Lecture 4: Feature importances and feature selection

Contents

Learning outcomes

From this lecture, students are expected to be able to:

- Explain the limitations of simple correlation-based approaches to identify feature importances.
- Interpret the coefficients of linear regression, including for scaled numeric features.
- Explain the purpose of feature selection.
- Discuss and compare different feature selection methods at a high level.
- Explain and use model-based feature selection.
- Explain recursive feature elimination at a high level.
- Use `sklearn`'s implementation of recursive feature elimination (`RFE`).
- Explain how RFECV (cross-validated recursive feature elimination) works.
- Explain forward and backward feature selection at a high level.
- Use `sklearn`'s implementation of forward and backward selection.

Imports and LOs

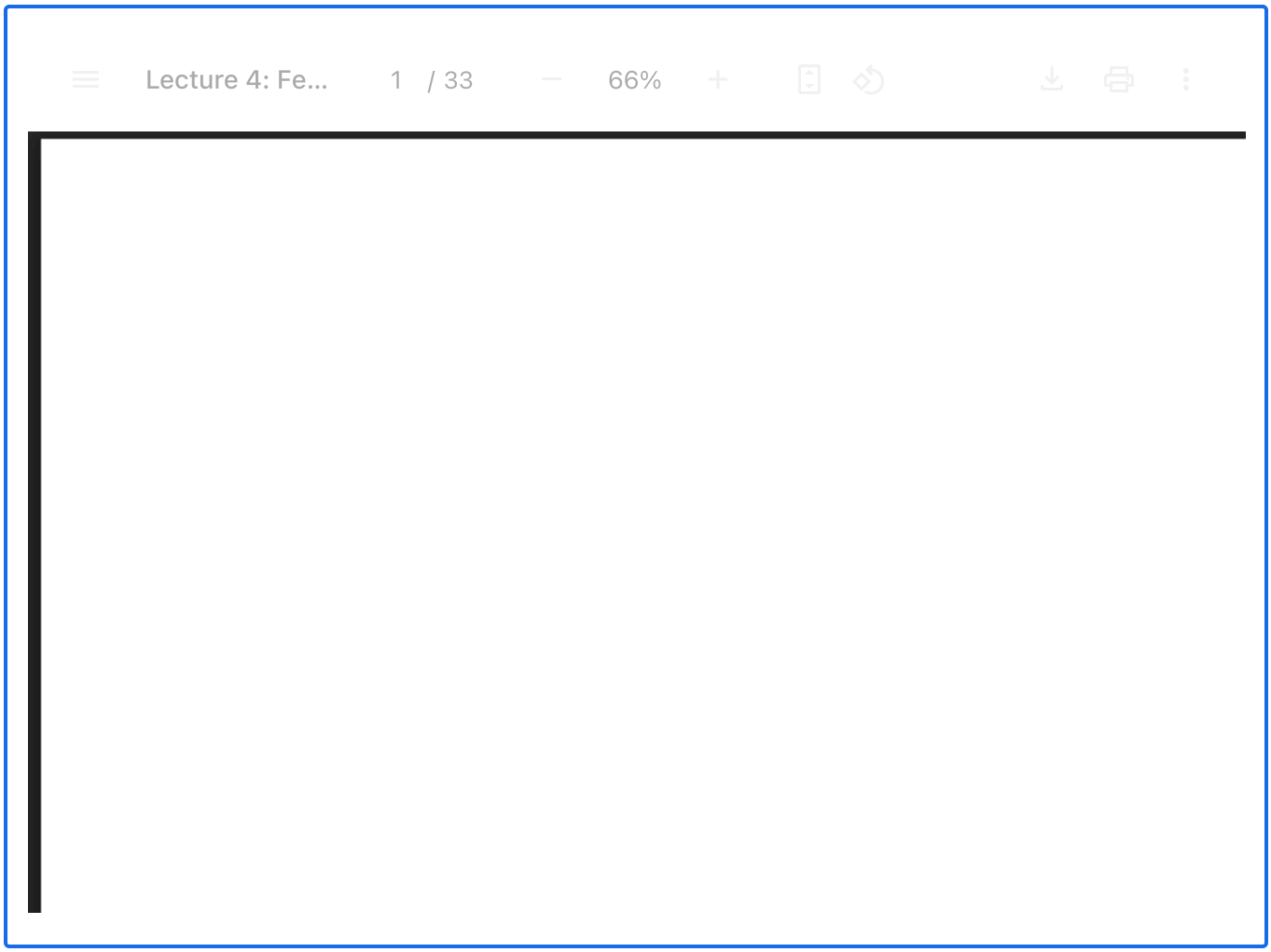
Imports

```
import os
import string
import sys
from collections import deque

import matplotlib.pyplot as plt
import altair_ally as aly
import numpy as np
import pandas as pd

sys.path.append("code/.")

from plotting_functions import *
from sklearn import datasets
from sklearn.compose import ColumnTransformer, make_column_transformer
from sklearn.dummy import DummyRegressor
from sklearn.ensemble import RandomForestRegressor
from sklearn.impute import SimpleImputer
from sklearn.linear_model import Ridge
from sklearn.model_selection import (
    cross_val_score,
    cross_validate,
    train_test_split,
)
from sklearn.pipeline import Pipeline, make_pipeline
from sklearn.preprocessing import OneHotEncoder, OrdinalEncoder, StandardScale
from sklearn.svm import SVR
from utils import *
```

```
import warnings

warnings.simplefilter(action="ignore", category=FutureWarning)
```

Lecture slides

Section 1 slides | Section 2 slides



Data

In the first part of this lecture, we'll be using Kaggle House Prices dataset, the dataset we used previously. As usual, to run this notebook you'll need to download the data. Unzip the data into a subdirectory called `data`. For this dataset, train and test have already been separated. We'll be working with the train portion in this lecture.

```
df = pd.read_csv("data/housing-kaggle-train.csv")
train_df, test_df = train_test_split(df, test_size=0.10, random_state=123)
train_df
```

	Id	MSSubClass	MSZoning	LotFrontage	LotArea	Street	Alley	LotSl
302	303	20	RL	118.0	13704	Pave	NaN	
767	768	50	RL	75.0	12508	Pave	NaN	
429	430	20	RL	130.0	11457	Pave	NaN	
1139	1140	30	RL	98.0	8731	Pave	NaN	
558	559	60	RL	57.0	21872	Pave	NaN	
...	...	...	...	...	...	...	...	
1041	1042	60	RL	NaN	9130	Pave	NaN	
1122	1123	20	RL	NaN	8926	Pave	NaN	
1346	1347	20	RL	NaN	20781	Pave	NaN	
1406	1407	85	RL	70.0	8445	Pave	NaN	
1389	1390	50	RM	60.0	6000	Pave	NaN	

1314 rows × 81 columns

- The prediction task is predicting `SalePrice` given features related to properties.
- Note that the target is numeric, not categorical.

```
train_df.shape
```

```
(1314, 81)
```

Let's separate `X` and `y`

```
X_train = train_df.drop(columns=["SalePrice"])
y_train = train_df["SalePrice"]

X_test = test_df.drop(columns=["SalePrice"])
y_test = test_df["SalePrice"]
```

