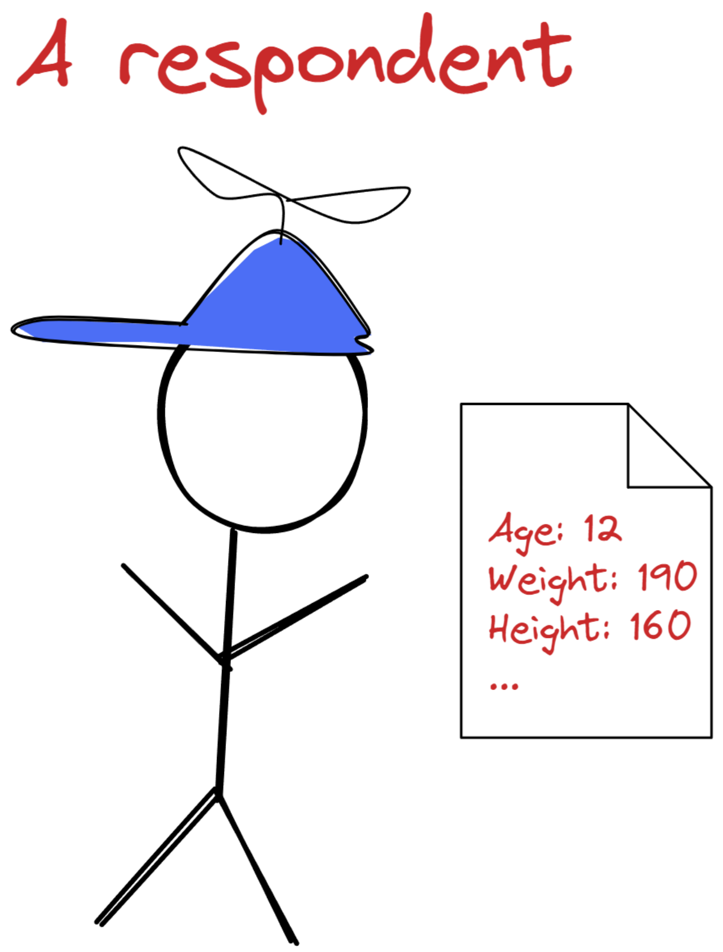
# Getting started with Isolation Forests

