# The two cases with trees

Machine Learning in Finance · Session 13

# The two cases

### A house in Aarhus V

Session 7 valued every 2024 sale with linear regression in logs on 28 columns. It missed by 1.26 million kroner, with 15 percent of the loans underwater.

### An hour in West Denmark

Session 10 asked whether tomorrow’s price at an hour beats today’s. Logistic regression ranked the hours of 2024 with an AUC of 0.833, and k-NN with 0.844.

Both sessions came before trees, forests and boosting. Today the same tables, splits and scores go to those models.

# Trees on the course tables

| table | the linear model | a forest | boosted trees |
|---|---|---|---|
| volatility, test RMSE | 0.218, ridge | 0.226 | 0.217 and 0.225 |
| credit, test AUC | 0.748, logistic regression | 0.777 | 0.777 |

On the volatility table the trees drew level with ridge. On the credit table they gained 0.029 of AUC. The two cases are two more tests.

# Today

- the house in **Aarhus V**: a tree, a forest and boosted trees on the 2024 sales
- the hours of **West Denmark**: the same models on tomorrow's price
- **choosing** a model: accuracy, compute and explanations
- the **course** and the **exam**, then your questions

There are no exercises and no case for this session.

# 1 · The house in Aarhus V

PART 1

Session 7’s valuation of every 2024 sale, against a tree, a forest and boosted trees.

# The house

| 150 m² | 1973 | 5 | villa | Aarhus V | 3.6 km |
|---|---|---|---|---|---|
| living area | year built | rooms | type | postal town | from the cathedral |

It sold for 3,800,000 kroner in January 2024. Session 7’s model valued it at 4,341,389, 14 percent above the price.

# The libraries of the course

Python Code

```
import numpy as np
import pandas as pd
from sklearn.linear_model import Ridge, LogisticRegression
from sklearn.preprocessing import StandardScaler
from sklearn.pipeline import Pipeline
from sklearn.model_selection import KFold, GridSearchCV
from sklearn.metrics import roc_auc_score
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor, RandomForestClassifier
from xgboost import XGBRegressor, XGBClassifier
```

NumPy and pandas came in Session 3, the linear models between Sessions 5 and 9, the trees in Session 11 and XGBoost in Session 12.

# The table and the split

Python Code

```
houses = pd.read_csv('data/aarhus_houses.csv')
wide = pd.get_dummies(houses, columns=['town', 'type'], dtype=int)
columns = [c for c in wide.columns if c not in ['price', 'sold', 'quarter', 'date', 'zip', 'lat', 'lon']]
train = wide[wide['sold'] <= 2023]
test = wide[wide['sold'] == 2024]
print(len(train), len(test), len(columns))
```

```
6212 1409 28
```

These are the same sales as in Session 7: 6,212 from 2021 to 2023 to learn from, and the 1,409 of 2024 to check on.

# Two scores

Python Code

```
def rmse(actual, predicted):
    return np.sqrt(((actual - predicted) ** 2).mean())

def underwater(actual, predicted):
    return (0.8 * predicted > actual).mean()
```

The first is the error in kroner. The second is the share of 2024 sales where a loan of 80 percent of the valuation is more than the price.

# Session 7's valuation

Python Code

```
folds = KFold(n_splits=5, shuffle=True, random_state=0)
pipe = Pipeline([('scale', StandardScaler()), ('ridge', Ridge())])
grid = {'ridge__alpha': [0.01, 0.1, 1, 10, 100, 1000]}
valuation = GridSearchCV(pipe, grid, cv=folds, scoring='neg_root_mean_squared_error')
valuation.fit(train[columns], np.log(train['price']))
valued = np.exp(valuation.predict(test[columns]))
print(round(rmse(test['price'], valued)), round(underwater(test['price'], valued), 3))
```

```
1257437 0.151
```

Ridge in logs, with alpha chosen on the folds: 1,257,437 kroner, and 15.1 percent of the loans underwater. Every new model is held to these two numbers.

# A tree

Python Code

```
tree = DecisionTreeRegressor(min_samples_leaf=20, random_state=0)
tree.fit(train[columns], np.log(train['price']))
valued = np.exp(tree.predict(test[columns]))
print(round(rmse(test['price'], valued)), round(underwater(test['price'], valued), 3))
print(tree.get_depth(), tree.get_n_leaves())
```

```
1270418 0.138
19 238
```

Leaves of at least 20 sales were chosen on the folds. The tree is 13,000 kroner worse than ridge and puts 13.8 percent of the loans underwater.

# A forest

Python Code

```
forest = RandomForestRegressor(n_estimators=100, max_features='sqrt', random_state=0)
forest.fit(train[columns], np.log(train['price']))
valued = np.exp(forest.predict(test[columns]))
print(round(rmse(test['price'], valued)), round(underwater(test['price'], valued), 3))
```

```
1113028 0.121
```

A hundred trees, each grown on a bootstrap sample and averaged, cut the error by 144,000 kroner to 1,113,028. The underwater share falls to 12.1 percent.