Let's identify feature types

```
drop_features = ["Id"]
numeric_features = [
    "BedroomAbvGr",
    "KitchenAbvGr",
    "LotFrontage",
    "LotArea",
    "OverallQual",
    "OverallCond",
    "YearBuilt",
    "YearRemodAdd",
    "MasVnrArea",
    "BsmtFinSF1",
    "BsmtFinSF2",
    "BsmtUnfSF",
    "TotalBsmtSF",
    "1stFlrSF",
    "2ndFlrSF",
    "LowQualFinSF",
    "GrLivArea",
    "BsmtFullBath",
    "BsmtHalfBath",
    "FullBath",
    "HalfBath",
    "TotRmsAbvGrd",
    "Fireplaces",
    "GarageYrBlt",
    "GarageCars",
    "GarageArea",
    "WoodDeckSF",
    "OpenPorchSF",
    "EnclosedPorch",
    "3SsnPorch",
    "ScreenPorch",
    "PoolArea",
    "MiscVal",
    "YrSold",
]
```

```
ordinal_features_reg = [
    "ExterQual",
    "ExterCond",
    "BsmtQual",
    "BsmtCond",
    "HeatingQC",
    "KitchenQual",
    "FireplaceQu",
    "GarageQual",
    "GarageCond",
    "PoolQC",
]
ordering = [
    "Po",
    "Fa",
    "TA",
    "Gd",
    "Ex",
]  # if N/A it will just impute something, per below
ordering_ordinal_reg = [ordering] * len(ordinal_features_reg)
ordering_ordinal_reg
```

```
[['Po', 'Fa', 'TA', 'Gd', 'Ex'],
 ['Po', 'Fa', 'TA', 'Gd', 'Ex'],
 ['Po', 'Fa', 'TA', 'Gd', 'Ex'],
 ['Po', 'Fa', 'TA', 'Gd', 'Ex'],
 ['Po', 'Fa', 'TA', 'Gd', 'Ex'],
 ['Po', 'Fa', 'TA', 'Gd', 'Ex'],
 ['Po', 'Fa', 'TA', 'Gd', 'Ex'],
 ['Po', 'Fa', 'TA', 'Gd', 'Ex'],
 ['Po', 'Fa', 'TA', 'Gd', 'Ex'],
 ['Po', 'Fa', 'TA', 'Gd', 'Ex']]
```

```
ordinal_features_oth = [
    "BsmtExposure",
    "BsmtFinType1",
    "BsmtFinType2",
    "Functional",
    "Fence",
]
ordering_ordinal_oth = [
    ["NA", "No", "Mn", "Av", "Gd"],
    ["NA", "Unf", "LwQ", "Rec", "BLQ", "ALQ", "GLQ"],
    ["NA", "Unf", "LwQ", "Rec", "BLQ", "ALQ", "GLQ"],
    ["Sal", "Sev", "Maj2", "Maj1", "Mod", "Min2", "Min1", "Typ"],
    ["NA", "MnWw", "GdWo", "MnPrv", "GdPrv"],
]
```

```
categorical_features = list(
    set(X_train.columns)
    - set(numeric_features)
    - set(ordinal_features_reg)
    - set(ordinal_features_oth)
    - set(drop_features)
)
categorical_features
```

```
['LandSlope',
 'GarageType',
 'PavedDrive',
 'GarageFinish',
 'SaleType',
 'Condition2',
 'Exterior1st',
 'RoofStyle',
 'Street',
 'Condition1',
 'RoofMatl',
 'LotShape',
 'Heating',
 'Neighborhood',
 'MSSubClass',
 'MasVnrType',
 'Utilities',
 'Exterior2nd',
 'MSZoning',
 'LotConfig',
 'LandContour',
 'MoSold',
 'HouseStyle',
 'BldgType',
 'CentralAir',
 'Alley',
 'Electrical',
 'MiscFeature',
 'Foundation',
 'SaleCondition']
```

```
from sklearn.compose import ColumnTransformer, make_column_transformer

numeric_transformer = make_pipeline(SimpleImputer(strategy="median"), Standard
ordinal_transformer_reg = make_pipeline(
    SimpleImputer(strategy="most_frequent"),
    OrdinalEncoder(categories=ordering_ordinal_reg),
)

ordinal_transformer_oth = make_pipeline(
    SimpleImputer(strategy="most_frequent"),
    OrdinalEncoder(categories=ordering_ordinal_oth),
)

categorical_transformer = make_pipeline(
    SimpleImputer(strategy="constant", fill_value="missing"),
    OneHotEncoder(handle_unknown="ignore", sparse_output=False),
)

preprocessor = make_column_transformer(
    ("drop", drop_features),
    (numeric_transformer, numeric_features),
    (ordinal_transformer_reg, ordinal_features_reg),
    (ordinal_transformer_oth, ordinal_features_oth),
    (categorical_transformer, categorical_features),
    verbose_feature_names_out=False
)
```

```
preprocessor.fit(X_train)
preprocessor.named_transformers_
```

```
{'drop': 'drop',
 'pipeline-1': Pipeline(steps=[('simpleimputer', SimpleImputer(strategy='median
                ('standardscaler', StandardScaler())]),
 'pipeline-2': Pipeline(steps=[('simpleimputer', SimpleImputer(strategy='most_
                ('ordinalencoder',
                 OrdinalEncoder(categories=[['Po', 'Fa', 'TA', 'Gd', 'Ex'],
                                            ['Po', 'Fa', 'TA', 'Gd', 'Ex'],
                                            ['Po', 'Fa', 'TA', 'Gd', 'Ex'],
                                            ['Po', 'Fa', 'TA', 'Gd', 'Ex'],
                                            ['Po', 'Fa', 'TA', 'Gd', 'Ex'],
                                            ['Po', 'Fa', 'TA', 'Gd', 'Ex'],
                                            ['Po', 'Fa', 'TA', 'Gd', 'Ex'],
                                            ['Po', 'Fa', 'TA', 'Gd', 'Ex'],
                                            ['Po', 'Fa', 'TA', 'Gd', 'Ex'],
                                            ['Po', 'Fa', 'TA', 'Gd', 'Ex']]))]
 'pipeline-3': Pipeline(steps=[('simpleimputer', SimpleImputer(strategy='most_
                ('ordinalencoder',
                 OrdinalEncoder(categories=[['NA', 'No', 'Mn', 'Av', 'Gd'],
                                            ['NA', 'Unf', 'LwQ', 'Rec', 'BLQ',
                                             'ALQ', 'GLQ'],
                                            ['NA', 'Unf', 'LwQ', 'Rec', 'BLQ',
                                             'ALQ', 'GLQ'],
                                            ['Sal', 'Sev', 'Maj2', 'Maj1',
                                             'Mod', 'Min2', 'Min1', 'Typ'],
                                            ['NA', 'MnWw', 'GdWo', 'MnPrv',
                                             'GdPrv']]))]),
 'pipeline-4': Pipeline(steps=[('simpleimputer',
                 SimpleImputer(fill_value='missing', strategy='constant')),
                ('onehotencoder',
                 OneHotEncoder(handle_unknown='ignore', sparse_output=False))]
```

```
ohe_columns = list(
    preprocessor.named_transformers_["pipeline-4"]
    .named_steps["onehotencoder"]
    .get_feature_names_out(categorical_features)
)
new_columns = (
    numeric_features + ordinal_features_reg + ordinal_features_oth + ohe_colum
)
```

```
X_train_enc = pd.DataFrame(
    preprocessor.transform(X_train), index=X_train.index, columns=new_columns
)
X_train_enc
```

	BedroomAbvGr	KitchenAbvGr	LotFrontage	LotArea	OverallQual	Over
302	0.154795	-0.222647	2.312501	0.381428	0.663680	-0
767	1.372763	-0.222647	0.260890	0.248457	-0.054669	1.
429	0.154795	-0.222647	2.885044	0.131607	-0.054669	-0
1139	0.154795	-0.222647	1.358264	-0.171468	-0.773017	-0
558	0.154795	-0.222647	-0.597924	1.289541	0.663680	-0
...	...	...	...	...	...	
1041	1.372763	-0.222647	-0.025381	-0.127107	-0.054669	2.
1122	0.154795	-0.222647	-0.025381	-0.149788	-1.491366	-2
1346	0.154795	-0.222647	-0.025381	1.168244	0.663680	1.
1406	-1.063173	-0.222647	0.022331	-0.203265	-0.773017	1.
1389	0.154795	-0.222647	-0.454788	-0.475099	-0.054669	0.

1314 rows × 262 columns

```
X_train_enc.shape
```

```
(1314, 262)
```

```
lr_pipe = make_pipeline(preprocessor, Ridge())
scores = cross_validate(lr_pipe, X_train, y_train, return_train_score=True)
pd.DataFrame(scores)
```

	fit_time	score_time	test_score	train_score
0	0.043408	0.014885	0.835749	0.916722
1	0.045034	0.014417	0.810073	0.919198
2	0.044893	0.014556	0.831611	0.912395
3	0.044396	0.014408	0.843992	0.914003
4	0.044608	0.014214	0.548831	0.920462

Feature importances

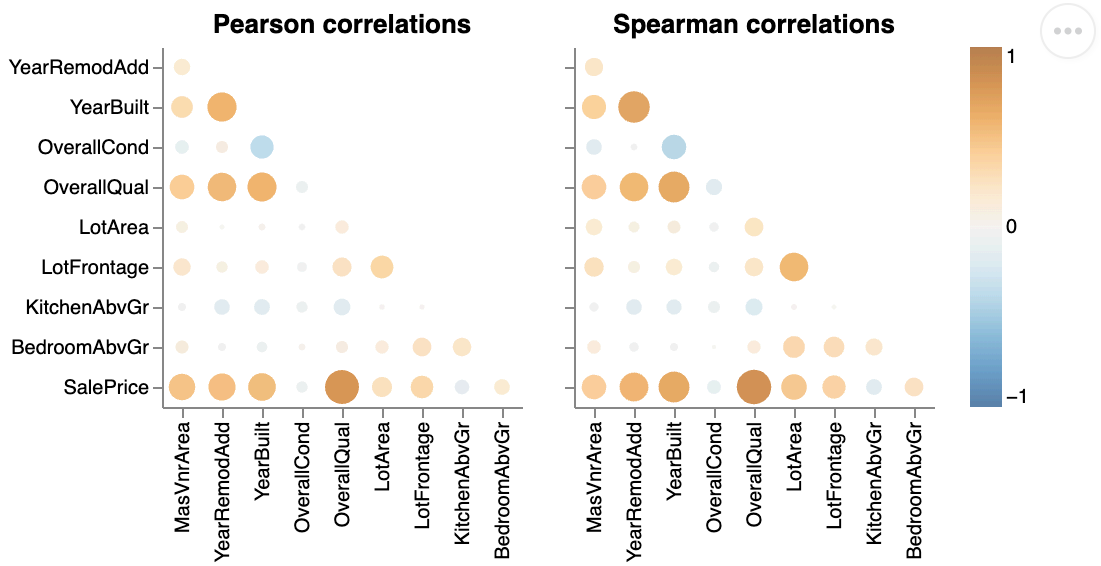
- How does the output depend upon the input?
- How do the predictions change as a function of a particular feature?
- If the model is bad interpretability does not make sense.

SimpleFeature correlations

- Let's look at the correlations between various features with other features and the target in our encoded data (first row/column).
- In simple terms here is how you can interpret correlations between two variables X and Y:
  - If Y goes up when X goes up, we say X and Y are positively correlated.
  - If Y goes down when X goes up, we say X and Y are negatively correlated.
  - If Y is unchanged when X changes, we say X and Y are uncorrelated.