ANOMALY DETECTION IN PYTHON

**Bekhruz (Bex) Tuychiev**
Kaggle Master, Data Science Content Creator

# Survey data

- A sample respondent:
  - 12 years old
  - 160 cm tall
  - weighs 190 pounds

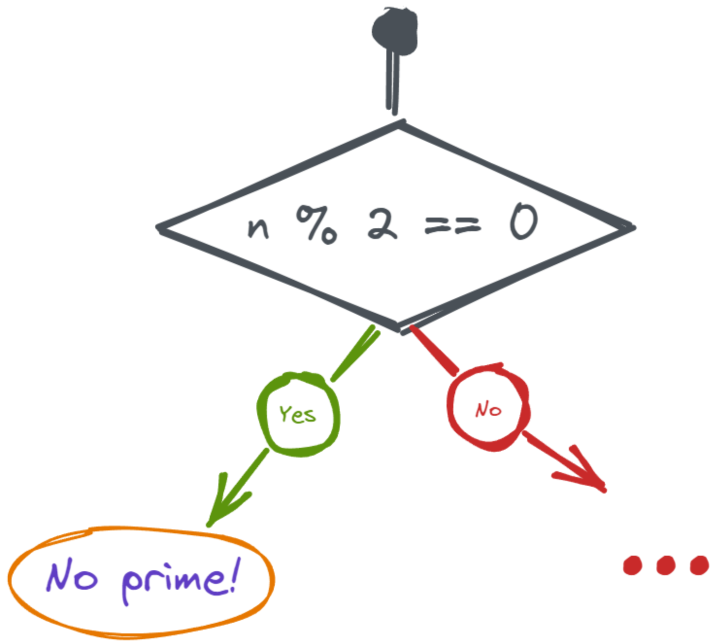


# Multivariate anomalies

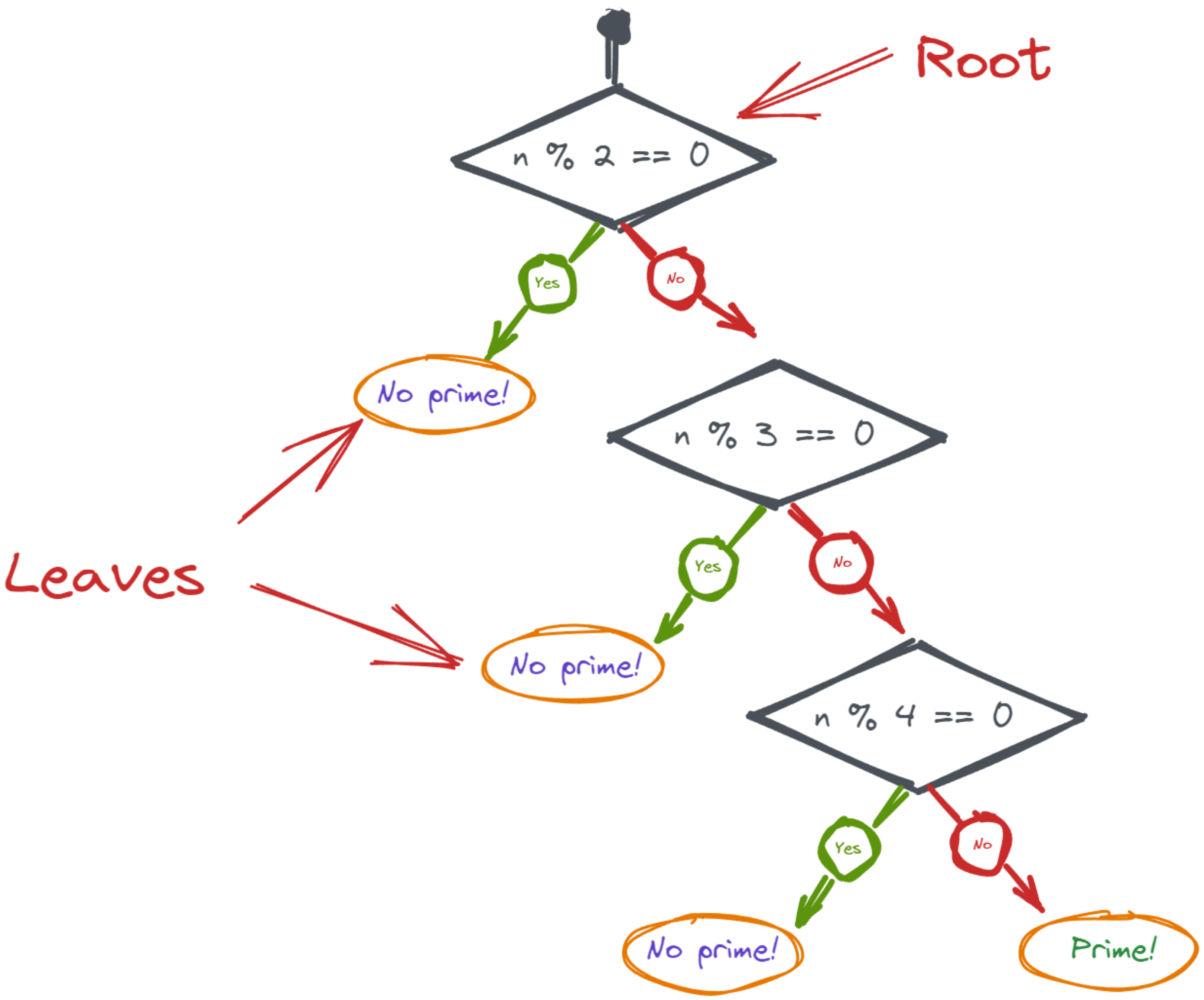
Multivariate anomalies:

- have two or more attributes
- attributes are not necessarily anomalous
- only anomalous when all attributes are considered

