AI Workflow

Data Gathering

Gather → Explore → Clean → Modify → Prepare → Train → Evaluate

Get Data | Clean, Prepare & Modify Data | Train Model | Measure

WEB SCRAPING

WEB CRAWLING

A "Crawler" discovers the different websites and downloads them as HTML. It will then follow other pages (<a href="#">) and download these. Note that a robots.txt file located at the root explains to the Crawler which pages it can download.

Example *(javascript)*

```
import puppeteer from 'puppeteer';
const browser = await puppeteer.launch();
const page = await browser.newPage();
await page.goto('https://xaviergeerinck.com');
const html = await page.evaluate(() => document.documentElement.outerHTML);
console.log(html);
```

DATA PARSING & EXTRACTION

Once data is downloaded, it can be parsed and extracted. For this it's common to utilize a technique called "Regular Expressions" or XPaths.

Example *(javascript)*

```
// XPath
const xPath = '//html[1]/body[1]/div[1]';
document.evaluate(xPath, document, null, XPathResult.FIRST_ORDERED_NODE_TYPE, null).singleNodeValue;

// Regular Expression (RegEx)
const str = "Hello World!"
const re = new RegExp(/[A-Za-z]*/, 'g');
const matches = str.match(re);
--> ["Hello", "", "World", "", ""]
```

PUBLIC DATASETS

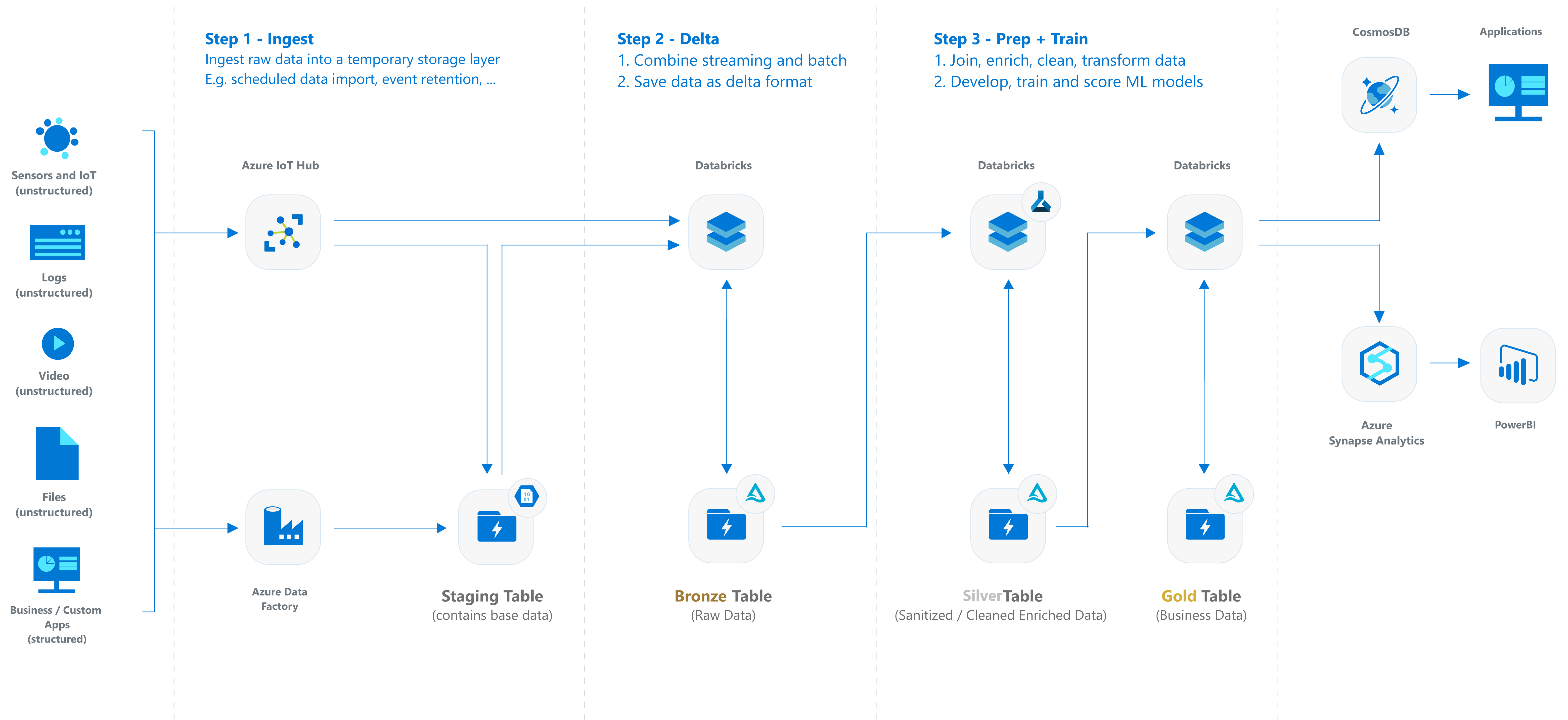
PROVIDER	URL
Kaggle	https://www.kaggle.com/datasets
Microsoft	https://azure.microsoft.com/en-us/services/open-datasets/catalog/
Google	https://console.cloud.google.com/marketplace/browse?filter=solution-type:dataset
Amazon	https://registry.opendata.aws/