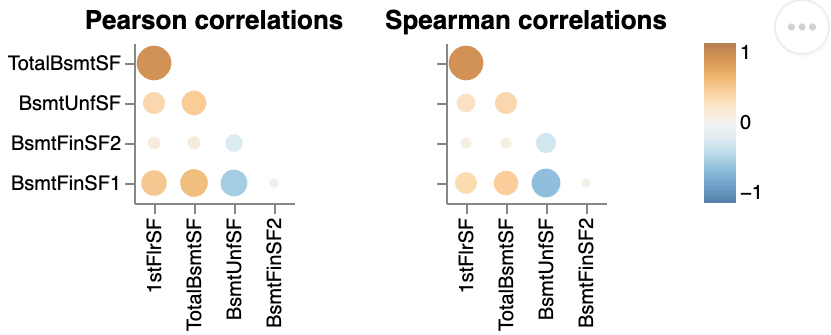
Let's examine the correlations among different columns, including the target column.

```
aly.corr(pd.concat((y_train, X_train_enc), axis=1).iloc[:, :10])
```



- We can immediately see that `SalePrice` is highly correlated with `OverallQual`.
- This is an early hint that `OverallQual` is a useful feature in predicting `SalePrice`.
- However, this approach is **extremely simplistic**.
  - It only looks at each feature in isolation.
  - It only looks at linear associations:
    - What if `SalePrice` is high when `BsmtFullBath` is 2 or 3, but low when it's 0, 1, or 4? They might seem uncorrelated.

```
aly.corr(pd.concat((y_train, X_train_enc), axis=1).iloc[:, 10:15])
```



- Looking at this diagram also tells us the relationship between features.
  - For example, `1stFlrSF` and `TotalBsmtSF` are highly correlated.
  - Do we need both of them?
  - If our model says `1stFlrSF` is very important and `TotalBsmtSF` is very unimportant, do we trust those values?
  - Maybe `TotalBsmtSF` only "becomes important" if `1stFlrSF` is removed.

- Sometimes the opposite happens: a feature only becomes important if another feature is *added*.

Feature importances in linear models

- With linear regression we can look at the *coefficients* for each feature.
- Overall idea: predicted price = intercept + $\sum_i$ coefficient i $\times$ feature i.

```
lr = make_pipeline(preprocessor, Ridge())
lr.fit(X_train, y_train);
```

Let's look at the coefficients.

```
lr_coefs = pd.DataFrame(
    data=lr.named_steps["ridge"].coef_,
    index=new_columns,
    columns=["Coefficient"]
)
lr_coefs.head(20)
```

	Coefficient
BedroomAbvGr	-3717.542624
KitchenAbvGr	-4552.332671
LotFrontage	-1582.710031
LotArea	5118.035161
OverallQual	12498.401830
OverallCond	4854.438906
YearBuilt	4234.888066
YearRemodAdd	317.185155
MasVnrArea	5253.253432
BsmtFinSF1	3681.749118
BsmtFinSF2	581.237935
BsmtUnfSF	-1273.072243
TotalBsmtSF	2759.043319
1stFlrSF	6744.462545
2ndFlrSF	13407.646050
LowQualFinSF	-447.627722
GrLivArea	15992.080694
BsmtFullBath	2305.121599
BsmtHalfBath	500.215865
FullBath	2836.007434

Let's try to interpret coefficients for different types of features.

Ordinal features

- The ordinal features are easiest to interpret.

```
print(ordinal_features_reg)
```

```
['ExterQual', 'ExterCond', 'BsmtQual', 'BsmtCond', 'HeatingQC', 'KitchenQual',
```

```
lr_coefs.loc["ExterQual"]
```

```
Coefficient    4236.969653
Name: ExterQual, dtype: float64
```

- Increasing by one category of exterior quality (e.g. good -> excellent) increases the predicted price by $\sim \$4195$.
  - Wow, that's a lot!
  - Remember this is just what the model has learned. It doesn't tell us how the world works.

```
one_example = X_test[:1]
```

```
one_example[["ExterQual"]]
```

	ExterQual
147	Gd

Let's perturb the example and change `ExterQual` to `Ex`.

```
one_example_perturbed = one_example.copy()
one_example_perturbed["ExterQual"] = "Ex"  # Change Gd to Ex
```

```
one_example_perturbed[["ExterQual"]]
```

	ExterQual
147	Ex

How does the prediction change after changing `ExterQual` from `Gd` to `Ex`?

```

```

```
print("Prediction on the original example: ", lr.predict(one_example))
print("Prediction on the perturbed example: ", lr.predict(one_example_perturbe
print(
    "After changing ExterQual from Gd to Ex increased the prediction by: ",
    lr.predict(one_example_perturbed) - lr.predict(one_example),
)
```

```
Prediction on the original example:  [224865.34161762]
Prediction on the perturbed example:  [229102.31127015]
After changing ExterQual from Gd to Ex increased the prediction by:  [4236.969(
```

That's exactly the learned coefficient for `ExterQual`!

```
lr_coefs.loc["ExterQual"]
```

```
Coefficient    4236.969653
Name: ExterQual, dtype: float64
```

So our interpretation is correct!

- Increasing by one category of exterior quality (e.g. good -> excellent) increases the predicted price by $\sim \$4195$.

Categorical features

- What about the categorical features?
- We have created a number of columns for each category with OHE and each category gets it's own coefficient.

```
print(categorical_features)
```

```
['LandSlope', 'GarageType', 'PavedDrive', 'GarageFinish', 'SaleType', 'Conditi(
```

```
lr_coefs_landslope = lr_coefs[lr_coefs.index.str.startswith("LandSlope")]
lr_coefs_landslope
```

	Coefficient
LandSlope_Gtl	468.638169
LandSlope_Mod	7418.923432
LandSlope_Sev	-7887.561602

- We can talk about switching from one of these categories to another by picking a "reference" category:

```
lr_coefs_landslope - lr_coefs_landslope.loc["LandSlope_Gtl"]
```

	Coefficient
LandSlope_Gtl	0.000000
LandSlope_Mod	6950.285263
LandSlope_Sev	-8356.199771

- If you change the category from `LandSlope_Gtl` to `LandSlope_Mod` the prediction price goes up by $\sim \$6963$
- If you change the category from `LandSlope_Gtl` to `LandSlope_Sev` the prediction price goes down by $\sim \$8334$

Note that this might not make sense in the real world but this is what our model decided to learn given this small amount of data.

```
one_example = X_test[:1]
one_example[['LandSlope']]
```

	LandSlope
147	Gtl

Let's perturb the example and change `LandSlope` to `Mod`.

```
one_example_perturbed = one_example.copy()
one_example_perturbed["LandSlope"] = "Mod"  # Change Gd to Ex
```

```
one_example_perturbed[["LandSlope"]]
```

	LandSlope
147	Mod

How does the prediction change after changing `LandSlope` from `Gtl` to `Mod`?

```
print("Prediction on the original example: ", lr.predict(one_example))
print("Prediction on the perturbed example: ", lr.predict(one_example_perturbe
print(
    "After changing ExterQual from Gd to Ex increased the prediction by: ",
    lr.predict(one_example_perturbed) - lr.predict(one_example),
)
```

```
Prediction on the original example:  [224865.34161762]
Prediction on the perturbed example:  [231815.62688064]
After changing ExterQual from Gd to Ex increased the prediction by:  [6950.285
```

Our interpretation above is correct!

```
lr_coefs.sort_values(by="Coefficient")
```

	Coefficient
RoofMatl_ClyTile	-191169.071745
Condition2_PosN	-105656.864205
Heating_OthW	-27263.223804
MSZoning_C (all)	-22001.877390
Exterior1st_ImStucc	-19422.775311
...	...
PoolQC	34182.041704
RoofMatl_CompShg	36525.193346
Neighborhood_NridgHt	37546.996765
Neighborhood_StoneBr	39931.371722
RoofMatl_WdShngl	83603.013120

262 rows × 1 columns

- For example, the above coefficient says that "If the roof is made of clay or tile, the predicted price is \$191K less"?
- Do we believe these interpretations??
  - Do we believe this is how the predictions are being computed? Yes.
  - Do we believe that this is how the world works? No.

Note

If you did `drop='first'` (we didn't) then you already have a reference class, and all the values are with respect to that one. The interpretation depends on whether we did `drop='first'`, hence the hassle.

Interpreting coefficients of numeric features

Let's look at coefficients of `PoolArea`, `LotFrontage`, `LotArea`.

```
lr_coefs.loc[["PoolArea", "LotFrontage", "LotArea"]]
```

	Coefficient
PoolArea	2817.196385
LotFrontage	-1582.710031
LotArea	5118.035161

Intuition:

- Tricky because numeric features are **scaled**!
- **Increasing** `PoolArea` by 1 scaled unit **increases** the predicted price by $\sim \$2822$.
- **Increasing** `LotArea` by 1 scaled unit **increases** the predicted price by $\sim \$5109$.
- **Increasing** `LotFrontage` by 1 scaled unit **decreases** the predicted price by $\sim \$1578$.

Does that sound reasonable?

- For `PoolArea` and `LotArea`, yes.
- For `LotFrontage`, that's surprising. Something positive would have made more sense?

It might be the case that `LotFrontage` is correlated with some other variable, which might have a larger positive coefficient.

BTW, let's make sure the predictions behave as expected:

Example showing how can we interpret coefficients of scaled features.

- What's one scaled unit for `LotArea`?
- The scaler subtracted the mean and divided by the standard deviation.
- The division actually changed the scale!
- For the unit conversion, we don't care about the subtraction, but only the scaling.

```
scaler = preprocessor.named_transformers_["pipeline-1"]["standardscaler"]
```

```

```

```
# `.scale_` holds the scaling factors for each feature, which is their SD by d
lr_scales = pd.DataFrame(
    data=scaler.scale_,
    index=numeric_features,
    columns=["Scale"]
)
lr_scales.head()
```

	Scale
BedroomAbvGr	0.821040
KitchenAbvGr	0.218760
LotFrontage	20.959139
LotArea	8994.471032
OverallQual	1.392082

- It seems like `LotArea` was divided by 8994 sqft.

```
lr_coefs.loc[["LotArea"]]
```

	Coefficient
LotArea	5118.035161

- The coefficient tells us that if we increase the **scaled** `LotArea` by one scaled unit the price would go up by $\approx \$5118$.
- One scaled unit represents ~ 8994 sqft in the original scale.