# Decision trees



# Decision trees

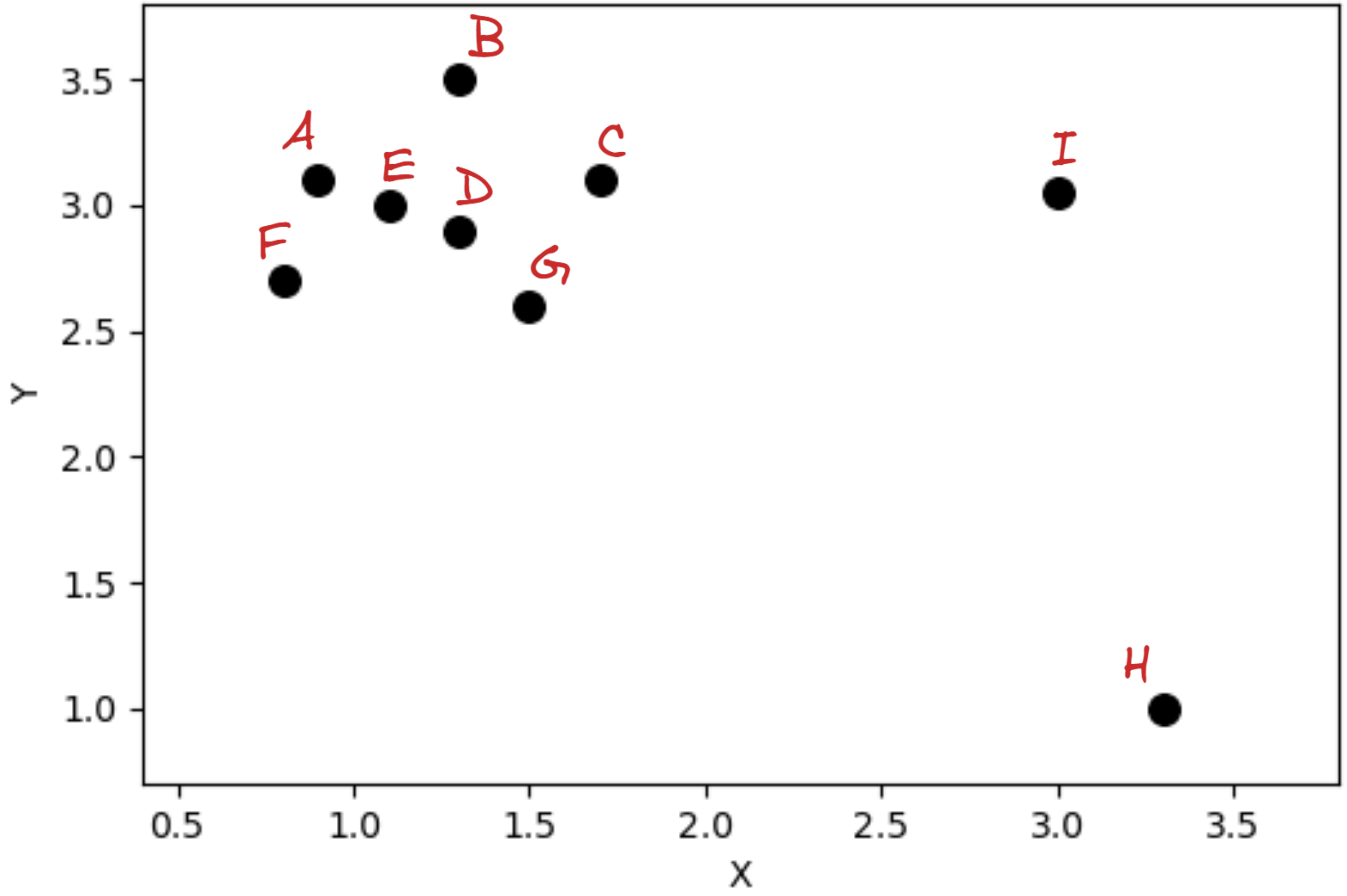


# Isolation Trees

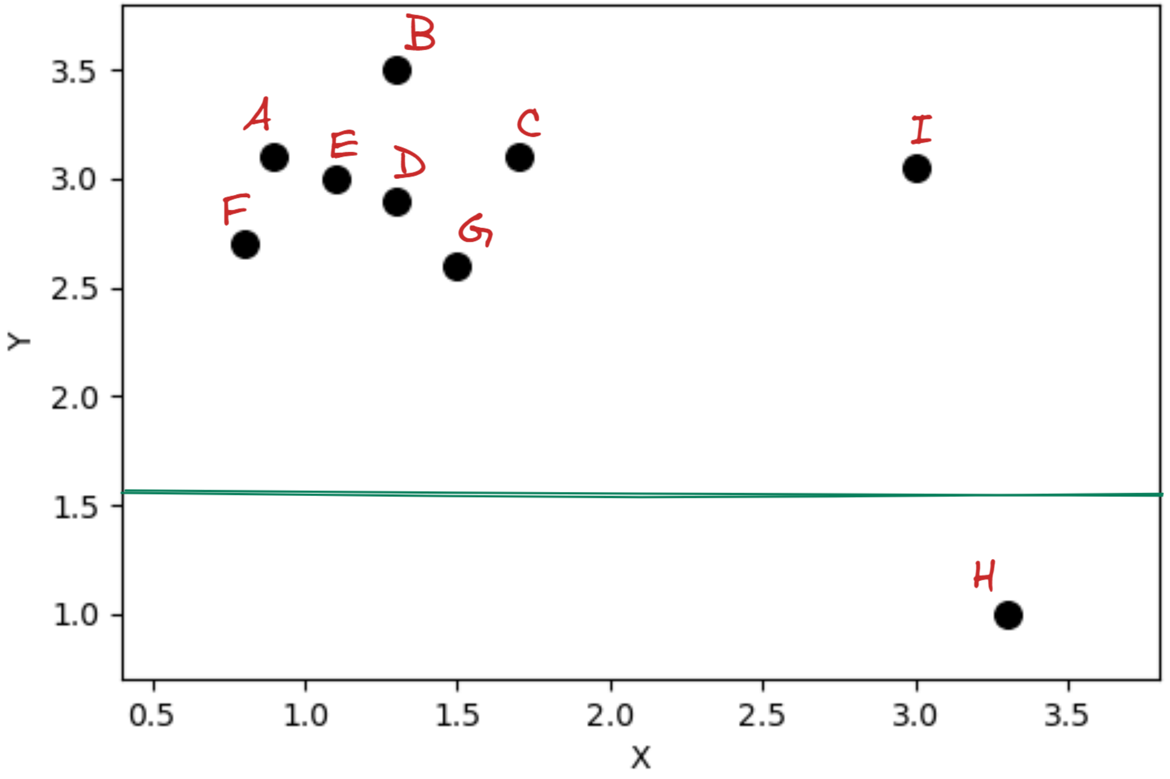
iTrees:

- short for isolation trees
- randomized versions of decision trees
- splitting (branching) occurs randomly
- random split is more likely to occur in inlier/outlier gap

