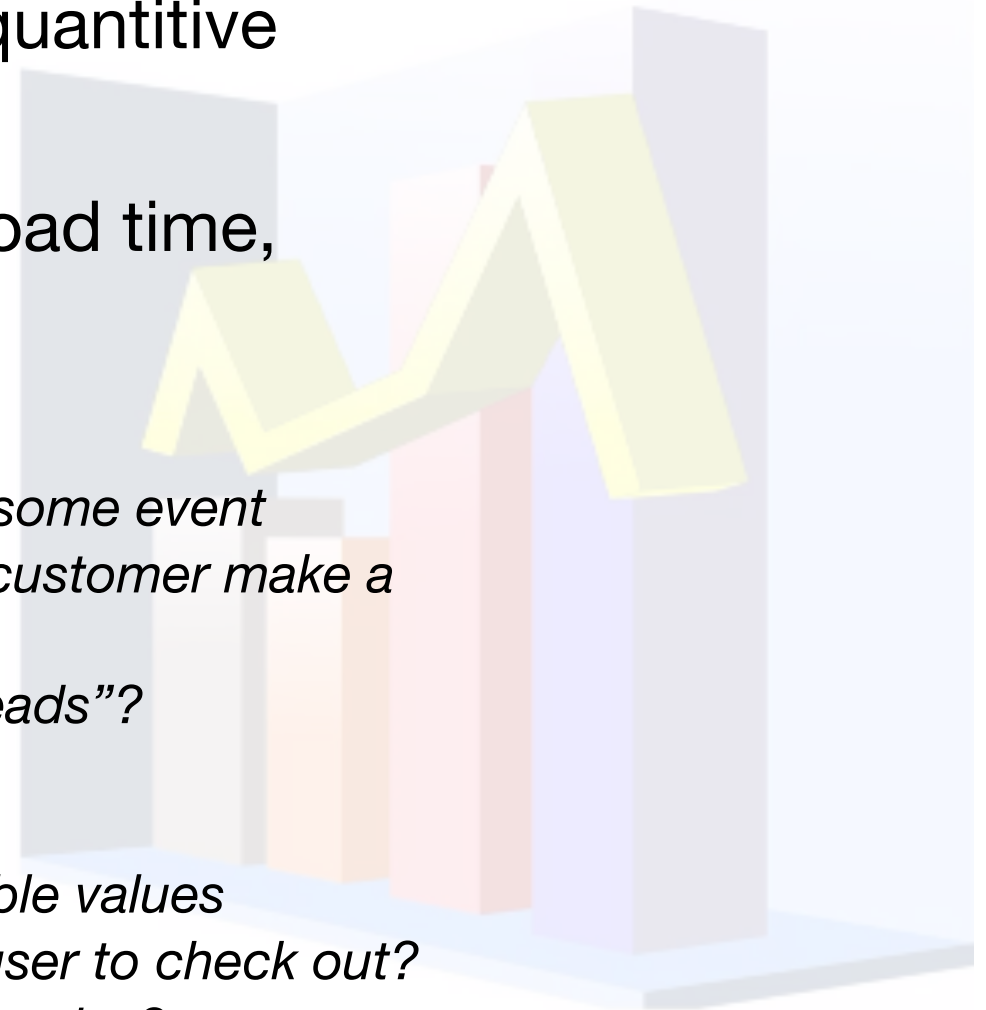
# Traveling Diabetes Clinic problem

- grokking the problem: what does the data look like? -

## numerical

most common

- represents some sort of quantitative measurement
- heights of people, page load time, stocks prices, etc.
- **discrete data**
  - *integer based, often counts of some event*
    - *how many purchases did a customer make a year?*
    - *How many time did I flip “heads”?*
- **continuous data**
  - *has an infinite number of possible values*
    - *How much did it take for a user to check out?*
    - *How much rain fell on a given day?*
    - *What was the temperature during the day?*



# Traveling Diabetes Clinic problem

- grokking the problem: what does the data look like? -

## categorical

- qualitative data that has no inherent mathematical meaning
- gender, Yes/No (binary data), race, state of residence, product category, political party, etc.
- you can assign numbers to categories in order to represent them more compactly, but the numbers don't have mathematical meaning