Let's examine whether this behaves as expected.

```
X_test_enc = pd.DataFrame(
    preprocessor.transform(X_test),
    index=X_test.index,
    columns=new_columns
)
```

```
one_ex_preprocessed = X_test_enc[:1]
one_ex_preprocessed
```

	BedroomAbvGr	KitchenAbvGr	LotFrontage	LotArea	OverallQual	Overal
147	0.154795	-0.222647	-0.025381	-0.085415	0.66368	-0.5

1 rows × 262 columns

```
orig_pred = lr.named_steps["ridge"].predict(one_ex_preprocessed)
orig_pred
```

```
/home/joel/miniconda3/envs/573/lib/python3.12/site-packages/sklearn/base.py:48
  warnings.warn(
```

```
array([224865.34161762])
```

```
one_ex_preprocessed_perturbed = one_ex_preprocessed.copy()
one_ex_preprocessed_perturbed["LotArea"] += 1  # we are adding one to the scal
one_ex_preprocessed_perturbed
```

	BedroomAbvGr	KitchenAbvGr	LotFrontage	LotArea	OverallQual	Overall
147	0.154795	-0.222647	-0.025381	0.914585	0.66368	-0.51

1 rows × 262 columns

We are expecting an increase of ~$5118 in the prediction compared to the original value of `LotArea`.

```
# The reason we get a warning here is because we are predicting on a datafram
# To avoid it we could use dataframes consistenly throughout
perturbed_pred = lr.named_steps["ridge"].predict(one_ex_preprocessed_perturbed
```

```
/home/joel/miniconda3/envs/573/lib/python3.12/site-packages/sklearn/base.py:48
  warnings.warn(
```

```
perturbed_pred - orig_pred
```

```
array([5118.03516073])
```

Our interpretation is correct!

- Humans find it easier to think about features in their original scale.
- How can we interpret this coefficient in the original scale?
- If I increase original `LotArea` by one square foot then the predicted price would go up by this amount:

```
# Coefficient learned on the scaled features / the scaling factor for this fea
lr_coefs.loc['LotArea'][0] / lr_scales.loc['LotArea'][0]
```

```
0.5690201394129655
```

```
one_example = X_test[:1]
```

```
one_example
```

	Id	MSSubClass	MSZoning	LotFrontage	LotArea	Street	Alley	LotShap
147	148	60	RL	NaN	9505	Pave	NaN	IF

1 rows × 80 columns

Let's perturb the example and add 1 to the `LotArea`.

```
one_example_perturbed = one_example.copy()
one_example_perturbed["LotArea"] += 1
```

if we add 8994.471032 to the original LotArea, the housing price prediction should go up by the coefficient 5109.35671794.

```
one_example_perturbed
```

	Id	MSSubClass	MSZoning	LotFrontage	LotArea	Street	Alley	LotShap
147	148	60	RL	NaN	9506	Pave	NaN	IF

1 rows × 80 columns

Prediction on the original example.

```
lr.predict(one_example)
```

```
array([224865.34161762])
```

Prediction on the perturbed example.

```
lr.predict(one_example_perturbed)
```

```
array([224865.91063776])
```

- What's the difference between predictions?
- Does the difference make sense given the coefficient of the feature?

```
lr.predict(one_example_perturbed) - lr.predict(one_example)
```

```
array([0.56902014])
```

Yes! Our interpretation is correct.

- That said don't read too much into these coefficients without statistical training.

Interim summary

- Correlation among features might make coefficients completely uninterpretable.
- Fairly straightforward to interpret coefficients of ordinal features.
- In categorical features, it's often helpful to consider one category as a reference point and think about relative importance.

- For numeric features, relative importance is meaningful after scaling.
- You have to be careful about the scale of the feature when interpreting the coefficients.
- Remember that explaining the model $\neq$ explaining the data or explaining how the world works.
- The coefficients tell us only about the model and they might not accurately reflect the data.

Feature selection: Introduction and motivation

- With so many ways to add new features, we can increase dimensionality of the data.
- More features means more complex models, which means increasing the chance of overfitting.

What is feature selection?

- Find the features (columns) X that are important for predicting y, and remove the features that aren't.
- Given X and y, find the columns in X that are important for predicting y.

Why feature selection?

- Interpretability: Models are more interpretable with fewer features. If you get the same performance with 10 features instead of 500 features, why not use the model with smaller number of features?
- Computation: Models fit/predict faster with fewer columns.
- Data collection: What type of new data should I collect? It may be cheaper to collect fewer columns.
- Fundamental tradeoff: Can I reduce overfitting by removing useless features?

Feature selection can often result in better performing (less overfit), easier to understand, and faster model.

How do we carry out feature selection?

- There are a number of ways.
- You could use domain knowledge to discard features.
- We are briefly going to look at a few automatic feature selection methods.
  - Model-based selection
  - Recursive feature elimination
  - Forward/Backward selection
- Very related to looking at feature importances.

```
results = {}
```

Let's start with the dummy model.

We'll be using the user-defined function `mean_std_cross_val_scores` to keep track of the results. You can find the code of the function in `code/utils.py`.

```
results["dummy"] = mean_std_cross_val_scores(
    DummyRegressor(), X_train, y_train, return_train_score=True
)
```

Let's try `Ridge` without any feature selection.

```
pipe_lr = make_pipeline(preprocessor, Ridge())

results["ridge"] = mean_std_cross_val_scores(
    pipe_lr, X_train, y_train, return_train_score=True
)
```

```
pd.DataFrame(results).T
```

	fit_time	score_time	test_score	train_score
dummy	0.003 (+/- 0.000)	0.001 (+/- 0.000)	-0.005 (+/- 0.006)	0.000 (+/- 0.000)
ridge	0.044 (+/- 0.003)	0.015 (+/- 0.001)	0.774 (+/- 0.127)	0.917 (+/- 0.003)

Now let's try `RandomForestRegressor`.

We have not talked about this model yet. At this point it's enough to know that it's a tree-based model which is a good off-the-shelf model. We'll be talking about it in Week 4.

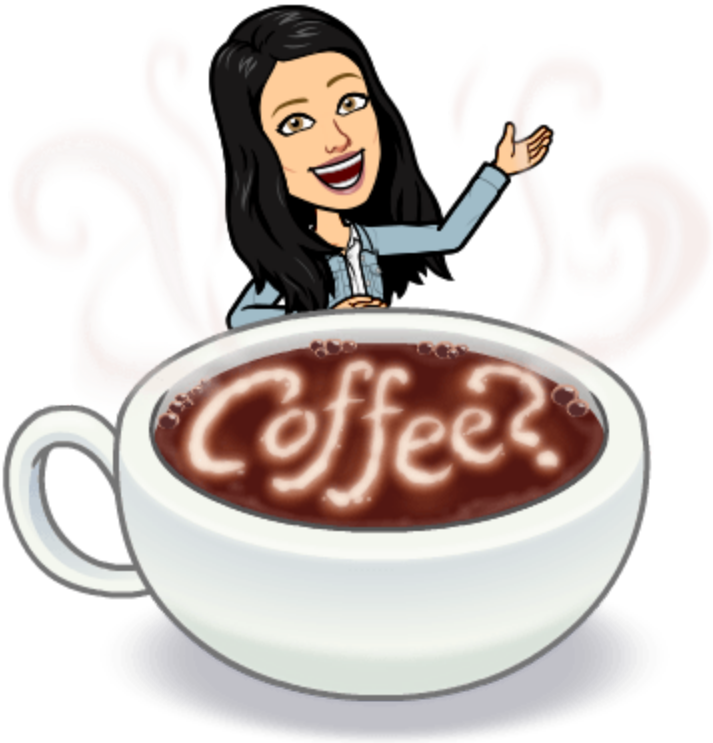
```
pipe_rf = make_pipeline(preprocessor, RandomForestRegressor())

results["rf"] = mean_std_cross_val_scores(
    pipe_rf, X_train, y_train, return_train_score=True
)
```

```
pd.DataFrame(results).T
```

	fit_time	score_time	test_score	train_score
dummy	0.003 (+/- 0.000)	0.001 (+/- 0.000)	-0.005 (+/- 0.006)	0.000 (+/- 0.000)
ridge	0.044 (+/- 0.003)	0.015 (+/- 0.001)	0.774 (+/- 0.127)	0.917 (+/- 0.003)
rf	2.456 (+/- 0.189)	0.020 (+/- 0.002)	0.858 (+/- 0.039)	0.979 (+/- 0.001)

Now we'll try a number of feature selection methods and compare the results with the scores above.

Break (5 min)



Feature selection using feature importances

Model-based feature selection

- Use a supervised machine learning model to judge the importance of each feature.
- Keep only the most important once.
- Supervised machine learning model used for feature selection can be different that the one used as the final estimator.
- Use a model which has some way to calculate feature importances.