COMPANY DATA PLATFORM

DESCRIPTION

A data platform is a centralized platform within an organization capturing a variety of different data sources and making them available to the different business units. This data platform takes care of tasks such ingestion, preparation and serving of the data to these different business units. With an ultimate goal of allowing business units to focus on their needs, without having to worry about cleaning and preparing their data in the correct format.

ARCHITECTURE EXAMPLE



AI Workflow

Data Exploration, Cleaning and Modification

Gather → **Explore** → **Clean** → **Modify** → Prepare → Train → Evaluate

Get Data | **Clean, Prepare & Modify Data** | Train Model | Measure

Note: Pandas is being utilized here due to its excellent compatibility with Spark, making it suited for Big Data Processing on a scalable system (assisted through the use of koalas) (=data engineering)
Once data is cleansed, SciKit can be utilized for model creation (= data science)

EXPLORE

EDA: Exploratory Data Analysis

TYPE

https://docs.python.org/3/library/functions.html#type

With one argument, return the type of an object

Command: type(var_name)

KEYS

https://docs.python.org/3/library/stdtypes.html?highlight=keys#dict.keys

Return a new view of the dictionary's keys

Command: var_name.keys()

NP & DF SHAPE

https://docs.scipy.org/doc/numpy/reference/generated/numpy.ndarray.shape.html

Tuple of dimensions for DataFrame or Numpy Array

Command: var_name.shape

DF HEAD

https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.head.html

Return the first n rows.

Command: df = df.head()

Example

```
> df = pd.DataFrame({'animal':['alligator', 'bee',
'falcon', 'lion', 'monkey', 'parrot', 'shark', 'whale', 'zebra']})
> df

      animal
0  alligator
1        bee
2     falcon
3       lion
4     monkey
```

PD MATRIX

https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.plotting.scatter_matrix.html

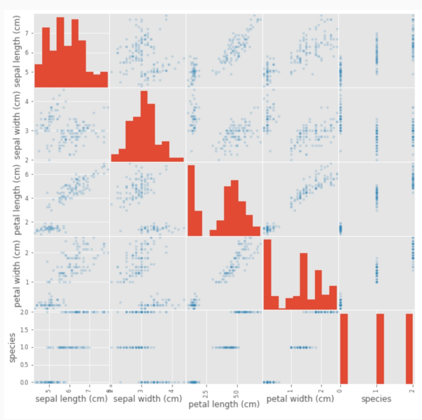
Draw a matrix of scatter plots.