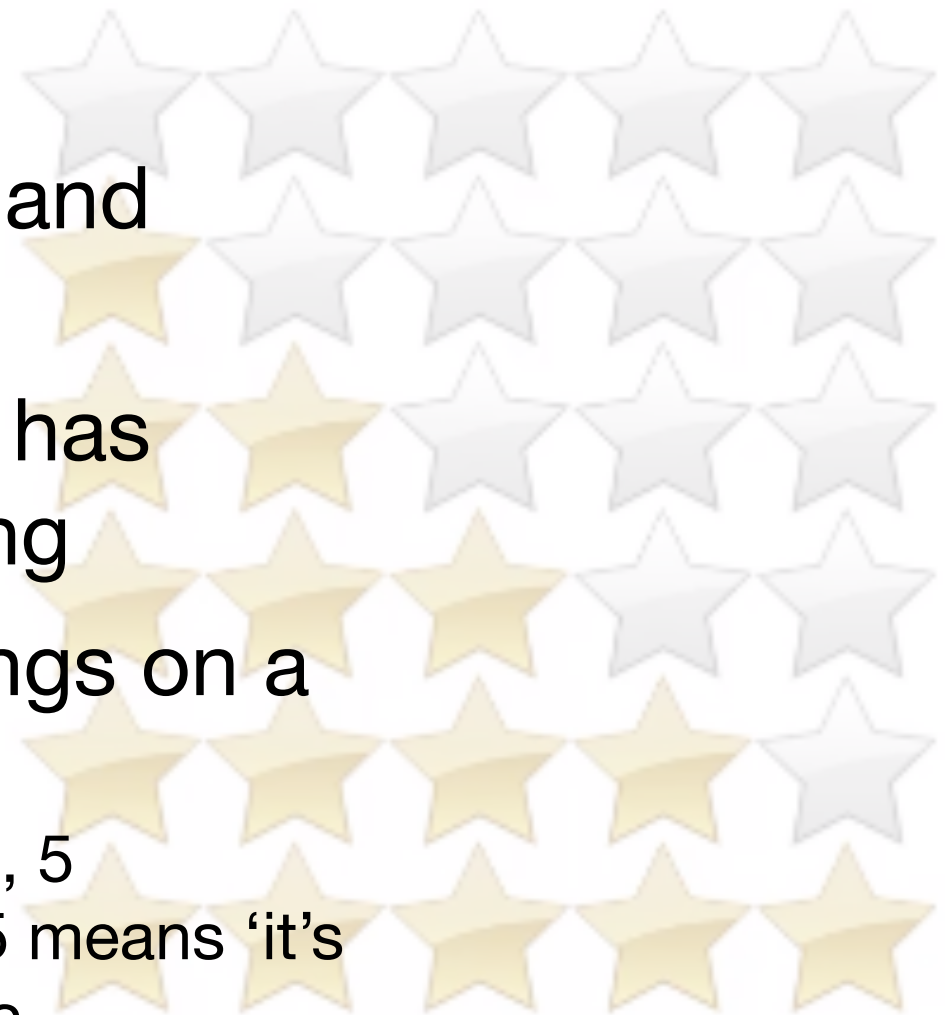


# Traveling Diabetes Clinic problem

- grokking the problem: what does the data look like? -

## ordinal

- mixture of numerical and categorical data
- categorical data that has mathematical meaning
- **Example:** movie ratings on a 1-5 scale
  - ratings must be 1, 2, 3, 4, 5
  - mathematical meaning: 5 means 'it's a better movie than 1, etc



# Traveling Diabetes Clinic problem

- grokking the problem: what does the data look like? -

## Quick Quiz

Are the following types of data **numerical**, **categorical**, or **ordinal**?

1. How much gas is in your car gas tank?
2. A rating of your overall health, choices are 1 ('poor'), 2 ('moderate'), 3('good'), or 4('excellent')?
3. Races of your classmates
4. Ages in years?
5. Money spent in store?