# Boosted trees, stopped on 2023

Python Code

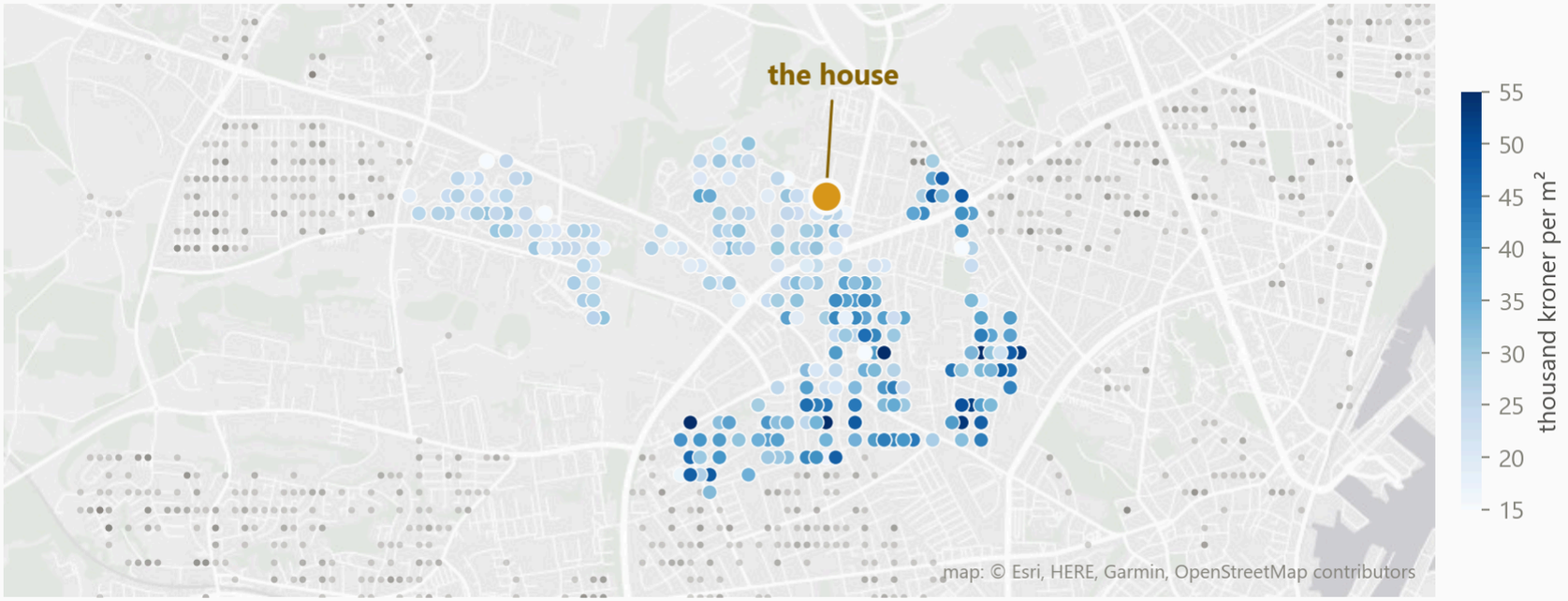
```
fit_rows, stop_rows = train[train['sold'] <= 2022], train[train['sold'] == 2023]
stopper = XGBRegressor(n_estimators=3000, learning_rate=0.05, max_depth=3,
                       early_stopping_rounds=100, random_state=0)
stopper.fit(fit_rows[columns], np.log(fit_rows['price']),
            eval_set=[(stop_rows[columns], np.log(stop_rows['price']))], verbose=False)
boosted = XGBRegressor(n_estimators=stopper.best_iteration + 1, learning_rate=0.05, max_depth=3, random_state=0)
boosted.fit(train[columns], np.log(train['price']))
valued = np.exp(boosted.predict(test[columns]))
print(stopper.best_iteration + 1, round(rmse(test['price'], valued)), round(underwater(test['price'], valued), 3))
```

```
974 1166375 0.116
```

Depth 3 and 974 trees did best on the 2023 sales. Refitted on all three years, the model misses by 1,166,375 kroner, with 11.6 percent underwater.

# Prices inside one town

Sales in Aarhus V, 2021 to 2024, by price per square metre; other towns in grey




Every house in Aarhus V gets the same town premium, yet one sale in ten there went below 21,500 kroner per square metre and one in ten above 44,900.

# Coordinates as columns

Python Code

```
place = ['area', 'rooms', 'built', 'centre_km', 'lat', 'lon', 'type_villa']
stopper = XGBRegressor(n_estimators=3000, learning_rate=0.05, max_depth=4,
                       early_stopping_rounds=100, random_state=0)
stopper.fit(fit_rows[place], np.log(fit_rows['price']),
            eval_set=[(stop_rows[place], np.log(stop_rows['price']))], verbose=False)
boosted_place = XGBRegressor(n_estimators=stopper.best_iteration + 1, learning_rate=0.05, max_depth=4, random_state=0)
boosted_place.fit(train[place], np.log(train['price']))
valued = np.exp(boosted_place.predict(test[place]))
print(stopper.best_iteration + 1, round(rmse(test['price'], valued)), round(underwater(test['price'], valued), 3))
```