Command: pd.scatter_matrix()

Example

```
# Load some data
iris = datasets.load_iris()
iris_df = pd.DataFrame(iris['data'],
columns=iris['feature_names'])
iris_df['species'] = iris['target']

pd.scatter_matrix(iris_df,
alpha=0.2, figsize=(10, 10))
plt.show()
```



PD DESCRIBE

https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.describe.html

Generate descriptive statistics that summarize the central

Command: df = df.describe()

Example

```
> s = pd.Series([1, 2, 3])
> s.describe()
count    3.0
mean     2.0
std      1.0
min      1.0
25%      1.5
50%      2.0
75%      2.5
max      3.0
dtype: float64
```

```
> s = pd.Series(['a', 'a', 'b', 'c'])
> s.describe()
count     4
unique    3
top       a
freq      2
dtype: object
```

CLEAN

Note: import pandas as pd & import numpy as np

DROP NULLS

https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.dropna.html

Return object with labels on given axis omitted where alternately any or all of the data are missing

Command: df = df.dropna()

Example

```
> df = pd.DataFrame([[np.nan, 1], [2, 3]])
> df.dropna()
0 [2, 3]
```

IMPUTE

https://pandas.pydata.org/pandas-docs/stable/user_guide/missing_data.html

Instead of discarding data, it's better to "Impute" it, i.e., to infer it from known part of the data

Example

```
> df = pd.DataFrame(np.random.randn(5, 3),
index=['a', 'b', 'c', 'd', 'e'], columns=['one', 'two', 'three'])

> df['one']['a'] = None # Set a value as undefined
> df = df.fillna(0) # Fill missing values with 0
> df = df.fillna(df.median()) # Fill missing values with median
```

REPLACE

https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.replace.html

Replace values by another one (e.g. NaN values by 0)

Command: s.replace(search_value, new_value)

Example

```
> df = pd.DataFrame([0])
> s.replace(0, 10) # Replace 0 by 10
   0
0 10
```

STANDARIZE (σ)

https://en.wikipedia.org/wiki/Standard_score

Standardization or z-score normalization takes into account the standard deviation
Formula: z = (x - μ) / σ where μ = mean σ = standard deviation

Example

```
> df = pd.DataFrame(np.random.randn(5, 3),
index=['a', 'b', 'c', 'd', 'e'], columns=['one', 'two', 'three'])

> df = (df - df.min()) / (df.max() - df.min())
```

NORMALIZE

https://en.wikipedia.org/wiki/Feature_scaling

Rescale the data to have values between 0 and 1
Formula: z = (x - min(x)) / (max(x) - min(x))

Example

```
> df = pd.DataFrame(np.random.randn(5, 3),
index=['a', 'b', 'c', 'd', 'e'], columns=['one', 'two', 'three'])

> df = (df - df.min()) / (df.max() - df.min())
```

MODIFICATION

BINNING

https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.cut.html

Return object with labels on given axis omitted where alternately any or all of the data are missing

Example

```
> rand_list = [np.random.randint(0, 100) for i in range(50)]
> pd.cut(rand_list, 5) # Create 5 equal sized bins
```

ONE-HOT ENCODING

https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.get_dummies.html

Convert categorical features to a numerical array of 0 or 1.
E.g. ['warm'] for [hot, warm, cold] becomes [0, 1, 0]

Example

```
> data = pd.DataFrame([['Male', 1], ['Female', 3], ['Female', 2]])

# One Hot encode and prefix new columns with ohe_
> pd.get_dummies(df, prefix='ohe')
```

LABEL ENCODING

https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.astype.html

Converts categorical feature to a numerical array of numbers
E.g. ['warm'] for [hot, warm, cold] becomes [1]

Example

```
> df = pd.DataFrame([['Male', 1], ['Female', 3], ['Female', 2]])
> df[0] = df[0].astype('category') # Convert to category
> df['0_encoded'] = df[0].cat.codes # Label encoding: .cat.codes
```