# Traveling Diabetes Clinic problem

- grokking the problem: what does the data look like? -

## ◆ descriptive statistics

- mean, mode, median

*measures of central tendency*, which are a bunch of descriptive statistics outlining the central values that our sample values tend to be around.

mean:  $\bar{x} = \frac{1}{m} \sum_{i=1}^m x^{(i)}$

```
mean_values = data_of_interest.mean()
print(mean_values)
```

```
Blood Glucose    120.894531
BMI               31.992578
Class            0.348958
dtype: float64
```

*The number in the superscript of  $x$  is not a power, but the index of the example in the sample.*

mode: which is the most frequent value across the sample.

```
mode_values = data_of_interest.mode()
print(mode_values)
```

	Blood Glucose	BMI	Class
0	99	32.0	0.0
1	100	NaN	NaN

median: which is simply the value that splits the sample into two halves: a lower one and a higher one.

```
median_values = data_of_interest.median()
print(median_values)
```

```
Blood Glucose    117.0
BMI              32.0
Class            0.0
dtype: float64
```

see Jupiter notebook: [Casus\\_Traveling-Diabetes\\_Clinic\\_2](#)



# Traveling Diabetes Clinic problem

- grokking the problem: what does the data look like? -

## ◆ descriptive statistics

- **range:** minimum and maximum values
  - sense of value variations
  - missing values -> solution

```
bmi_zeros_mask = data_of_interest.loc[:, "BMI"] != 0
bg_zeros_mask = data_of_interest.loc[:, "Blood Glucose"] != 0


clean_data_of_interest = data_of_interest[bmi_zeros_mask & bg_zeros_mask]
```

**Pandas** provide us with a very neat way of doing such filtering and querying into its data frames.

```
min_values = data_of_interest.min()
max_values = data_of_interest.max()

print(min_values)
print("=====")
print(max_values)
```

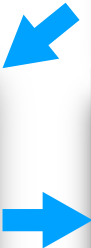
```
Blood Glucose    0.0
BMI              0.0
Class            0.0
dtype: float64
=====
Blood Glucose    199.0
BMI              67.1
Class            1.0
dtype: float64
```



```
min_values = clean_data_of_interest.min()
max_values = clean_data_of_interest.max()

print(min_values)
print("=====")
print(max_values)
```

```
Blood Glucose    44.0
BMI              18.2
Class            0.0
dtype: float64
=====
Blood Glucose    199.0
BMI              67.1
Class            1.0
dtype: float64
```



see Jupiter notebook: [Casus\\_Traveling-Diabetes\\_Clinic\\_2](#)