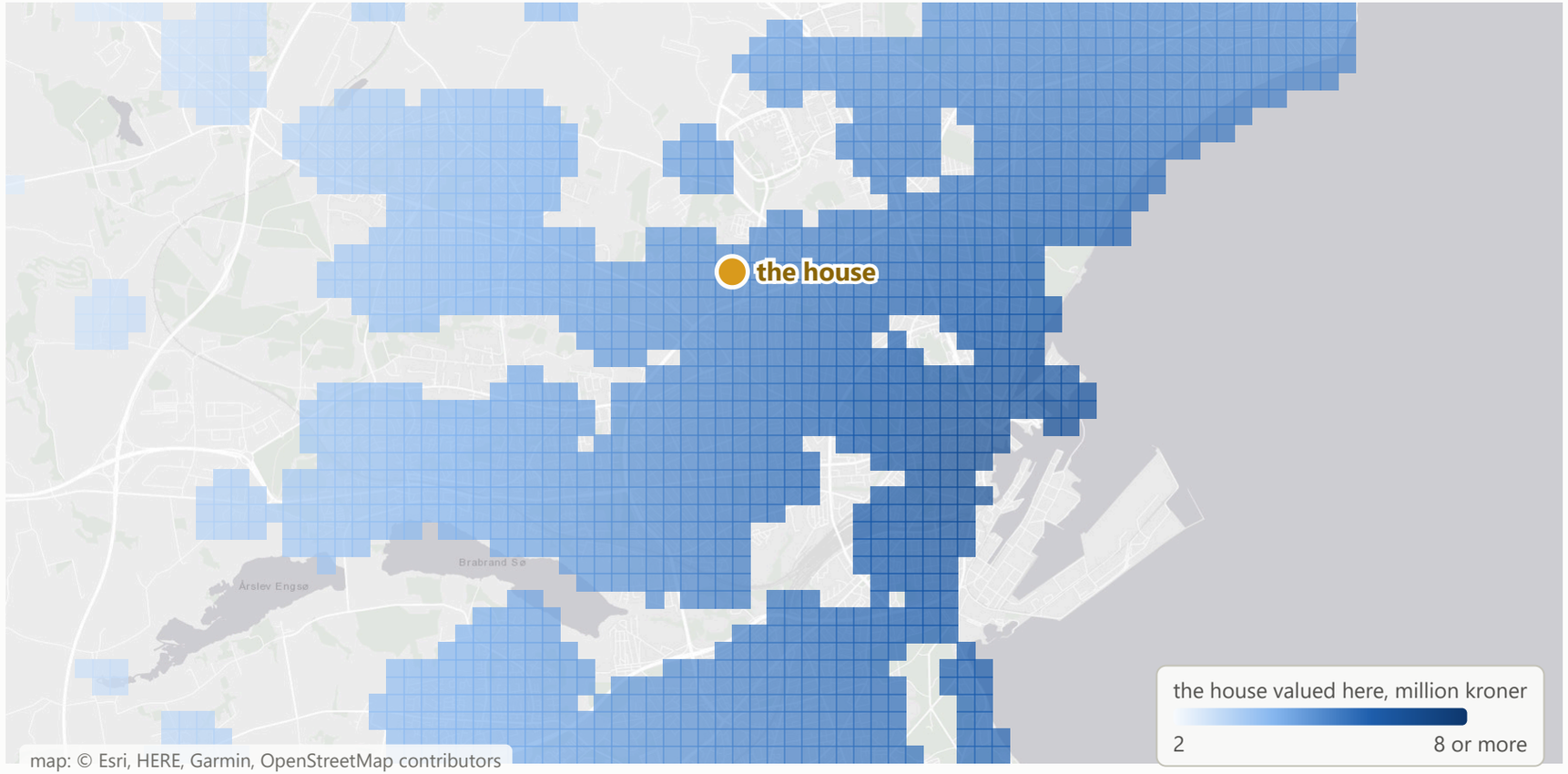
```
1017 1030321 0.099
```

Latitude and longitude replace the 22 town columns, and depth 4 did best on 2023. The error falls to 1,030,321 kroner and the underwater share to 9.9 percent.