DATE EXTRACTION

https://docs.python.org/3/library/datetime.html

Extract parts of the date

Example

```
> from datetime import date
> df = pd.DataFrame({ 'date': ['01-01-2000', '31-12-2019'] })
> df['date'] = pd.to_datetime(df.date, format="%d-%m-%Y")
> df['date'].dt.year # Print year
> df['date'].dt.day_name() # Print weekday
```

PIVOT TABLES

https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.pivot_table.html

A table of statistics that summarize the data of a more extensive table. E.g. list of countries and cities to count of cities in that country

Example

```
> df = pd.DataFrame({ "city": ["brussels", "antwerp", "gent",
"seattle"], "country": ["BE", "BE", "BE", "USA"] })
> pd.pivot_table(df, values="city", index=['country'],
    aggfunc=np.count_nonzero)
```

SPLITTING

https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.Series.str.split.html

Split the characters in a column. E.g. name to first_name and last_name is quite common.

Example

```
> df = pd.DataFrame({ "name": ["Xavier Geerinck",
     "Jane Middle Doe"] })
> df['first_name'] = df.name.str.split(" ").map(lambda x: x[0])
> df['last_name'] = df.name.str.split(" ")
     .map(lambda x: "   ".join(x[1:]))
```

AI Workflow

Model Preparation & Training

Gather → Explore → Clean → Modify → **Prepare** → **Train** → Evaluate

Get Data	Clean, Prepare & Modify Data	**Train Model**	Measure

PREPARE

DATA PARTITIONING - TRAIN / VALIDATION / HOLDOUT SETS

Before we can create a Machine Learning Model, we need to create our train / validation and holdout sets by partitioning our data. Splitting this data can be done through sklearn or pandas. In the example on the right, you can see how to do this with Pandas.

DATA		
Train (~70% small, ~95% large)	**Validate** (~15% small, ~2.5% large)	**Test** *(= holdout)* (~15% small, ~2.5% large)

Example

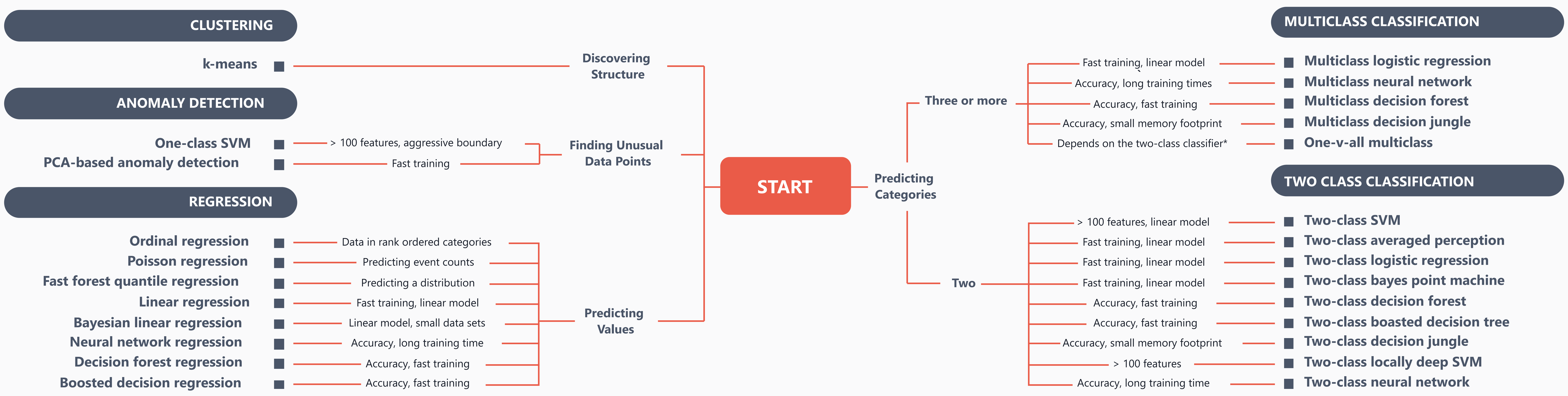
```
> df = pd.DataFrame([['Male', 1], ['Female', 3], ['Female', 2]])
> probs = np.random.rand(len(df))
> msk_training = probs < 0.70
> msk_test = (probs >= 0.7) & (probs < 0.85)
> msk_validation = probs >= 0.85

df_training = df[msk_training]
df_test = df[msk_test]
df_validation = df[msk_validation]
```