Let's examine the least important features according to the model (features associated with coefficients with small magnitude).

```
pipe_lr.fit(X_train, y_train);
```

```
data = {
    "coefficient": pipe_lr.named_steps["ridge"].coef_.tolist(),
    "magnitude": np.absolute(pipe_lr.named_steps["ridge"].coef_),
}
coef_df = pd.DataFrame(data, index=new_columns).sort_values("magnitude")
coef_df
```

	coefficient	magnitude
BsmtFinType2	-8.235387	8.235387
SaleCondition_Alloca	-14.618810	14.618810
GarageArea	101.756286	101.756286
HouseStyle_SFoyer	106.185037	106.185037
MiscFeature_missing	-138.411353	138.411353
...	...	...
Neighborhood_NridgHt	37546.996765	37546.996765
Neighborhood_StoneBr	39931.371722	39931.371722
RoofMatl_WdShngl	83603.013120	83603.013120
Condition2_PosN	-105656.864205	105656.864205
RoofMatl_ClyTile	-191169.071745	191169.071745

262 rows × 2 columns

- Seems like there are a number of features with tiny coefficients compared to other features.
- These features are not going to have a big impact on the prediction.

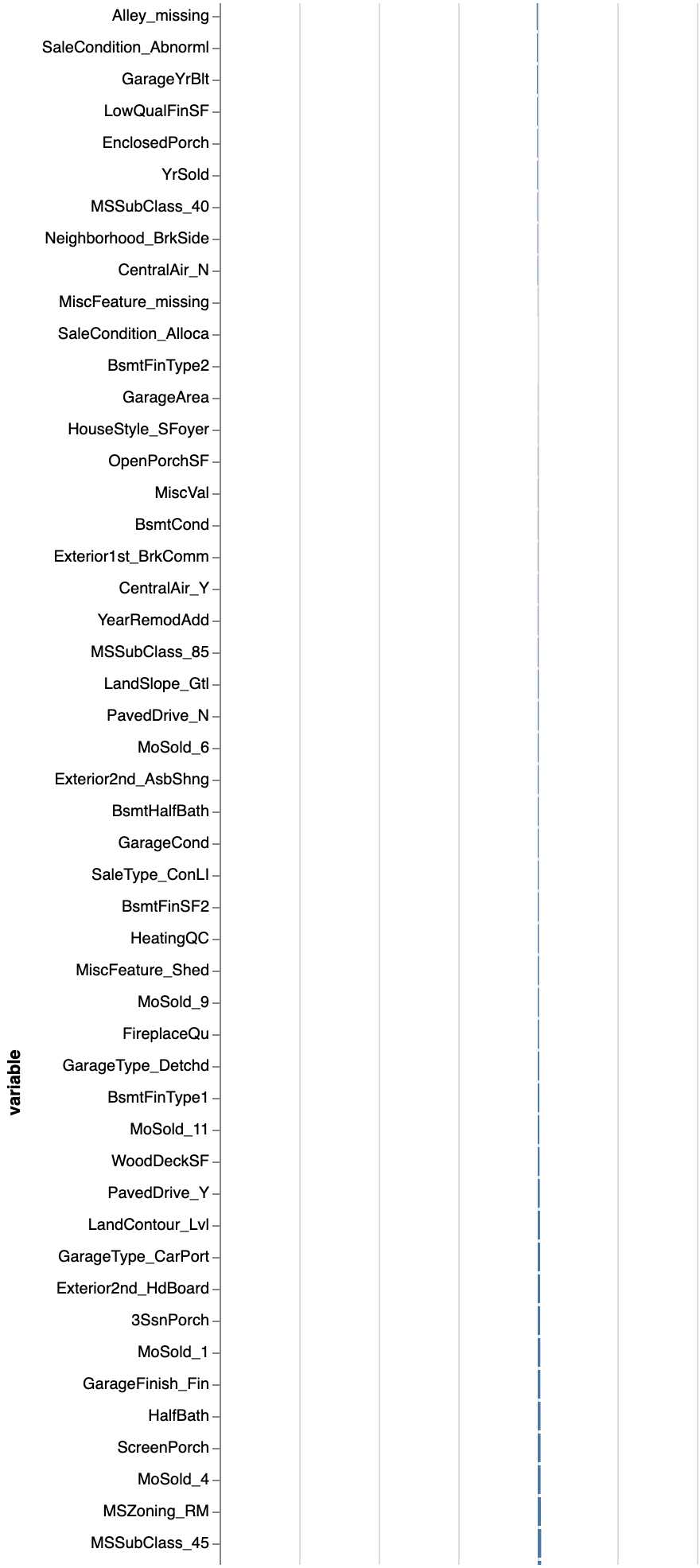
```
coefs = (
    pd.DataFrame(pipe_lr.named_steps["ridge"].coef_, new_columns, columns=["co
    .query("coef != 0")
    .sort_values("coef")
    .reset_index()
    .rename(columns={"index": "variable"})
)
# coefs.style.background_gradient('PuOr')
```

Let's examine the coefficients.

```
import altair as alt

alt.Chart(
    coefs,
    title=pipe_lr.named_steps["ridge"].__str__()[:-2] + " Coefficients",
).mark_bar().encode(y=alt.Y("variable", sort="x"), x="coef")
```

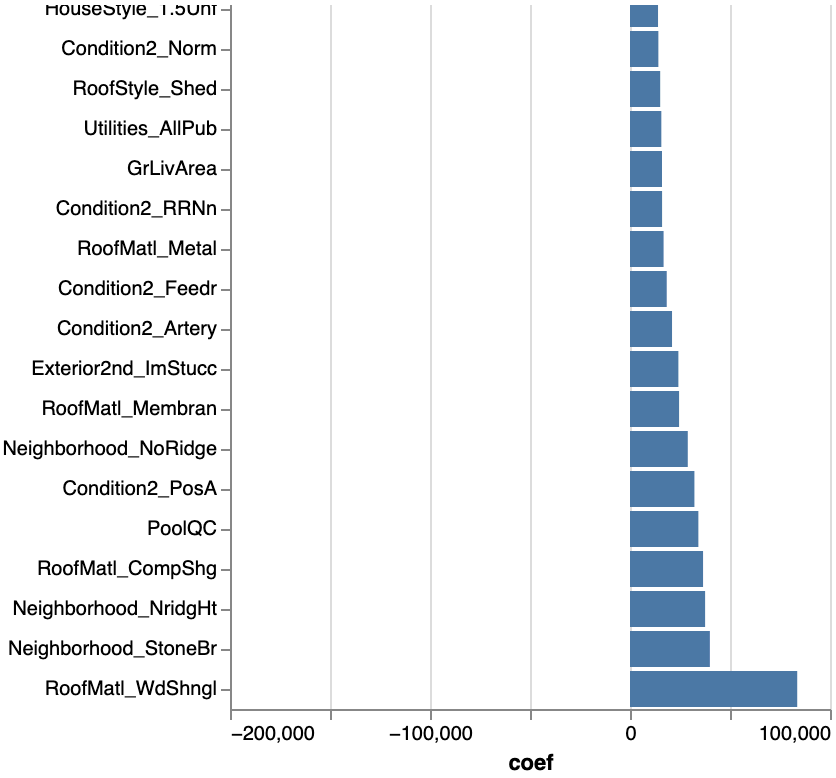

Ridge Coefficients
RoofMatl_ClyTile
Condition2_PosN
Heating_OthW
MSZoning_C (all)
Exterior1st_ImStucc
LotShape_IR3
Condition1_RRAe
Neighborhood_Edwards
Utilities_NoSeWa
HouseStyle_2.5Fin
Foundation_Wood
Neighborhood_Gilbert
SaleType_COD
Neighborhood_Mitchel
LotConfig_FR3
Exterior2nd_Stucco
GarageType_2Types
MSSubClass_120
Neighborhood_NWAmes
Street_Grvl
BldgType_Twnhs
MSSubClass_160
Neighborhood_Timber
SaleType_WD
RoofStyle_Gable
Neighborhood_NAmes
Neighborhood_CollgCr
HouseStyle_2Story
SaleType_ConLw
MSSubClass_180
LandContour_Bnk
RoofStyle_Gambrel
Neighborhood_ClearCr
Condition2_RRAe
Exterior2nd_Other
MSSubClass_75
Neighborhood_MeadowV
Neighborhood_OldTown
MiscFeature_Gar2
HouseStyle_1.5Fin
LandSlope_Sev
Exterior1st_AsphShn
Exterior2nd_Wd Shng
BldgType_TwnhsE
RoofStyle_Hip
MSSubClass_190
MoSold_10
Exterior2nd_Stone


Electrical_SBrkr
Neighborhood_SWISU
SaleCondition_Family
SaleCondition_Partial
Neighborhood_IDOTRR
MasVnrType_BrkCmn
Neighborhood_Sawyer
Neighborhood_SawyerW
GarageFinish_RFn
LotConfig_FR2
KitchenAbvGr
SaleType_Oth
BedroomAbvGr
Electrical_FuseF
ExterCond
Exterior2nd_MetalSd
Exterior1st_HdBoard
MoSold_8
MoSold_2
Condition1_RRNe
Condition1_Feedr
LandContour_Low
GarageType_BuiltIn
Exterior2nd_AsphShn
Electrical_FuseA
Neighborhood_Blmngtn
MasVnrType_BrkFace
Exterior1st_Wd Sdng
Exterior1st_VinylSd
Condition1_PosN
Fence
PavedDrive_P
GarageFinish_Unf
MoSold_12
Condition1_Artery
Electrical_missing
LotFrontage
Alley_Grvl
MiscFeature_TenC
Foundation_BrkTil
Exterior1st_Plywood
Exterior2nd_BrkFace
GarageType_Attchd
BsmtUnfSF
Exterior1st_Stucco
Exterior2nd_Plywood
RoofStyle_Mansard
Neighborhood_Blueste
Exterior2nd_Brk Cmn


variable
Alley_missing
SaleCondition_Abnorml
GarageYrBlt
LowQualFinSF
EnclosedPorch
YrSold
MSSubClass_40
Neighborhood_BrkSide
CentralAir_N
MiscFeature_missing
SaleCondition_Alloca
BsmtFinType2
GarageArea
HouseStyle_SFoyer
OpenPorchSF
MiscVal
BsmtCond
Exterior1st_BrkComm
CentralAir_Y
YearRemodAdd
MSSubClass_85
LandSlope_Gtl
PavedDrive_N
MoSold_6
Exterior2nd_AsbShng
BsmtHalfBath
GarageCond
SaleType_ConLI
BsmtFinSF2
HeatingQC
MiscFeature_Shed
MoSold_9
FireplaceQu
GarageType_Detchd
BsmtFinType1
MoSold_11
WoodDeckSF
PavedDrive_Y
LandContour_Lvl
GarageType_CarPort
Exterior2nd_HdBoard
3SsnPorch
MoSold_1
GarageFinish_Fin
HalfBath
ScreenPorch
MoSold_4
MSZoning_RM
MSSubClass_45