# The house anywhere in the inner city PLAY

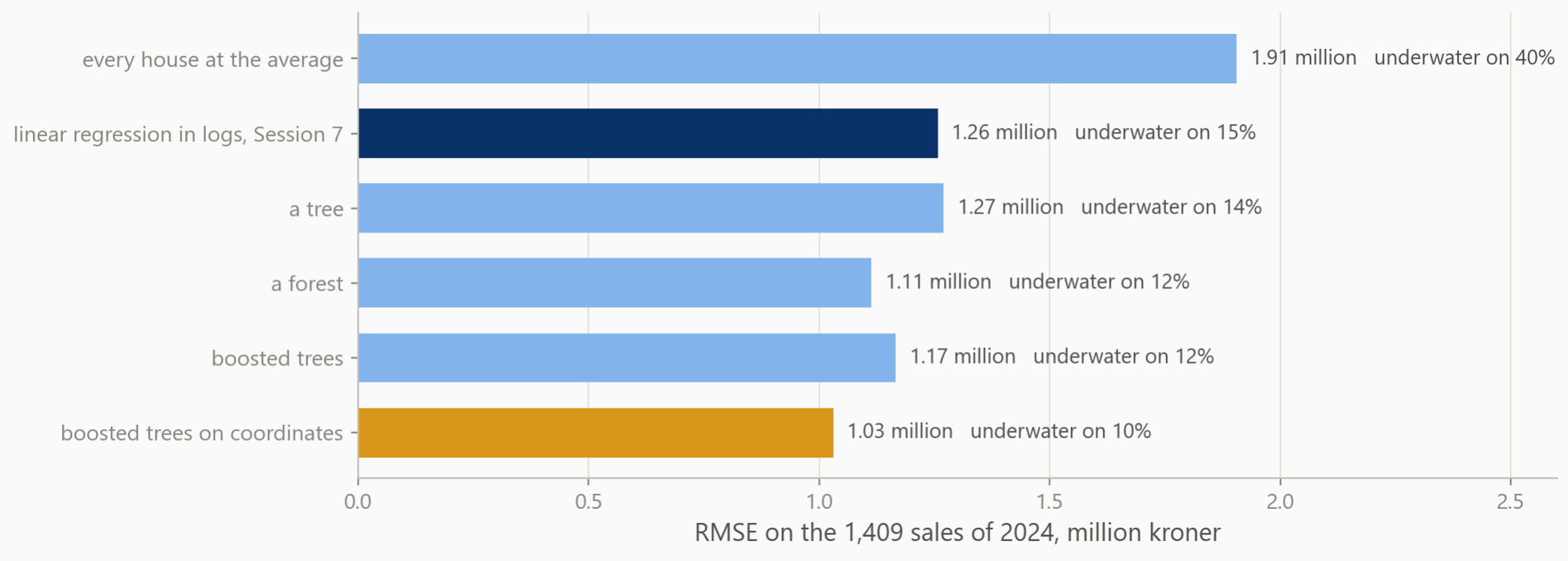
linear regression | boosted trees

At the house's own spot: **4.99 million**. Point at the map to read another spot.



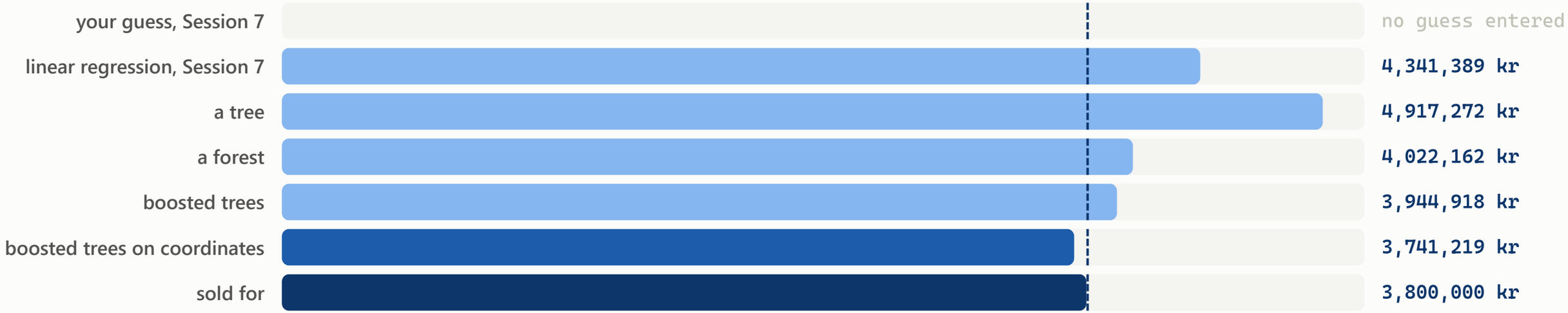

Each square is the house, moved there. Linear regression on the same columns can only tilt the map and follow the distance, and misses by 1.31 million.

# The error, model by model




The forest and both boosted models beat ridge. Boosting on coordinates cuts the underwater share from 15 percent to 10.

# The house, valued by every model




Boosted trees on coordinates came closest on this house, 58,781 kroner below the price. One house says little; the scores on the 1,409 sales say more.

# 2 · Tomorrow’s price at this hour

PART 2

Session 10’s question on the same 2024 hours, with the same scores, and with a forest and boosted trees.

# The question, again

| 0.833 | 0.844 | 957,809 kr |
| --- | --- | --- |
| logistic regression, AUC on 2024 | k-NN, AUC on 2024 | logistic regression, trading 2024 at a threshold of 0.6 |

An hour is up when its price is higher than at the same hour the day before. These were Session 10’s best models of that label.

# The hours, as in Session 10

Python Code

```
power = pd.read_csv('data/power.csv', parse_dates=['time']).set_index('time')
power['price_before'] = power.groupby('hour')['price'].shift(1)
power['up'] = (power['price'] > power['price_before']).astype(int)
power['wind_change'] = power['wind'] - power.groupby('hour')['wind'].shift(1)
power['change_before'] = power['price_before'] - power.groupby('hour')['price'].shift(2)
days = pd.get_dummies(power['weekday'], prefix='weekday', dtype=int)
power = power.join(days).dropna()
power_train, power_test = power.loc[:'2023'], power.loc['2024']
print(len(power_train), len(power_test))
```

```
17422 8663
```

The columns are built as in Session 10, with the weekday as seven columns: 17,422 hours to learn from and 8,663 to check on.