# Traveling Diabetes Clinic problem

- grokking the problem: what does the data look like? -

## ◆ descriptive statistics

### • quantiles

- gives us a more detailed picture about the distribution of the data

### • boxplots

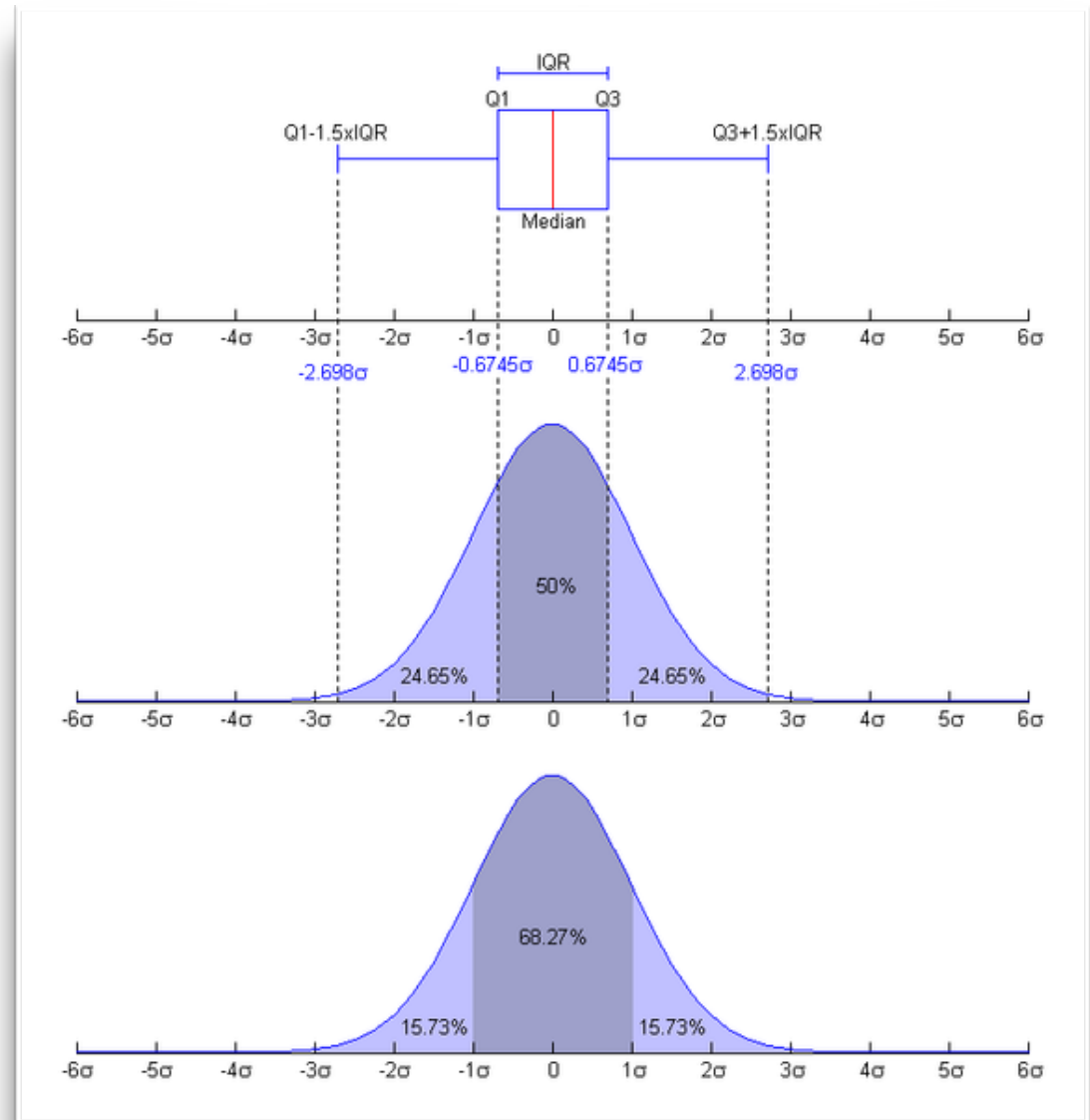
- outliers

**Q1** = meetwaarde op  $\frac{1}{4}$  (25%)

**Q2** = mediaan = de middelste waarde (50%)

**Q3** = meetwaarde op  $\frac{3}{4}$  (75%)

1. Calculate Q1 ( the first Quarter)
2. Calculate Q3 ( the third Quartile)
3. Find IQR (InterQuartile Range) =  $(Q3 - Q1)$
4. Find the lower Range =  $Q1 - (1.5 * IQR)$
5. Find the upper Range =  $Q3 + (1.5 * IQR)$