TRAINING

ML ALGORITHMS

Below an overview is given for the different Algorithms that are commonly used in Machine Learning. This is far from a complete list, but gives you an idea how how to tackle finding the correct algorithm for the type of task.



AVOIDING OVERFITTING

Overfitting is the concept of when we trained a model which performs excellent on our training / test set, but performs poorly on sets it hasn't seen before. This for example can happen when tuning our algorithm's hyperparameters until it performs he best, since we are validating it against our test set all the time. Thus tuning the Hyperparameters with leaking knowledge from the test set. One often used way to prevent overfitting is **"Cross Validation"**.

K-FOLD CROSS VALIDATION

In k-fold cross validation, we split the training set in k sets. We then execute the following:

1. Train a model using k - 1 folds
2. Validate the resulting model on the remaining part of the data

■ = **Test Data** □ = **Training Data**

Iteration 1	**Fold 1**	Fold 2	Fold 3	Fold 4	···	Fold k
Iteration 2	Fold 1	**Fold 2**	Fold 3	Fold 4	···	Fold k
Iteration 3	Fold 1	Fold 2	**Fold 3**	Fold 4	···	Fold k
Iteration 4	Fold 1	Fold 2	Fold 3	**Fold 4**	···	Fold k
	⋮	⋮	⋮	⋮		
Iteration k	Fold 1	Fold 2	Fold 3	Fold 4	···	**Fold k**

AUTOML & GRIDSEARCH

AutoML is the vision of executing the full End-to-End pipeline of data preparation, algorithm selection and algorithm tuning automatically. For algorithm tuning, a technique called "**Grid Search**" is often used to automatically tune the different model hyperparameters.

Example *https://scikit-learn.org/stable/modules/grid_search.html*

```
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import GridSearchCV

# Hyperparameter Grid
param_grid = {'C': np.logspace(-5, 8, 15)}

# Initialize Logistic Regression Classifier
model_logreg = LogisticRegression()

# Initialize GridSearch with Cross Validation = 5
model_logreg_cv = GridSearchCV(model_logreg, param_grid, cv=5)

# Fit to data
model_logreg_cv.fit(X, y)

# Tuned Parameters
print("Tuned Logistic Regression Parameters: {}".format(model_logreg_cv.best_params_))
print("Best score is {}".format(model_logreg_cv.best_score_))
```

AI Workflow

Model Evaluation - Classification

Gather → **Explore** → **Clean** → **Modify** → **Prepare** → **Train** → **Evaluate**

Get Data | Clean, Prepare & Modify Data | Train Model | Measure

CONFUSION MATRIX

		ACTUAL VALUE	
		POSITIVE	NEGATIVE
PREDICTED VALUE	POSITIVE	**TP** (True Positive)	**FP** (False Positive)
	NEGATIVE	**FN** (False Negative)	**TN** (True Negative)

Recall **Precision**

TP (True Positive) & TN (True Negative):
The prediction matches the Truth correctly
E.g. we correctly predicted an object as a car

FP (False Positive) & FN (False Negative):
The prediction does not match the Truth
E.g. we incorrectly predicted an object as a car, while it was something else

METRICS

ACCURACY

Formula: $\frac{(TP + TN)}{(TP + TN + FP + FN)}$ OR $\frac{\#CORRECT\_PREDICTIONS}{\#TOTAL}$

Summary: How well does the model perform?