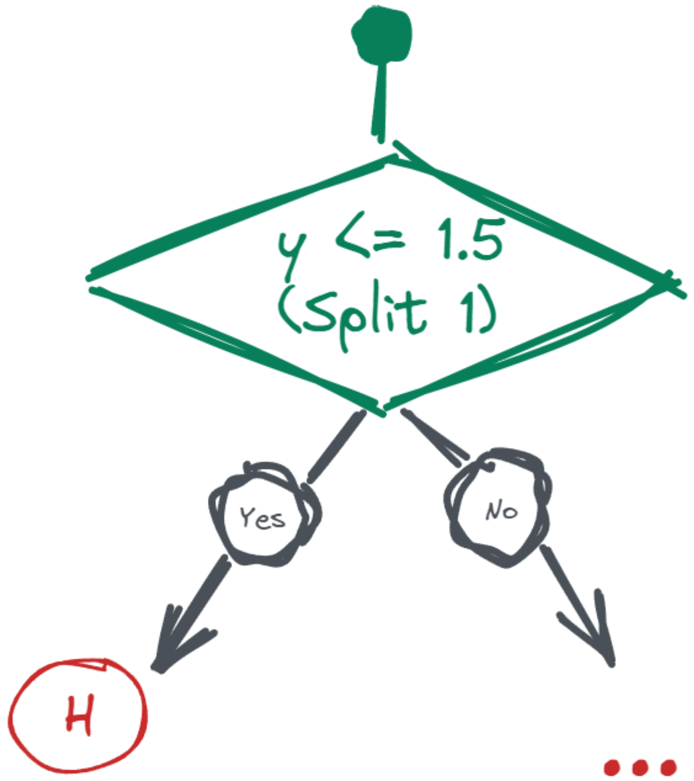
# Example 2D data



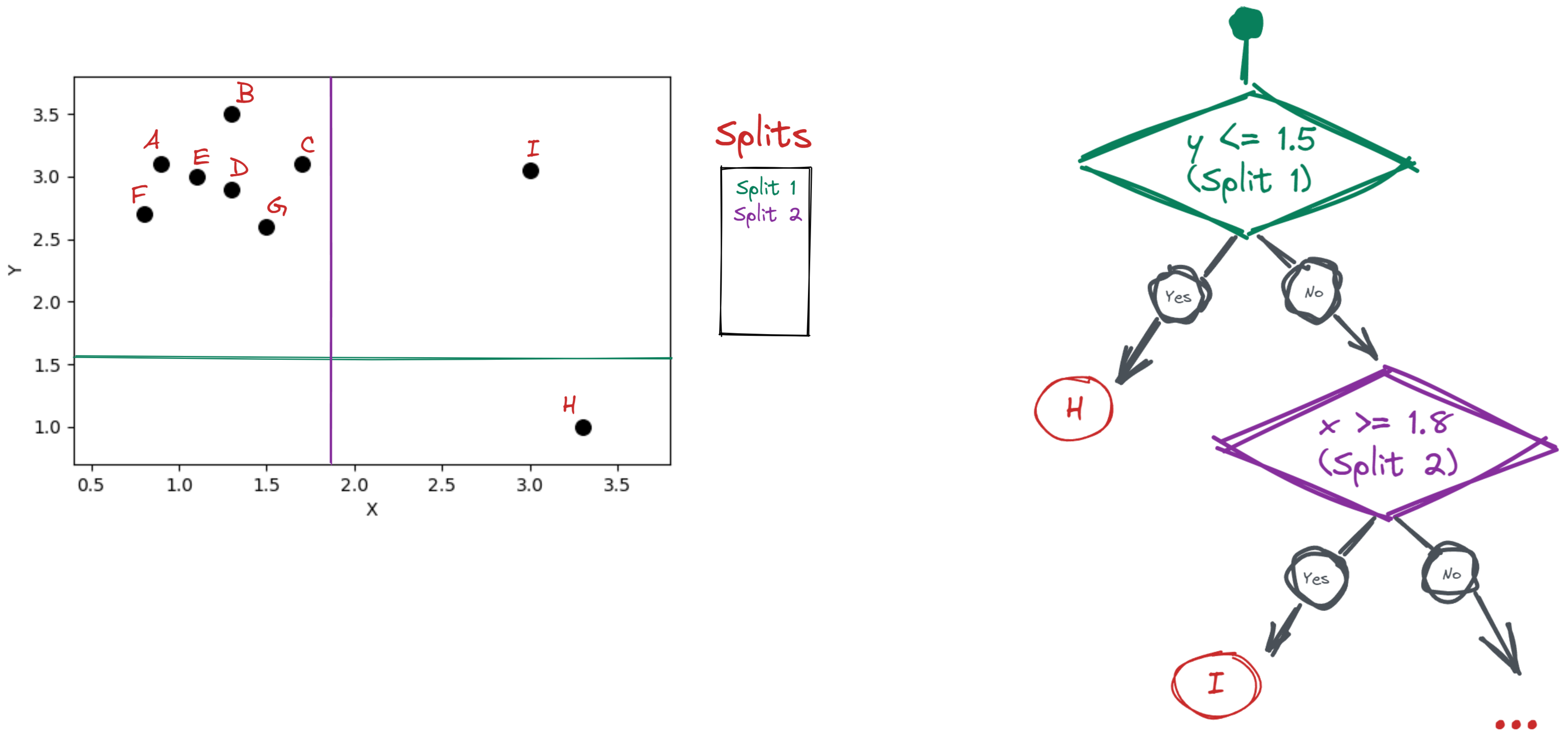
# Fitting an iTree



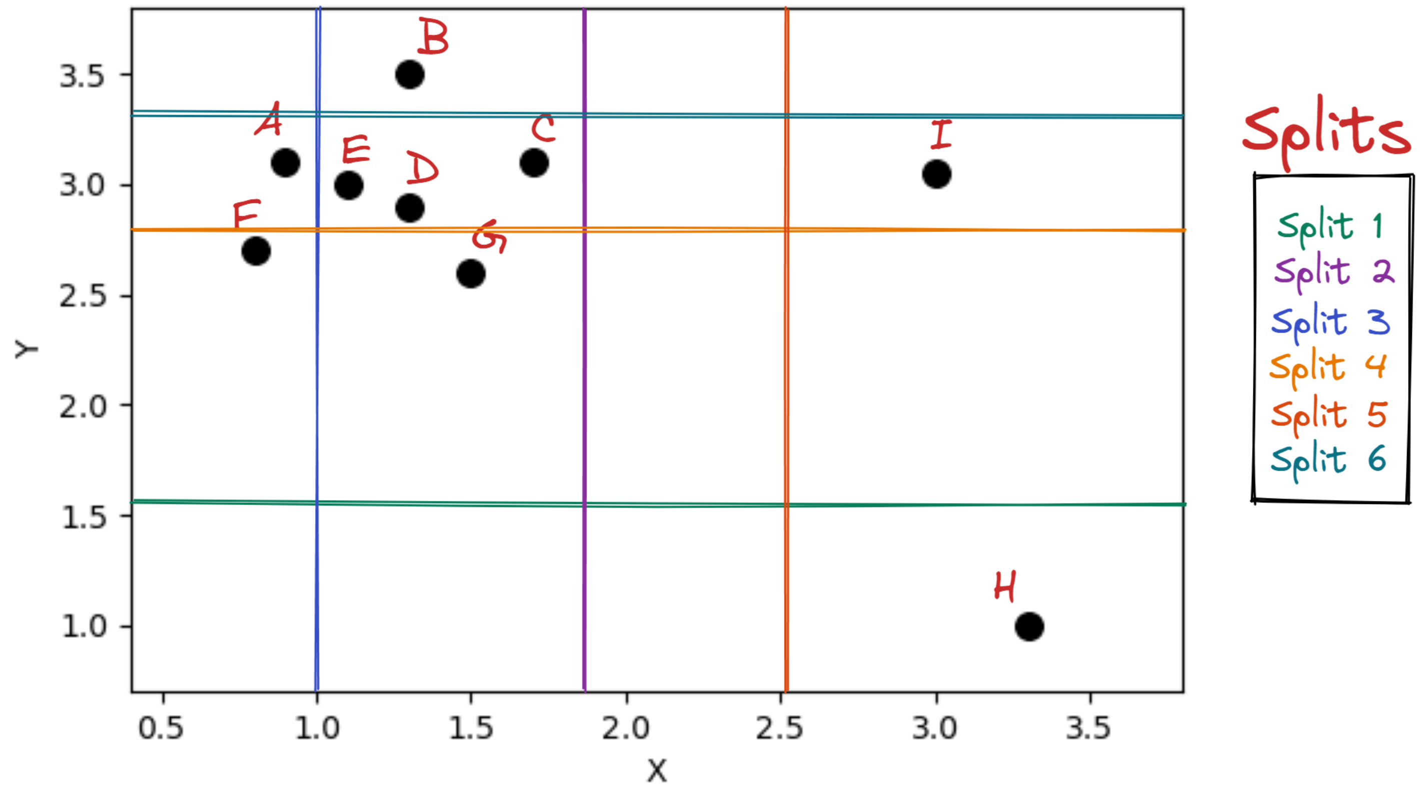


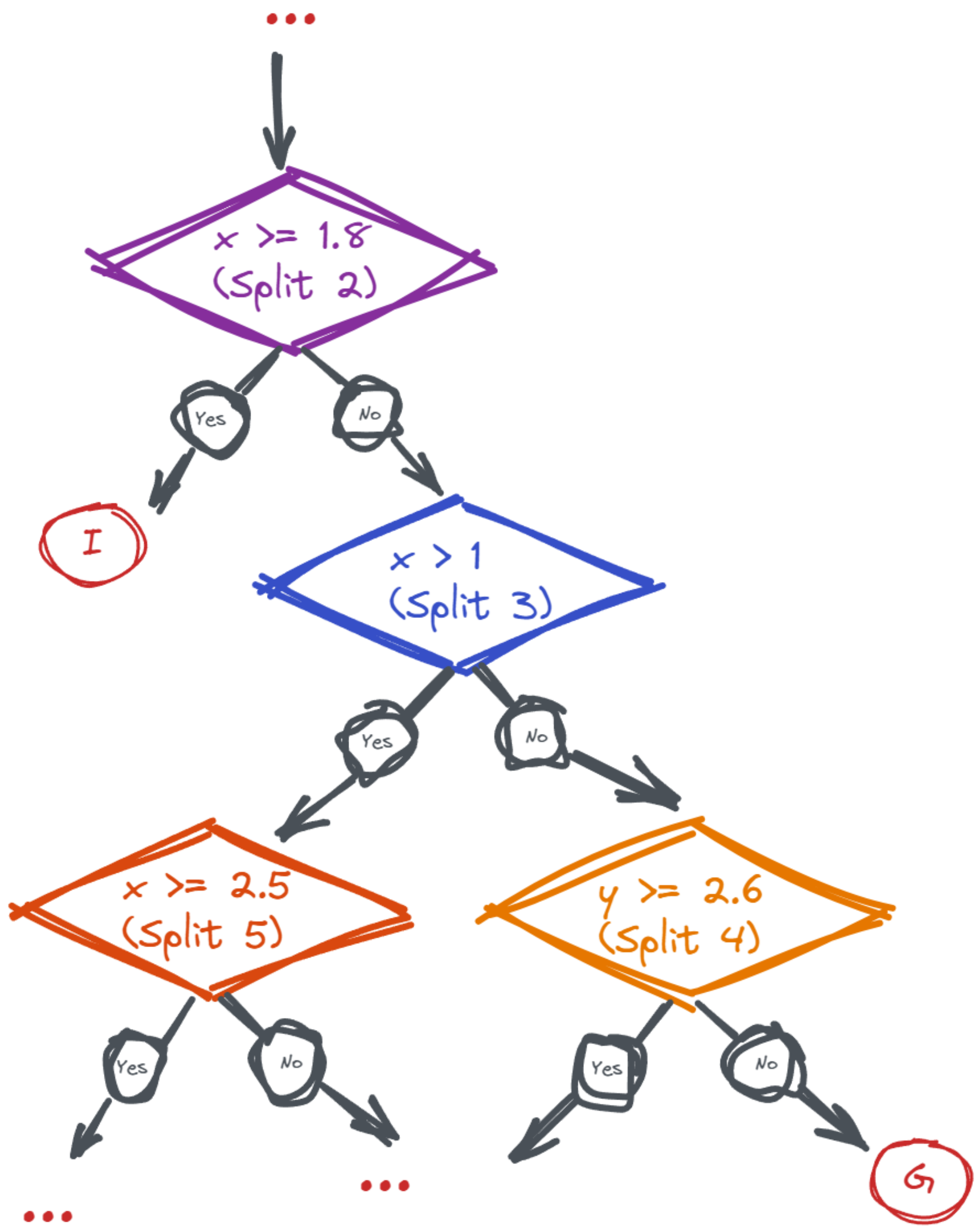


# Fitting an iTree



# Fitting an iTree

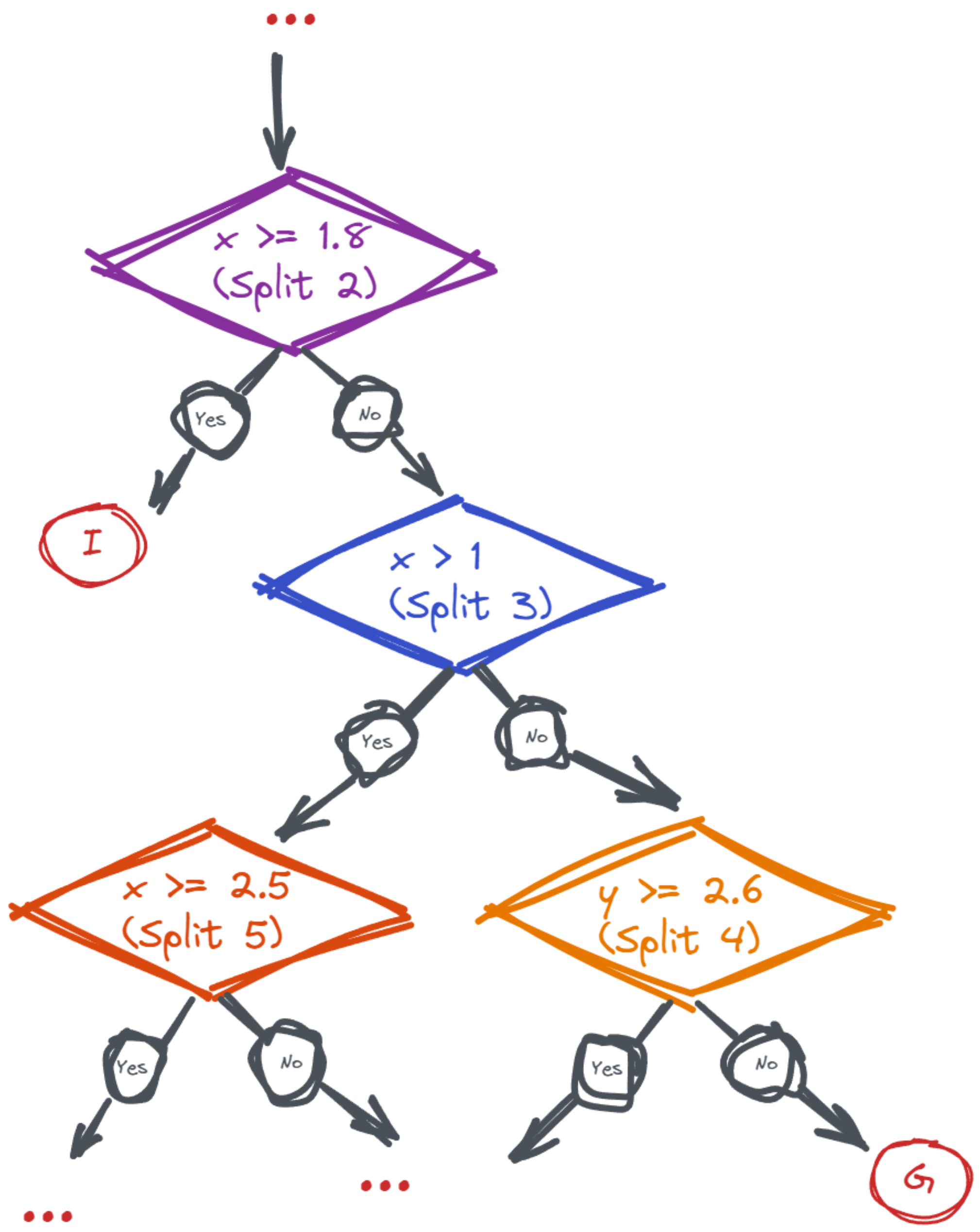


# How points are classified

Points are outliers:

- if close to the root node
- or require fewer splits



# US Airbnb data

```
import pandas as pd

airbnb_df = pd.read_csv("airbnb.csv")
```

# US Airbnb data

```
airbnb_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10000 entries, 0 to 9999
Data columns (total 6 columns):
 #   Column                          Non-Null Count  Dtype
 0   minimum_nights                  10000 non-null  int64
 1   number_of_reviews               10000 non-null  int64
 2   reviews_per_month               10000 non-null  float64
 3   calculated_host_listings_count  10000 non-null  int64
 4   availability_365                10000 non-null  int64
 5   price                           10000 non-null  int64
dtypes: float64(1), int64(5)
```

# fit_predict

```python