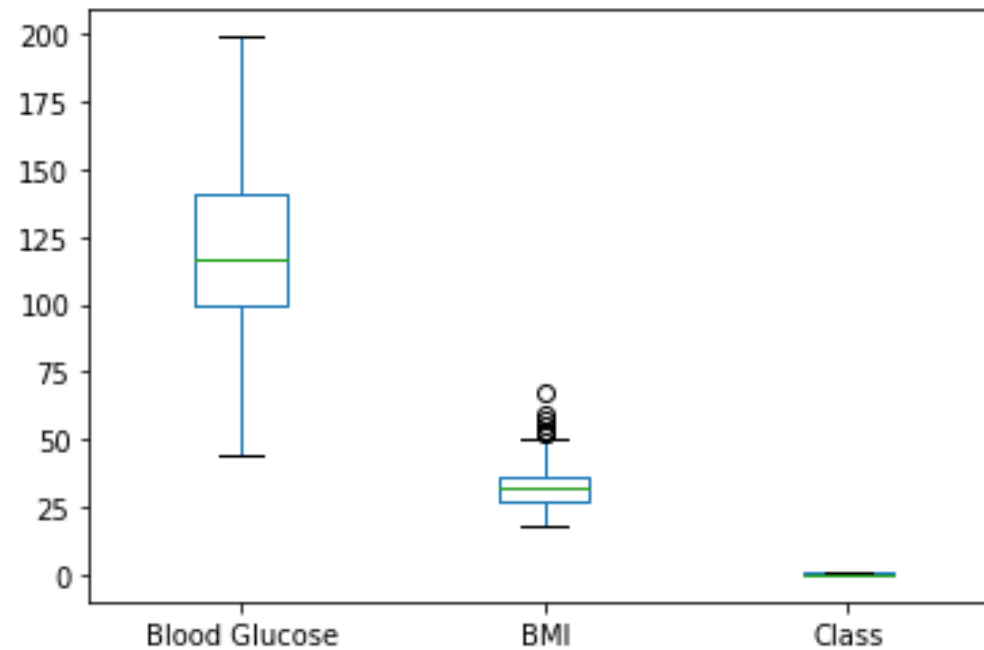
# Traveling Diabetes Clinic problem

- grokking the problem: what does the data look like? -

```
quartiles = clean_data_of_interest.quantile([0.25, 0.5, 0.75])  
  
print(quartiles)  
clean_data_of_interest.plot.box()
```

	Blood Glucose	BMI	Class
0.25	99.75	27.5	0.0
0.50	117.00	32.3	0.0
0.75	141.00	36.6	1.0

<matplotlib.axes.\_subplots.AxesSubplot at 0x11d0847d0>



# Traveling Diabetes Clinic problem

- grokking the problem: what does the data look like? -

## ◆ descriptive statistics

- how is the spreading of data around central tendency?
- **variance** and **standard deviation**
  - numeric measurements about the spreading of data around central tendency
- **histogram plot**
  - visual distribution of data

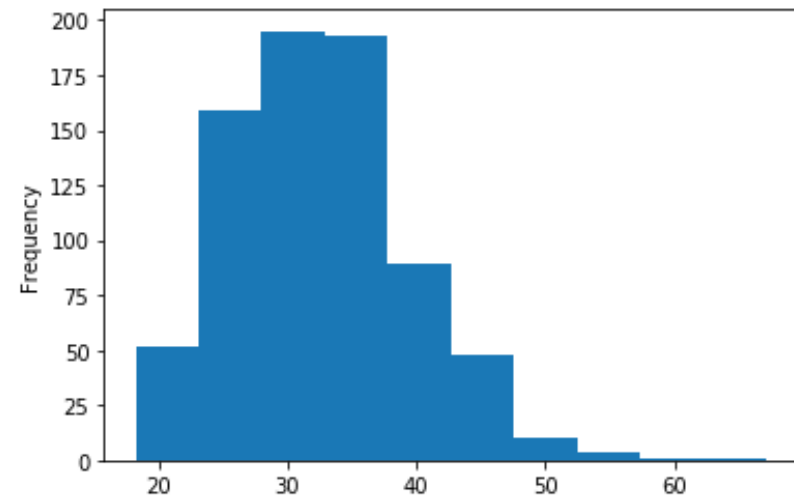
```
sample_vars = clean_data_of_interest.var()
sample_stds = clean_data_of_interest.std()
```

```
print(sample_vars)
print("=====")
print(sample_stds)
```

```
Blood Glucose    936.433323
BMI              48.010018
Class            0.228121
dtype: float64
=====
Blood Glucose    30.601198
BMI              6.928926
Class            0.477621
dtype: float64
```

```
clean_data_of_interest.loc[:, "BMI"].plot.hist()
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x11d189750>
```



see Jupiter notebook: Casus\_Traveling-Diabetes\_Clinic\_2

# Traveling Diabetes Clinic problem

- grokking the problem: what does the data look like? -

## ◆ descriptive statistics

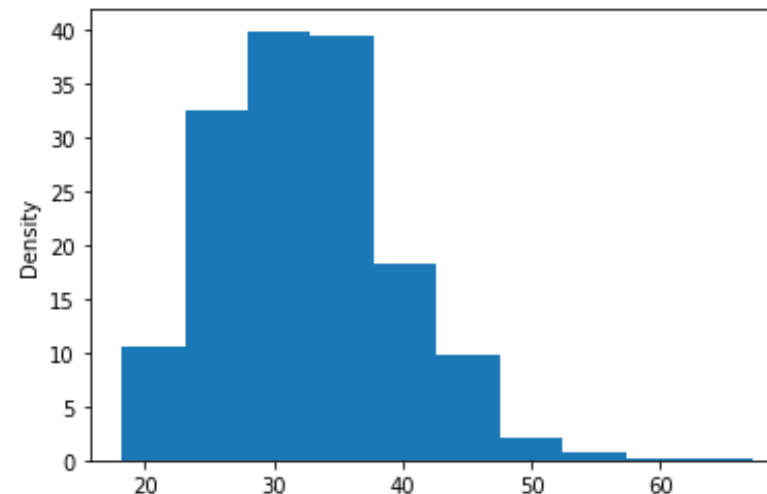
- **histogram plot**
  - sample distribution
- **density plots**
  - gives us a way to leap out of the sample and start to approximating the distribution of the original population

```
bin_size = 4.89 # the bin size in the 10 bins histogram

data_size = len(clean_data_of_interest.loc[:, "BMI"])
weights_seq = [1 / bin_size] * data_size

clean_data_of_interest.loc[:, "BMI"].plot.hist(weights=weights_seq) \
.set_ylabel("Density") # changes the label on y-axis
```