# Boosted trees on the same columns

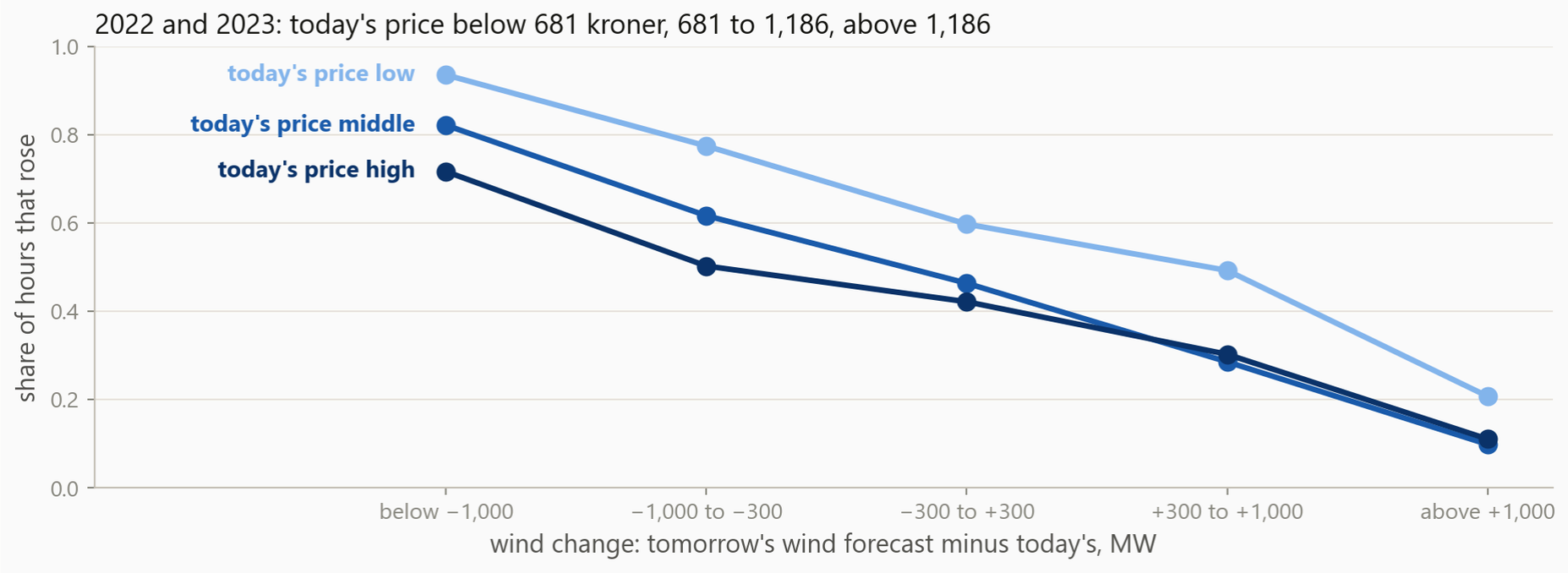
Python Code

```
near = ['wind_change', 'change_before', 'weekday', 'hour']
boosted_up = XGBClassifier(n_estimators=319, learning_rate=0.05, max_depth=3, random_state=0)
boosted_up.fit(power_train[near], power_train['up'])
p = boosted_up.predict_proba(power_test[near])[:, 1]
print(roc_auc_score(power_test['up'], p))
```

```
0.8426027446181137
```

Depth 3 and 319 trees did best on 2023. The trees match k-NN on 2024, 0.843 against 0.844, and do no better than logistic regression on the folds.

# Today’s price and the wind




After a forecast fall of over 1,000 MW, 94 percent of hours rose when today’s price was low and 72 percent when it was high.

# Three more columns

Python Code

```
more = near + ['price_before', 'wind', 'solar']
boosted_more = XGBClassifier(n_estimators=58, learning_rate=0.05, max_depth=4, random_state=0)
boosted_more.fit(power_train[more], power_train['up'])
p_more = boosted_more.predict_proba(power_test[more])[:, 1]
print(roc_auc_score(power_test['up'], p_more))
```

```
0.8563612276052337
```

The trees now also see today’s price and the wind and sun forecasts themselves. With 58 trees of depth 4, chosen on 2023, the AUC rises to 0.856.

# Your turn 1

Give logistic regression the same three columns. How much does it gain, and why does it gain less than the trees?

Python Code

```
features = ['wind_change'] + list(days.columns) + ['change_before']
logit = Pipeline([('scale', StandardScaler()), ('logit', LogisticRegression())])
logit.fit(power_train[features + [...]], power_train['up'])
p_logit = logit.predict_proba(power_test[features + [...]])[:, 1]
print(roc_auc_score(power_test['up'], p_logit))
```

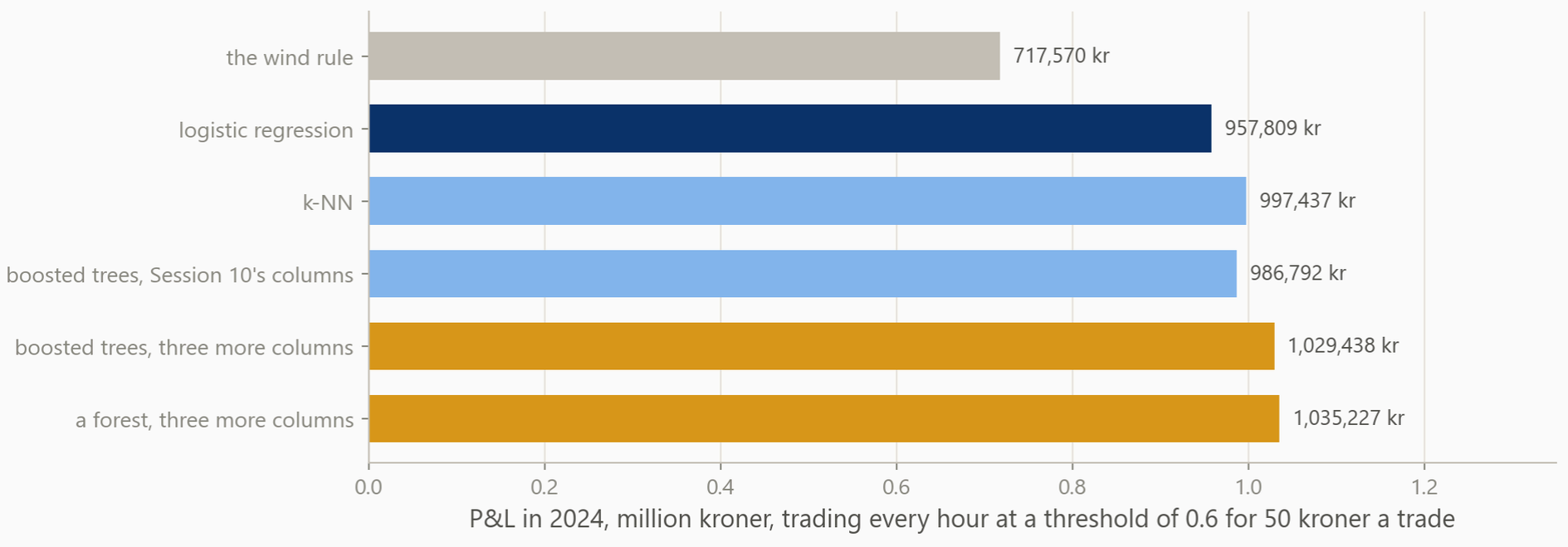
`features` is Session 10's list, which scored 0.833. Replace both `[...]` with the three new columns.