from pyod.models.iforest import IForest

iforest = IForest()
labels = iforest.fit_predict(airbnb_df)

print(labels)
```

```
array([0, 0, 0, ..., 1, 0, 0])
```

# Filter outliers

```
outliers = airbnb_df[labels == 1]

print(outliers.shape)
```

```
(1000, 6)
```

# Let's practice!

ANOMALY DETECTION IN PYTHON

# Overview of Isolation Forest hyperparameters

ANOMALY DETECTION IN PYTHON

**Bekhruz (Bex) Tuychiev**
Kaggle Master, Data Science Content Creator

# Most important hyperparameters

Hyperparameters which influence `IForest` the most:

- `contamination`
- `n_estimators`
- `max_samples`
- `max_features`

# What is contamination?

How `IForest` classifies data points:

1. Raw anomaly scores are generated
2. Set a threshold called `contamination`
3. The highest percentage of anomaly scores denoted with `contamination` are chosen as outlying datapoints

# Setting contamination

```
from pyod.models.iforest import IForest

# Accepts a value between 0 and 0.5
iforest = IForest(contamination=0.05)
```

# What is n_estimators?

```
# More trees for larger datasets
iforest = IForest(n_estimators=1000)


iforest.fit(airbnb_df)
```

# max_samples and max_features

```
iforest = IForest(n_estimators=200, max_samples=0.6, max_features=0.9)


iforest.fit(airbnb_df)
```

# Tree growth

- iTrees:
  - grow in a randomized fashion
  - split is chosen randomly between feature min and max
  - grow until:
    - all points are isolated
    - maximum depth is reached

# Max tree depth

- Equals the logarithm of the sample size

# IForest advantages

- Very efficient on large datasets
- Doesn't need all normal instances like other algorithms
- No statistical assumptions
- Performs well out-of-the-box

# Challenges of outlier detection

- Supervised-learning models rely on metrics like RMSE or log loss
- Outlier detection is an unsupervised-learning problem
- Outlier classifiers should be combined with supervised-learning models

# Let's practice!

ANOMALY DETECTION IN PYTHON

# Hyperparameter tuning of Isolation Forest

ANOMALY DETECTION IN PYTHON

**Bekhruz (Bex) Tuychiev**
Kaggle Master, Data Science Content Creator

# Tuning contamination

- No determined way of tuning it
- Have to rely on:
  - intuition
  - EDA insights