Text(0, 0.5, 'Density')



# Traveling Diabetes Clinic problem

- grokking the problem: what does the data look like? -

## 1. make histogram plots for BG

- Generate the histogram plots (both frequency and density) for the Blood Glucose values.

## 2. data sample biased or unbiased?

- Is the data sample we're using biased, or unbiased? Think about how you can determine that using the tools you've learned.



# Traveling Diabetes Clinic problem

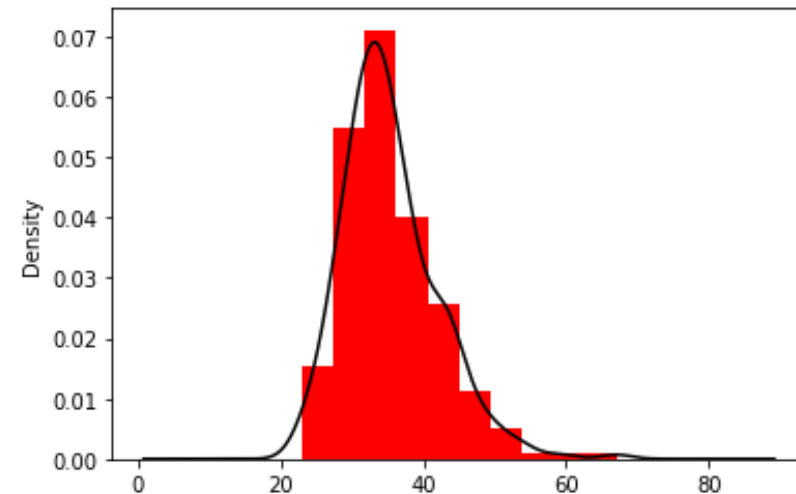
- grokking the problem: where did the data come from? -

We ended with a look at how the data within the sample is distributed ... approximate the original distribution of the data.

Using probability theory we'll have a much better model: **Naive Bayes Model** with scikit-learn probability and distributions

```
: diabetics.loc[:, "BMI"].plot.hist(color='red', normed=True)  
diabetics.loc[:, "BMI"].plot.density(color='black')
```

```
: <matplotlib.axes._subplots.AxesSubplot at 0x115711d10>
```



```
1 from sklearn.naive_bayes import GaussianNB  
2 from sklearn.model_selection import train_test_split  
3  
4 X = clean_data_of_interest.loc[:, ["Blood Glucose", "BMI"]]  
5 y = clean_data_of_interest.loc[:, "Class"]  
6 X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=42)  
7  
8 classifier = GaussianNB()  
9 classifier.fit(X_train, y_train)  
10  
11 accuracy = classifier.score(X_test, y_test)  
12 print("Prediction Accuracy: {:.2f}%".format(accuracy * 100))
```

Prediction Accuracy: 79.79%

```
accuracy =  
classifier.score(X_test,  
y_test)  
print("Prediction Accuracy:  
{:.2f}%".format(accuracy *  
100))
```

Prediction Accuracy: 35.94%

see Jupiter notebook: Casus\_Traveling-Diabetes\_Clinic

## ARTIFICIAL INTELLIGENCE

Early artificial intelligence stirs excitement.



## MACHINE LEARNING

Machine learning begins to flourish.



## DEEP LEARNING

Deep learning breakthroughs drive AI boom.



1950's 1960's 1970's 1980's 1990's 2000's 2010's

# Vragen? opmerkingen?