# Your turn 1: answer

```
logit.fit(power_train[features + ['price_before', 'wind', 'solar']], power_train['up'])
p_logit = logit.predict_proba(power_test[features + ['price_before', 'wind', 'solar']])[
```

The AUC is 0.836, up 0.003, where the trees gained 0.014. Logistic regression keeps one slope for the wind change at every price level; the trees need not.

# The P&L in 2024




The wind rule trades at one half, the models at 0.6. The forest earns 77,418 kroner more than logistic regression in 2024.

# Your eight hours, again

| tomorrow, at the hour | what happened | your first answer | your revised answer | logistic regression | k-NN | a forest | boosted trees |
|---|---|---|---|---|---|---|---|
| Tue 2 Jan, 13:00 | higher | **50%** | **50%** | 10% | 10% | 22% | 17% |
| Wed 24 Apr, 19:00 | higher | 50% | 50% | 60% | 61% | 60% | **61%** |
| Thu 23 May, 12:00 | higher | 50% | 50% | 94% | 89% | **96%** | 90% |
| Fri 12 Jul, 20:00 | not higher | 50% | 50% | **36%** | 40% | 43% | 41% |
| Sat 31 Aug, 10:00 | not higher | 50% | 50% | 25% | **16%** | 20% | 23% |
| Sun 15 Sep, 09:00 | not higher | 50% | 50% | 61% | **39%** | 65% | 74% |
| Thu 10 Oct, 18:00 | not higher | 50% | 50% | 6% | 6% | **2%** | 5% |
| Mon 2 Dec, 17:00 | higher | 50% | 50% | 88% | **89%** | 78% | 78% |
| **Brier score, average** | | **0.250** | **0.250** | **0.194** | **0.166** | **0.185** | **0.209** |
| **log loss, average** | | **0.693** | **0.693** | **0.590** | **0.532** | **0.525** | **0.603** |

Shaded: the closest forecast on each hour, and the lowest score. Your answers come from the Session 10 deck, opened in this browser. Answers not set count as 50 percent.

Of the four models, k-NN has the lowest Brier score here and the forest the lowest log loss. Eight hours cannot separate models 0.01 of AUC apart.

# 3 · Choosing a model

PART 3

When a more flexible model does better, what it costs to fit and to run, and what it can explain.

# Four tables, four answers

| table | rows to learn from | the linear model | the best trees | the change |
|---|---|---|---|---|
| volatility, test RMSE | 1,894 days | ridge 0.218 | boosted trees 0.217 | none |
| credit, test AUC | 21,000 borrowers | logistic regression 0.748 | a forest 0.777 | +0.029 |
| houses, test RMSE | 6,212 sales | ridge 1.26 million | boosted, coordinates 1.03 million | -18 percent |
| hours, test AUC | 17,422 hours | logistic regression 0.833 | a forest 0.857 | +0.024 |

The trees gained most where location or a changing slope matters, and nothing where the target is mostly noise around a straight line.

# Where the trees gained

### A map

The town columns hold one premium per town. Boosted trees cut the coordinates into districts and took 18 percent off the error.

### A slope that changes

A fall in the wind raises tomorrow’s price less often when today’s price is high. A linear model has one slope for it.