Alley_Pave
Exterior1st_WdShing
Heating_Floor
Exterior2nd_Wd Sdng
Fireplaces
BsmtFullBath
HouseStyle_SLvl
Exterior1st_Stone
MoSold_3
TotalBsmtSF
PoolArea
FullBath
LotConfig_Inside
LotConfig_Corner
MasVnrType_Stone
MoSold_7
Foundation_CBlock
Exterior2nd_CBlock
Exterior1st_CBlock
LotShape_IR1
Neighborhood_Somerst
Exterior1st_CemntBd
MoSold_5
BsmtFinSF1
Foundation_PConc
BldgType_1Fam
Condition1_RRNn
Exterior2nd_VinylSd
MasVnrType_missing
MSSubClass_80
YearBuilt
ExterQual
Electrical_Mix
Foundation_Stone
TotRmsAbvGrd
BldgType_Duplex
MSSubClass_90
GarageQual
HouseStyle_2.5Unf
SaleCondition_Normal
Heating_GasW
Functional
OverallCond
GarageFinish_missing
GarageType_missing
LotArea
Condition1_PosA
Exterior1st_AsbShng
LotShape_Reg


Heating_Grav
MasVnrArea
Foundation_Slab
BsmtExposure
Neighborhood_Veenker
MSSubClass_20
MSZoning_RL
Heating_GasA
MSSubClass_60
MSSubClass_30
KitchenQual
1stFlrSF
MSZoning_RH
BsmtQual
MSZoning_FV
Exterior1st_MetalSd
GarageCars
LandSlope_Mod
RoofMatl_Tar&Grv
SaleCondition_AdjLand
SaleType_CWD
Neighborhood_Crawfor
MSSubClass_70
SaleType_Con
Heating_Wall
RoofMatl_Roll
Condition1_RRAn
Neighborhood_BrDale
MiscFeature_Othr
Electrical_FuseP
Condition1_Norm
GarageType_Basment
Exterior2nd_CmentBd
LotShape_IR2
BldgType_2fmCon
Neighborhood_NPkVill
LandContour_HLS
SaleType_ConLD
MSSubClass_50
Street_Pave
SaleType_New
Exterior1st_BrkFace
OverallQual
HouseStyle_1Story
LotConfig_CulDSac
Condition2_RRAn
RoofMatl_WdShake
2ndFlrSF
RoofStyle_Flat



- How about getting rid of some of these features?
  - That's the idea of Model-based feature selection.
- How do we use model-based selection?
  - Decide a threshold.
  - Discard all features whose feature importances / coefficients are smaller than the threshold.

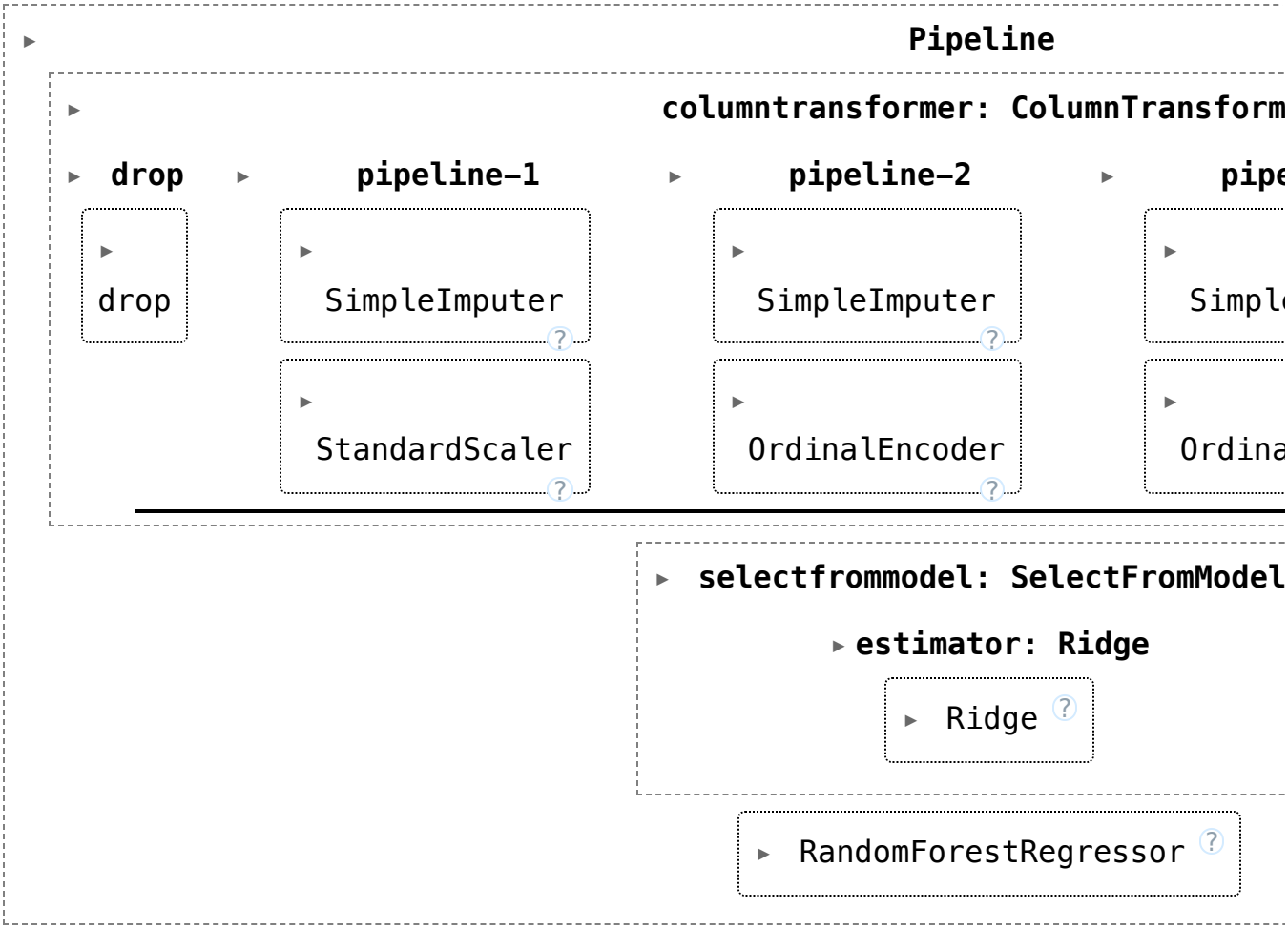
- To use model-based selection, we use `SelectFromModel` transformer.
- It selects features which have the feature importances greater than the provided threshold.
- Below I'm using `Ridge` for feature selection with threshold "median" of feature importances.
- Approximately how many features will be selected?

```
from sklearn.feature_selection import SelectFromModel

select_lr = SelectFromModel(Ridge(), threshold="median")
```

- We can put the feature selection transformer in a pipeline.
- Should we put it before preprocessing or after preprocessing?

```
pipe_rf_model_based = make_pipeline(
    # Feature selection with Ridge and final estimator is RandomForestRegresso
    preprocessor, select_lr, RandomForestRegressor(random_state=42)
)
```

```
pipe_rf_model_based.fit(X_train, y_train)
```



How many features were passed to feature selection?

```
pipe_rf_model_based.named_steps["selectfrommodel"].n_features_in_
```

```
262
```

What's the threshold used for feature selection?

```
pipe_rf_model_based.named_steps["selectfrommodel"].threshold_
```

```
4843.526785655846
```

How many features were selected? That's the same as the number of features passed to the final model, which we can find out using the `n_features_in_` attribute like above.

```
# Number of selected features
pipe_rf_model_based['randomforestregressor'].n_features_in_
```

```
131
```

About half of the features are selected because our threshold is the median.