  - domain knowledge
  - business expectations

# Survey example

- Research similar surveys
- Learn the proportion of the poorest and the wealthiest
- Research is better than blindly choosing a value

# Big Mart sales data

```python
import pandas as pd

big_mart = pd.read_csv("big_mart_sales.csv")
big_mart.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 7060 entries, 0 to 7059
Data columns (total 5 columns):
 #   Column            Non-Null Count  Dtype
 0   weight            7060 non-null   float64
 1   fat_content       7060 non-null   object
 2   type              7060 non-null   object
 3   max_retail_price  7060 non-null   float64
 4   sales             7060 non-null   float64
dtypes: float64(3), object(2)
```

# Encode categoricals

```
big_mart = pd.get_dummies(big_mart)
```

| | weight | max_retail_price | sales | fat_content_low_fat | fat_content_regular |
|---|---|---|---|---|---|
| 0 | 9.30 | 249.8092 | 3735.1380 | 1 | 0 |
| 1 | 5.92 | 48.2692 | 443.4228 | 0 | 1 |
| 2 | 17.50 | 141.6180 | 2097.2700 | 1 | 0 |
| 3 | 19.20 | 182.0950 | 732.3800 | 0 | 1 |
| 4 | 8.93 | 53.8614 | 994.7052 | 1 | 0 |