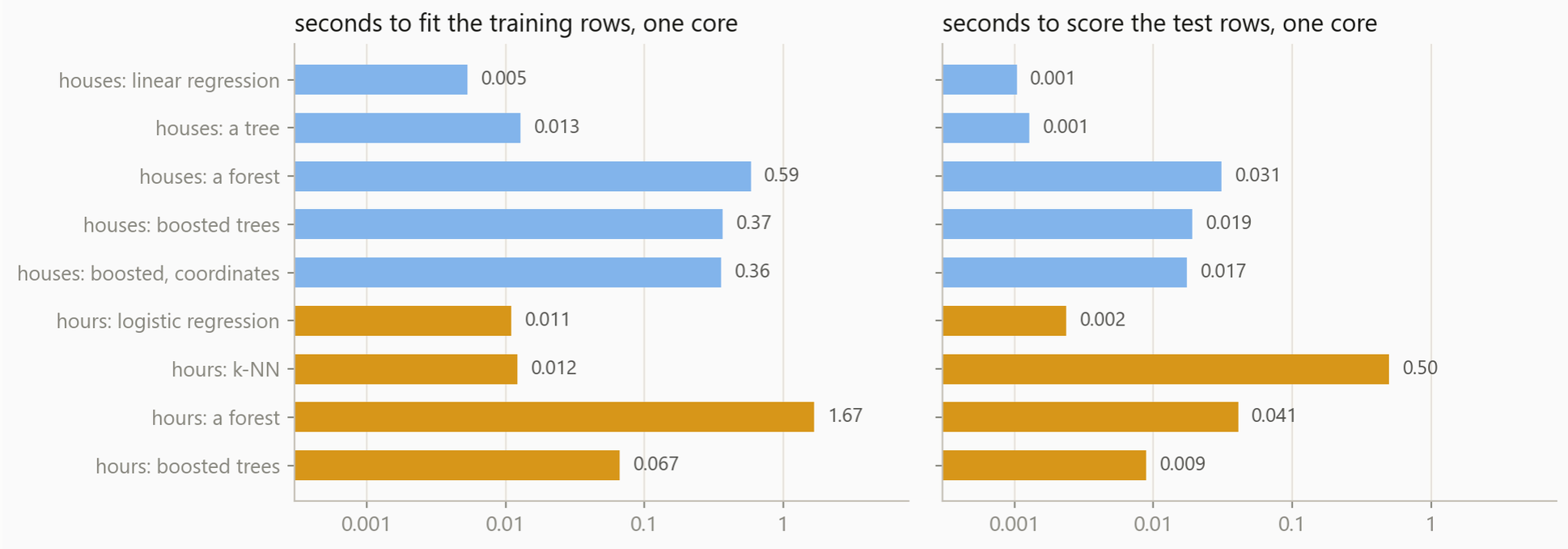
### Steps

On the credit table the default rate jumps from 12 to 30 percent at the first month late. A tree’s cut takes a step like that directly.

### Mostly noise

Next month’s volatility is close to this month’s plus noise. On 1,894 days the trees only matched ridge.

# The time to fit and to predict




k-NN fits at once but takes half a second to score 2024; the forest takes over a second to fit. A grid of settings multiplies every fit.

# Two fits, timed PREDICT

Python Code

```
import time
start = time.perf_counter()
RandomForestClassifier(n_estimators=100, min_samples_leaf=20, random_state=0).fit(power_train[more], power_train['up'])
print('a forest     :', round(time.perf_counter() - start, 2), 'seconds')
start = time.perf_counter()
XGBClassifier(n_estimators=58, learning_rate=0.05, max_depth=4, random_state=0).fit(power_train[more], power_train['up'])
print('boosted trees:', round(time.perf_counter() - start, 2), 'seconds')
```

Before you run it: which of the two takes longer in your browser, and by how much? Both models score about 0.857 on 2024.

# Thirty more square metres

Python Code

```
six = test.sample(6, random_state=3)
bigger = six.copy()
bigger['area'] = bigger['area'] + 30
print(np.exp(valuation.predict(bigger[columns]) - valuation.predict(six[columns])).round(3))
print(np.exp(boosted_place.predict(bigger[place]) - boosted_place.predict(six[place])).round(3))
```

```
[1.147 1.147 1.147 1.147 1.147 1.147]
[1.157 1.08  1.351 1.178 1.205 1.126]
```

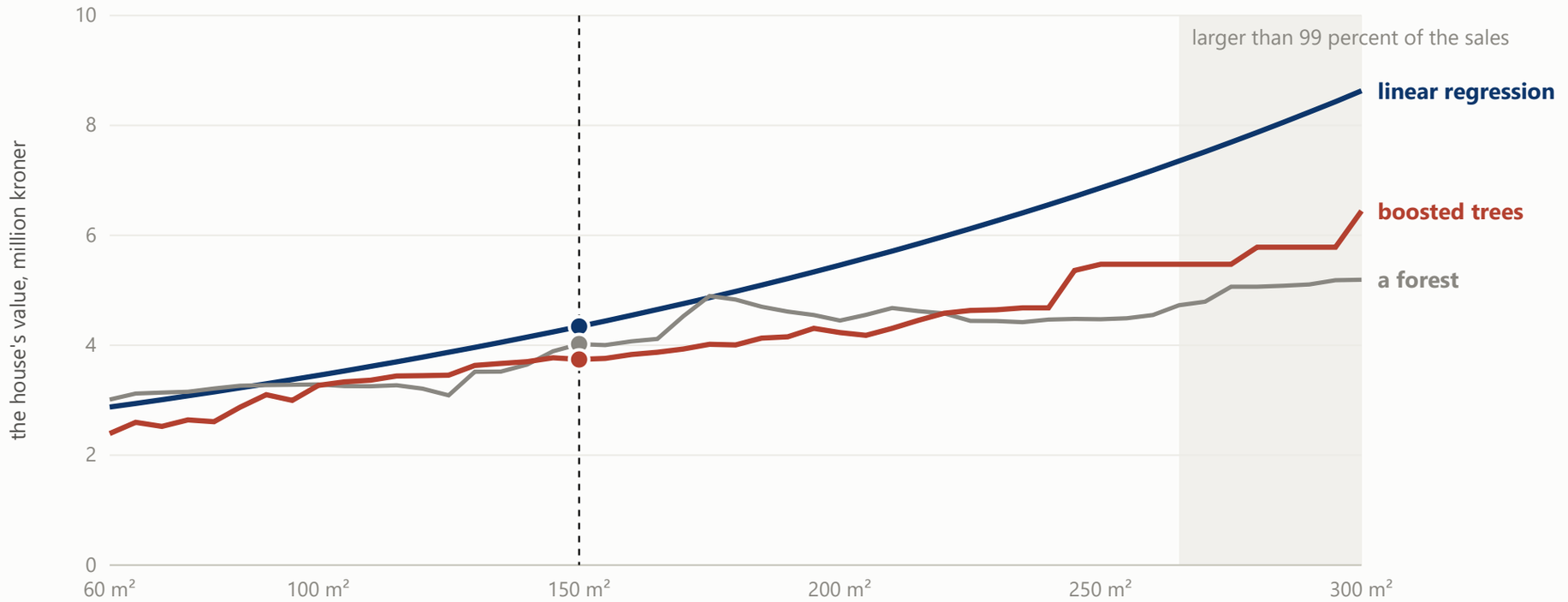
In logs, linear regression adds 14.7 percent to every house. Boosted trees add between 8 and 35 percent, depending on the house and where it stands.