Example: Our model is 95% accurate

RECALL

Formula: $\frac{(TP)}{(TP + FN)}$ OR $\frac{\#CORRECT\_POSITIVE\_PREDICTIONS}{\#TRUE\_TRUTH\_VALUES}$

Summary: How often did we wrongly classify something as not true (= false?)

Example: 5% of the time we said it was not a car, while it was a car (we could've hit it)

PRECISION

Formula: $\frac{(TP)}{(TP + FP)}$ OR $\frac{\#CORRECT\_POSITIVE\_PREDICTIONS}{\#POSITIVE\_SAMPLES}$

Summary: How often are we correct in our positive prediction? (or how much are we being wrong?)

Example: 5% of the time we said an object was a car, while it actually was not. (wrong action will be taken - e.g. increasing speed)

F-SCORE (AND F1 SCORE)

Formula: $F_\beta = (1 + \beta^2) \frac{(PRECISION * RECALL)}{(\beta^2 * PRECISION) + RECALL}$

$F_1 = 2 * \frac{(PRECISION * RECALL)}{(PRECISION + RECALL)}$

Summary: Utilize the precision and recall to create a test's accuracy through the "harmonic mean". Also known as the Sørensen–Dice Coefficient

Example: Our model is 88% accurate based on high-business impact markers (#wrong detections and #false positives)

ROC CURVE

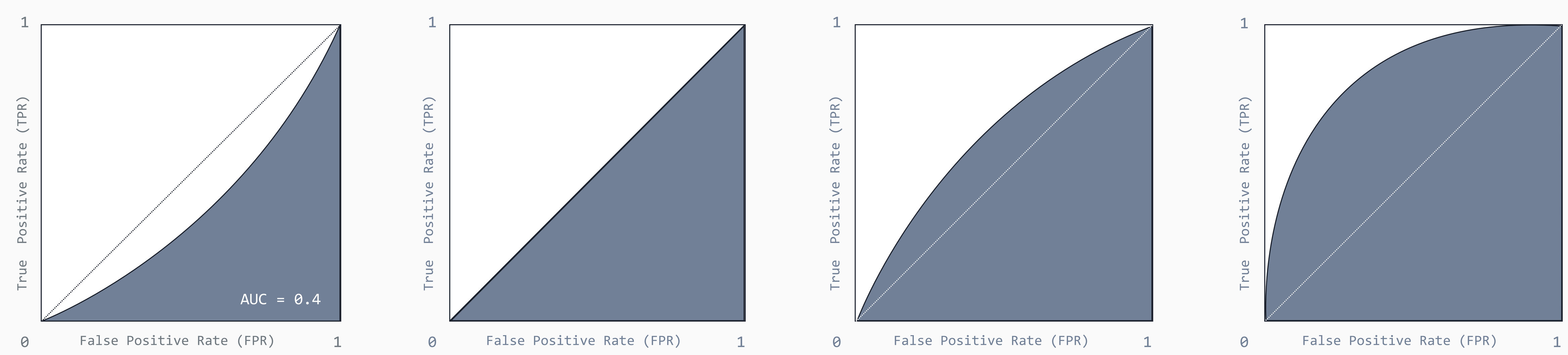
Formula: False Positive Rate (FPR) = X-Axis
True Positive Rate (TPR) = Y-Axis

$TPR = \frac{(TP)}{(TP + FN)}$ $FPR = \frac{(FP)}{(FP + TN)}$

Summary: The ROC curve allows us to select the optimal model and discard suboptimal ones.

Method:
1. Discretize the threshold for the confidence score (e.g. confidence score of [0, 1] becomes [0.0, ..., 0.9, 1.0]
2. Calculate the confusion matrix for the given threshold
3. Determine the TPR and FPR and plot them

Examples:



READ MORE

Performance Markers (https://xaviergeerinck.com/ai-performance-markers)
F1 Score (https://scikit-learn.org/stable/modules/generated/sklearn.metrics.f1_score.html#sklearn.metrics.f1_score)