# evaluate_outlier_classifier

```python
def evaluate_outlier_classifier(model, data):
    # Get labels
    labels = model.fit_predict(data)

    # Return inliers
    return data[labels == 0]
```

# evaluate_regressor

```
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error
```

# evaluate_regressor

```
def evaluate_regressor(inliers):
    X = inliers.drop("sales", axis=1)
    y = inliers[['sales']]

    X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=10)

    lr = LinearRegression()
    lr.fit(X_train, y_train)

    preds = lr.predict(X_test)
    rmse = mean_squared_error(y_test, preds, squared=False)

    return round(rmse, 3)
```

# Tuning contamination

```
contaminations = [0.05, 0.1, 0.2, 0.3]
scores = dict()

for c in contaminations:
    # Instantiate IForest with the current c
    iforest = IForest(contamination=c, random_state=10)

    # Get inliers with the current IForest
    inliers = evaluate_outlier_classifier(iforest, big_mart)

    # Calculate and store RMSE into scores
    scores[c] = evaluate_regressor(inliers)
```

# Look at the output

```python
print(scores)
```

```
{0.05: 1148.555, 0.1: 1147.48, 0.2: 1082.307, 0.3: 1029.33}
```

# Tuning multiple hyperparameters

```
estimators = [100, 200, 300,]
max_samples = [0.6, 0.8, 1]
scores = dict()
```

# Cartesian product

```python
from itertools import product

list(product(estimators, max_samples))
```

```
[(100, 0.6),
 (100, 0.8),
 (100, 1),
 (200, 0.6),
 (200, 0.8),
 (200, 1),
 (300, 0.6),
 (300, 0.8),
 (300, 1)]
```

# Inside the loop

```
estimators = [100, 200, 300,]
max_samples = [0.6, 0.8, 1]
scores = dict()

for e, m in product(estimators, max_samples):
    # Instantiate an IForest
    iforest = IForest(n_estimators=e, max_samples=m, contamination=.3)

    # Get the inliers with the current IForest
    inliers = evaluate_outlier_classifier(iforest, big_mart)

    # Calculate and store RMSE into scores
    scores[(e, m)] = evaluate_regressor(inliers)
```

# Looking at the output

```
print(scores)
```

```
{(100, 0.6): 959.398,
 (100, 0.8): 986.056,
 (100, 1): 1195.875,
 (200, 0.6): 947.628,
 (200, 0.8): 933.115,
 (200, 1): 1195.875,
 (300, 0.6): 949.412,
 (300, 0.8): 935.962,
 (300, 1): 1195.875}
```

# Parallel execution

```
# Faster compuation with n_jobs=-1
iforest = IForest(n_estimators=1000, n_jobs=-1)


iforest.fit(big_mart)
```

# Let's practice!

ANOMALY DETECTION IN PYTHON

# Interpreting the output of IForest

ANOMALY DETECTION IN PYTHON

**Bekhruz (Bex) Tuychiev**
Kaggle Master, Data Science Content Creator

# An alternative

```python
from pyod.models.iforest import IForest

iforest = IForest(contamination=0.2, max_features=0.5, random_state=1)

iforest = iforest.fit(airbnb_df)

labels = iforest.labels_
print(labels)
```

```
array([0, 0, 0, ..., 1, 0, 0])
```

# Predictions on new data

```
import numpy as np

new_data = [[34, 40, 0.44, 3, 2, 90]]

iforest.predict(new_data)
```

```
array([0])
```

# Probability scores

```
all_probs = iforest.predict_proba(airbnb_df)
print(all_probs)
```

```
array([[0.71401381, 0.28598619],
       [0.75553703, 0.24446297],
       [0.6844169 , 0.3155831 ],
       ...,
       ])
```

```
print(all_probs.shape)
```

```
(10000, 2)
```

# Outlier probability scores

```python
outliers = airbnb_df[iforest.labels_ == 1]
outlier_probs = iforest.predict_proba(outliers)

print(outlier_probs[:10])
```

```
array([[0.51999538, 0.48000462],
       [0.61789522, 0.38210478],
       [0.61802032, 0.38197968],
       [0.35184434, 0.64815566],
       [0.57533286, 0.42466714],
       [0.59038933, 0.40961067],
       [0.57677613, 0.42322387],
       [0.54158826, 0.45841174],
       [0.49118093, 0.50881907],
       [0.21387357, 0.78612643]])
```

# Abandoning contamination

```
# Fit to Airbnb
iforest = IForest(max_features=0.5, random_state=1)
iforest.fit(airbnb_df)

# Calculate probabilities
probs = iforest.predict_proba(airbnb_df)

# Propbs for outliers
outlier_probs = probs[:, 1]
```

# Abandoning contamination

```
# Filter only when probability is higher than 65%
outliers = airbnb_df[outlier_probs >= 0.65]


print(len(outliers))
```

```
193
```

# Let's practice!

ANOMALY DETECTION IN PYTHON