# A bigger house

PLAY

area 150 m²

boosted trees, told to rise




At 150 m², in million kroner: linear regression **4.34** · a forest **4.02** · boosted trees **3.74**

Boosted trees value the house lower at some larger areas, and the forest stops rising beyond the largest sales it learned from.

# Monotone constraints

Python Code

```
stopper = XGBRegressor(n_estimators=3000, learning_rate=0.05, max_depth=4, early_stopping_rounds=100,
                       monotone_constraints={'area': 1}, random_state=0)
stopper.fit(fit_rows[place], np.log(fit_rows['price']),
            eval_set=[(stop_rows[place], np.log(stop_rows['price']))], verbose=False)
rising = XGBRegressor(n_estimators=stopper.best_iteration + 1, learning_rate=0.05, max_depth=4,
                      monotone_constraints={'area': 1}, random_state=0)
rising.fit(train[place], np.log(train['price']))
valued = np.exp(rising.predict(test[place]))
print(stopper.best_iteration + 1, round(rmse(test['price'], valued)), round(underwater(test['price'], valued), 3))
```

```
547 1062937 0.101
```

`monotone_constraints` lets the value only rise with the area. Every one of 200 test houses had a fall before and none has one now, for 33,000 kroner of RMSE.

# Explanations in credit

The EU’s AI Act counts models that score the creditworthiness of people as high-risk. A person refused on such a score may ask for the main elements of the decision.

A coefficient states the same reason for every customer. A sum of a thousand trees has to be explained one decision at a time.

Regulation (EU) 2024/1689 (the AI Act), Annex III, point 5(b), and Article 86(1).

# The trade-offs, model by model

| model | houses: RMSE, and seconds to fit | hours: AUC, and seconds to fit | one prediction comes from | beyond the training rows |
|---|---|---|---|---|
| linear or logistic regression | 1.26 million, 0.005 s | 0.833, 0.011 s | one coefficient per column | the slope continues |
| k-NN | | 0.844, 0.012 s; 0.50 s to predict | the nearest training rows | the nearest training rows |
| a tree | 1.27 million, 0.013 s | | a path of up to 19 questions | the value of a leaf |
| a forest | 1.11 million, 0.59 s | 0.857, 1.67 s | an average of 100 trees | flat |
| boosted trees | 1.03 million, 0.36 s | 0.856, 0.067 s | a sum of 58 to 1,017 trees | flat |

None of them wins every column. Which one matters depends on who uses the model and how often it is refitted.

# Choosing a model

1. Score a baseline and a linear model first.
2. Fit a forest and boosted trees on the same folds.
3. Keep the flexible model only when it wins on the folds and on the test rows.
4. Check what it does to one row: its direction, and far from the data.
5. Price the compute: fitting, tuning and predicting.
6. Ask who has to understand one prediction.

# 4 · The course, and the exam

PART 4

A look back over thirteen sessions, the exam, and time for your questions.

# Thirteen sessions

| | | |
|---|---|---|
| **1 to 3** | Python: variables, loops, functions and dictionaries; NumPy, pandas and matplotlib | the risk report begins |
| **4** | features, targets, splits and the scores of a prediction | no model yet |
| **5 and 6** | linear regression, folds, ridge and lasso | the volatility forecast |
| **7** | a house in Aarhus, with every tool so far | a case lecture |
| **8 and 9** | logistic regression, thresholds, costs and k-NN | the jump warning |
| **10** | tomorrow’s power price in West Denmark | a case lecture |
| **11 and 12** | trees, forests and boosted trees | the report as a whole |
| **13** | **the two cases with trees** | today |

# The routine behind every model

### A baseline first

The training average, the majority class or persistence. Every score goes next to the baseline’s.

### A split that respects time

When the rows are days or hours, the test rows come after the training rows.

### Settings on validation rows

Folds or a later block choose alpha, C, k, the depth and the number of trees.

### The test rows once

They score the model already chosen, and the folds are reported beside them.

### A model read

Its coefficients, its importance, or what it does to one row.

### A model priced

The time to fit and to predict, and who has to understand it.

# The exam

**180 minutes**
on site, offline, in VS Code

**Two parts**
50 points each

**No AI**
and no internet; everything else is allowed

**Two files**
the answer notebook and the blank PDF

Three mock exams and a two-page guide to preparing are on the Exam Info page of the course site.

# Before the exam

- While you are online, save the course ZIP, the lecture PDFs and the cheatsheet from the Exam info page.
- Install the course's packages with the setup page's one line, then run `setup_check.ipynb` from the ZIP with Wi-Fi off: it should say ready.
- Switch off AI completion in VS Code, as the guide shows. The suggestion list that offers `head` after `df.` is allowed and stays on.
- Sit one mock exam in three uninterrupted hours, with Wi-Fi and AI switched off.

# Questions

# Questions about the exam, the course, or anything else

The lectures, the notebooks, the cheatsheet and the mock exams stay on the course site. Email:

**jobo@econ.au.dk**