Which were the selected features are selected? We can find out by using the `get_feature_names_out()` method of the pipeline. In this case, want to include the entire pipeline except the model (so both preprocessing and selection step since they both impact which features the model will see). For this purpose, we can index the pipeline like a list to include all steps except the last one (which is the model) via `pipeline[:-1]`.

```
# Names of selected features
pipe_rf_model_based[:-1].get_feature_names_out()
```

```
array(['LotArea', 'OverallQual', 'OverallCond', 'MasVnrArea', '1stFlrSF',
       '2ndFlrSF', 'GrLivArea', 'GarageCars', 'BsmtQual', 'KitchenQual',
       'PoolQC', 'BsmtExposure', 'Functional', 'LandSlope_Mod',
       'LandSlope_Sev', 'GarageType_2Types', 'GarageType_Basment',
       'GarageType_missing', 'GarageFinish_missing', 'SaleType_COD',
       'SaleType_CWD', 'SaleType_Con', 'SaleType_ConLD', 'SaleType_ConLw',
       'SaleType_New', 'SaleType_WD', 'Condition2_Artery',
       'Condition2_Feedr', 'Condition2_Norm', 'Condition2_PosA',
       'Condition2_PosN', 'Condition2_RRAe', 'Condition2_RRAn',
       'Condition2_RRNn', 'Exterior1st_AsbShng', 'Exterior1st_AsphShn',
       'Exterior1st_BrkFace', 'Exterior1st_ImStucc',
       'Exterior1st_MetalSd', 'RoofStyle_Flat', 'RoofStyle_Gable',
       'RoofStyle_Gambrel', 'RoofStyle_Hip', 'RoofStyle_Shed',
       'Street_Grvl', 'Street_Pave', 'Condition1_Norm', 'Condition1_PosA',
       'Condition1_RRAe', 'Condition1_RRAn', 'RoofMatl_ClyTile',
       'RoofMatl_CompShg', 'RoofMatl_Membran', 'RoofMatl_Metal',
       'RoofMatl_Roll', 'RoofMatl_Tar&Grv', 'RoofMatl_WdShake',
       'RoofMatl_WdShngl', 'LotShape_IR2', 'LotShape_IR3', 'LotShape_Reg',
       'Heating_GasA', 'Heating_Grav', 'Heating_OthW', 'Heating_Wall',
       'Neighborhood_BrDale', 'Neighborhood_ClearCr',
       'Neighborhood_CollgCr', 'Neighborhood_Crawfor',
       'Neighborhood_Edwards', 'Neighborhood_Gilbert',
       'Neighborhood_IDOTRR', 'Neighborhood_MeadowV',
       'Neighborhood_Mitchel', 'Neighborhood_NAmes',
       'Neighborhood_NPkVill', 'Neighborhood_NWAmes',
       'Neighborhood_NoRidge', 'Neighborhood_NridgHt',
       'Neighborhood_OldTown', 'Neighborhood_SWISU',
       'Neighborhood_Sawyer', 'Neighborhood_SawyerW',
       'Neighborhood_StoneBr', 'Neighborhood_Timber',
       'Neighborhood_Veenker', 'MSSubClass_20', 'MSSubClass_30',
       'MSSubClass_50', 'MSSubClass_60', 'MSSubClass_70', 'MSSubClass_75',
       'MSSubClass_120', 'MSSubClass_160', 'MSSubClass_180',
       'MSSubClass_190', 'MasVnrType_BrkCmn', 'Utilities_AllPub',
       'Utilities_NoSeWa', 'Exterior2nd_CmentBd', 'Exterior2nd_ImStucc',
       'Exterior2nd_Other', 'Exterior2nd_Stone', 'Exterior2nd_Stucco',
       'Exterior2nd_Wd Shng', 'MSZoning_C (all)', 'MSZoning_FV',
       'MSZoning_RH', 'MSZoning_RL', 'LotConfig_CulDSac', 'LotConfig_FR3',
       'LandContour_Bnk', 'LandContour_HLS', 'MoSold_10',
       'HouseStyle_1.5Fin', 'HouseStyle_1.5Unf', 'HouseStyle_1Story',
       'HouseStyle_2.5Fin', 'HouseStyle_2Story', 'BldgType_2fmCon',
       'BldgType_Twnhs', 'BldgType_TwnhsE', 'Electrical_FuseP',
       'Electrical_SBrkr', 'MiscFeature_Gar2', 'MiscFeature_Othr',
       'Foundation_Slab', 'Foundation_Wood', 'SaleCondition_AdjLand',
       'SaleCondition_Family', 'SaleCondition_Partial'], dtype=object)
```

If we compare this list to the visualization of the coefficients above, we can see that it is indeed the coefficients with low magnitude that are not selected.

Taking a look at the results using only half of the features, we can see that the model performs about as well as the model with all 263 features, but it is noticeably simpler to interpret since it has fewer features!

```
results["model_based_fs + rf"] = mean_std_cross_val_scores(
    pipe_rf_model_based, X_train, y_train, return_train_score=True
)
pd.DataFrame(results).T
```

	fit_time	score_time	test_score	train_score
dummy	0.003 (+/- 0.000)	0.001 (+/- 0.000)	-0.005 (+/- 0.006)	0.000 (+/- 0.000)
ridge	0.044 (+/- 0.003)	0.015 (+/- 0.001)	0.774 (+/- 0.127)	0.917 (+/- 0.003)
rf	2.456 (+/- 0.189)	0.020 (+/- 0.002)	0.858 (+/- 0.039)	0.979 (+/- 0.001)
model_based_fs + rf	1.006 (+/- 0.140)	0.019 (+/- 0.001)	0.846 (+/- 0.037)	0.978 (+/- 0.002)

Recursive feature elimination (RFE)

- Similar to model-based selection,
- Based on feature importances
- But it's different in the sense that it **iteratively** eliminates unimportant features.
- It builds a series of models. At each iteration, discards the least important feature according to the model.
- Computationally expensive

RFE algorithm

1. Decide the number of features to select.
2. **Assign** importances to features, e.g. by fitting a model and looking at `coef_` or `feature_importances_`.
3. **Remove** the least important feature.
4. **Repeat** steps 2-3 until only the desired number of features is remaining.

Note that this is **not** the same as just removing all the less important features in one shot!

How do we use it?

```
from sklearn.feature_selection import RFE

rfe = RFE(Ridge(), n_features_to_select=120)

pipe_rf_rfe = make_pipeline(preprocessor, rfe, RandomForestRegressor(random_st

results["rfe + rfe"] = mean_std_cross_val_scores(
    pipe_rf_rfe, X_train, y_train, return_train_score=True
)
```

```
pd.DataFrame(results).T
```

	fit_time	score_time	test_score	train_score
dummy	0.003 (+/- 0.000)	0.001 (+/- 0.000)	-0.005 (+/- 0.006)	0.000 (+/- 0.000)
ridge	0.044 (+/- 0.003)	0.015 (+/- 0.001)	0.774 (+/- 0.127)	0.917 (+/- 0.003)
rf	2.456 (+/- 0.189)	0.020 (+/- 0.002)	0.858 (+/- 0.039)	0.979 (+/- 0.001)
model_based_fs + rf	1.006 (+/- 0.140)	0.019 (+/- 0.001)	0.846 (+/- 0.037)	0.978 (+/- 0.002)
rfe + rfe	1.589 (+/- 0.143)	0.020 (+/- 0.002)	0.862 (+/- 0.034)	0.980 (+/- 0.002)

Slightly better results with less features!

```
pipe_rf_rfe.fit(X_train, y_train);
```

How many features are selected?

```
pipe_rf_rfe.named_steps["rfe"].n_features_
```

```
120
```

What features are selected?

```
pipe_rf_rfe[:-1].get_feature_names_out()
```

```
array(['LotArea', 'OverallQual', 'OverallCond', 'YearBuilt', 'MasVnrArea',
       'BsmtUnfSF', 'TotalBsmtSF', '1stFlrSF', '2ndFlrSF', 'GrLivArea',
       'GarageCars', 'BsmtQual', 'KitchenQual', 'PoolQC', 'BsmtExposure',
       'LandSlope_Mod', 'LandSlope_Sev', 'GarageType_2Types',
       'GarageType_missing', 'GarageFinish_Fin', 'GarageFinish_missing',
       'SaleType_COD', 'SaleType_CWD', 'SaleType_Con', 'SaleType_ConLD',
       'SaleType_New', 'Condition2_Artery', 'Condition2_Feedr',
       'Condition2_Norm', 'Condition2_PosA', 'Condition2_PosN',
       'Condition2_RRAe', 'Condition2_RRAn', 'Condition2_RRNn',
       'Exterior1st_AsphShn', 'Exterior1st_BrkFace',
       'Exterior1st_ImStucc', 'RoofStyle_Flat', 'RoofStyle_Gable',
       'RoofStyle_Gambrel', 'RoofStyle_Hip', 'RoofStyle_Shed',
       'Street_Grvl', 'Street_Pave', 'Condition1_Norm', 'Condition1_RRAe',
       'Condition1_RRAn', 'Condition1_RRNn', 'RoofMatl_ClyTile',
       'RoofMatl_CompShg', 'RoofMatl_Membran', 'RoofMatl_Metal',
       'RoofMatl_Roll', 'RoofMatl_Tar&Grv', 'RoofMatl_WdShake',
       'RoofMatl_WdShngl', 'LotShape_IR2', 'LotShape_IR3', 'Heating_OthW',
       'Heating_Wall', 'Neighborhood_BrDale', 'Neighborhood_ClearCr',
       'Neighborhood_CollgCr', 'Neighborhood_Crawfor',
       'Neighborhood_Edwards', 'Neighborhood_Gilbert',
       'Neighborhood_IDOTRR', 'Neighborhood_MeadowV',
       'Neighborhood_Mitchel', 'Neighborhood_NAmes',
       'Neighborhood_NPkVill', 'Neighborhood_NWAmes',
       'Neighborhood_NoRidge', 'Neighborhood_NridgHt',
       'Neighborhood_OldTown', 'Neighborhood_Sawyer',
       'Neighborhood_SawyerW', 'Neighborhood_StoneBr',
       'Neighborhood_Timber', 'MSSubClass_20', 'MSSubClass_30',
       'MSSubClass_50', 'MSSubClass_60', 'MSSubClass_70', 'MSSubClass_80',
       'MSSubClass_120', 'MSSubClass_160', 'MSSubClass_180',
       'MSSubClass_190', 'MasVnrType_missing', 'Utilities_AllPub',
       'Utilities_NoSeWa', 'Exterior2nd_CmentBd', 'Exterior2nd_ImStucc',
       'Exterior2nd_Other', 'Exterior2nd_Stone', 'Exterior2nd_Stucco',
       'Exterior2nd_Wd Shng', 'MSZoning_C (all)', 'MSZoning_FV',
       'LotConfig_CulDSac', 'LotConfig_FR2', 'LotConfig_FR3',
       'LandContour_Bnk', 'LandContour_HLS', 'LandContour_Low',
       'MoSold_10', 'HouseStyle_1.5Fin', 'HouseStyle_1.5Unf',
       'HouseStyle_1Story', 'HouseStyle_2.5Fin', 'HouseStyle_2Story',
       'BldgType_2fmCon', 'BldgType_Twnhs', 'BldgType_TwnhsE',
       'Electrical_FuseP', 'MiscFeature_Gar2', 'Foundation_Wood',
       'SaleCondition_AdjLand', 'SaleCondition_Family'], dtype=object)
```

- Let's compare the selected features with features selected with model-based selection.

```
set(pipe_rf_model_based[:-1].get_feature_names_out()) - set(pipe_rf_rfe[:-1].g
```

```
{'Condition1_PosA',
 'Electrical_SBrkr',
 'Exterior1st_AsbShng',
 'Exterior1st_MetalSd',
 'Foundation_Slab',
 'Functional',
 'GarageType_Basment',
 'Heating_GasA',
 'Heating_Grav',
 'LotShape_Reg',
 'MSSubClass_75',
 'MSZoning_RH',
 'MSZoning_RL',
 'MasVnrType_BrkCmn',
 'MiscFeature_Othr',
 'Neighborhood_SWISU',
 'Neighborhood_Veenker',
 'SaleCondition_Partial',
 'SaleType_ConLw',
 'SaleType_WD'}
```

```
set(pipe_rf_rfe[:-1].get_feature_names_out()) - set(pipe_rf_model_based[:-1].g
```

```
{'BsmtUnfSF',
 'Condition1_RRNn',
 'GarageFinish_Fin',
 'LandContour_Low',
 'LotConfig_FR2',
 'MSSubClass_80',
 'MasVnrType_missing',
 'TotalBsmtSF',
 'YearBuilt'}
```

It seems like most of the over 100 selected features are the same, but there are some selected by each method that was not selected by the other one.

Feature selection and categorical features

- What about categorical features?
- What if it removes one of the one-hot encoded feature and keeps other columns?
- Does it make sense to have one level missing like this?
- This seems to be unresolved in sklearn: https://github.com/scikit-learn/scikit-learn/issues/8480

```
bldg_type_cols = {col for col in X_train_enc.columns if col.startswith("BldgTy
bldg_type_cols
```

```
{'BldgType_1Fam',
 'BldgType_2fmCon',
 'BldgType_Duplex',
 'BldgType_Twnhs',
 'BldgType_TwnhsE'}
```

```
# Which bldg_type levels were removed in the feature selection?
bldg_type_cols - set(pipe_rf_rfe[:-1].get_feature_names_out())
```

```
{'BldgType_1Fam', 'BldgType_Duplex'}
```

(Optional) Feature selection and OHE

- If you are using OHE with `drop='first'`, and then feature selection, the choice to drop the first is no longer arbitrary.
  - Say you have categories A,B,C,D
  - you drop='first'
  - so then you have B,C,D and 0,0,0 means A.
  - Then you feature select out D
  - Now you have B,C
  - but now 0,0 means A or D.
  - So now the feature removed by feature selection becomes mixed up with the "arbitrary" feature A.

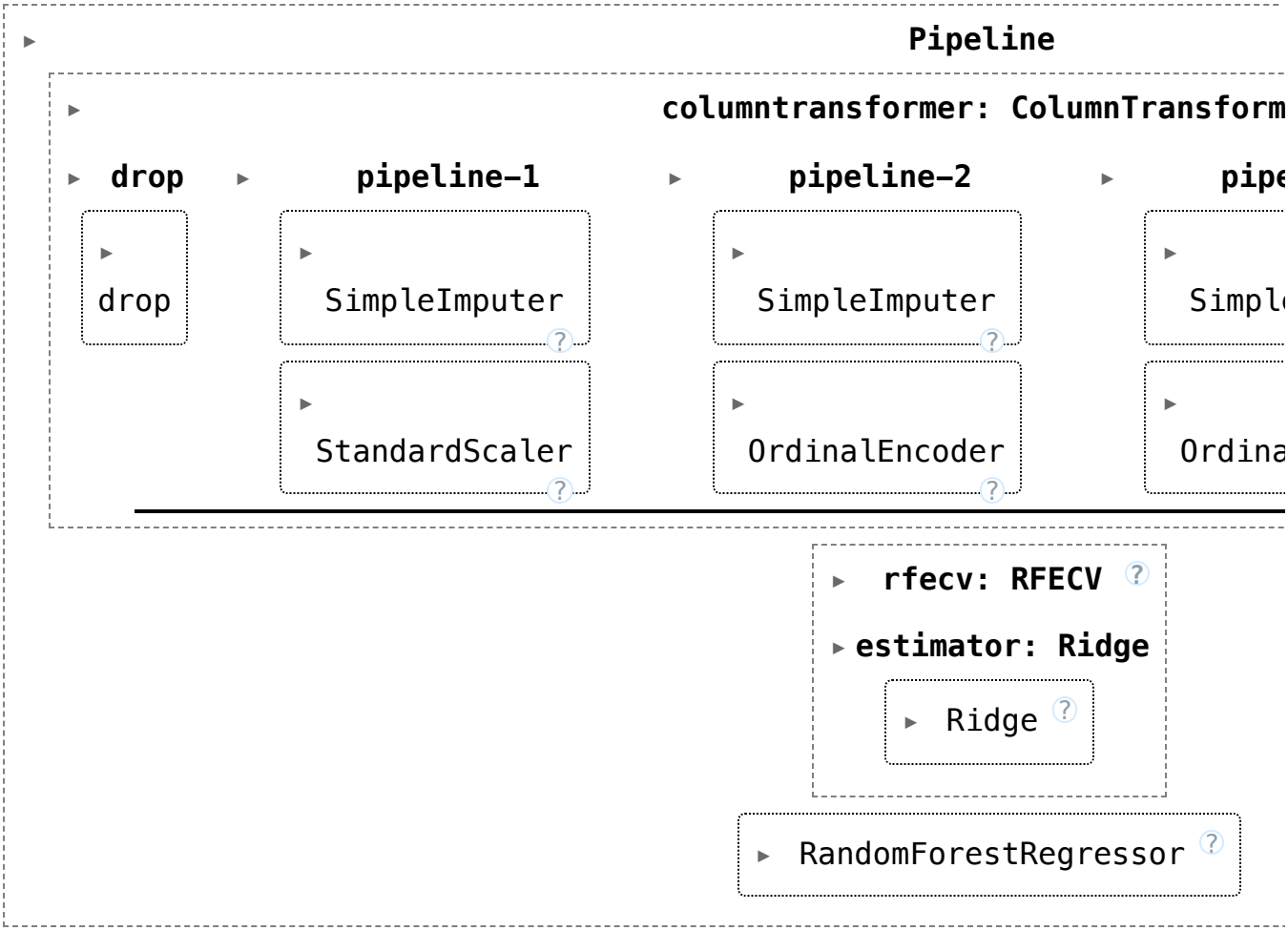
RFE with cross-validation (RFECV)

- We arbitrarily picked 120 features before.
- How do we know what value to pass to `n_features_to_select`?

- Use `RFECV` which uses cross-validation to select number of features.
- Slow because there is cross validation within cross validation

```
from sklearn.feature_selection import RFECV

rfecv = RFECV(Ridge())

pipe_rf_rfecv = make_pipeline(
    preprocessor, rfecv, RandomForestRegressor(random_state=42)
)
pipe_rf_rfecv.fit(X_train, y_train);
```

```
pipe_rf_rfecv
```



Are we improving the final scores with RFECV?

```
pipe_rf_rfecv = make_pipeline(
    preprocessor, rfecv, RandomForestRegressor(random_state=42)
)

results["rf+rfecv"] = mean_std_cross_val_scores(
    pipe_rf_rfecv, X_train, y_train, return_train_score=True
)
```

```
pd.DataFrame(results).T
```

	fit_time	score_time	test_score	train_score
dummy	0.003 (+/- 0.000)	0.001 (+/- 0.000)	-0.005 (+/- 0.006)	0.000 (+/- 0.000)
ridge	0.044 (+/- 0.003)	0.015 (+/- 0.001)	0.774 (+/- 0.127)	0.917 (+/- 0.003)
rf	2.456 (+/- 0.189)	0.020 (+/- 0.002)	0.858 (+/- 0.039)	0.979 (+/- 0.001)
model_based_fs + rf	1.006 (+/- 0.140)	0.019 (+/- 0.001)	0.846 (+/- 0.037)	0.978 (+/- 0.002)
rfe + rfe	1.589 (+/- 0.143)	0.020 (+/- 0.002)	0.862 (+/- 0.034)	0.980 (+/- 0.002)
rf+rfecv	4.934 (+/- 0.461)	0.018 (+/- 0.001)	0.846 (+/- 0.060)	0.976 (+/- 0.007)

How many features are selected?

```
pipe_rf_rfecv.named_steps["rfecv"].n_features_
```

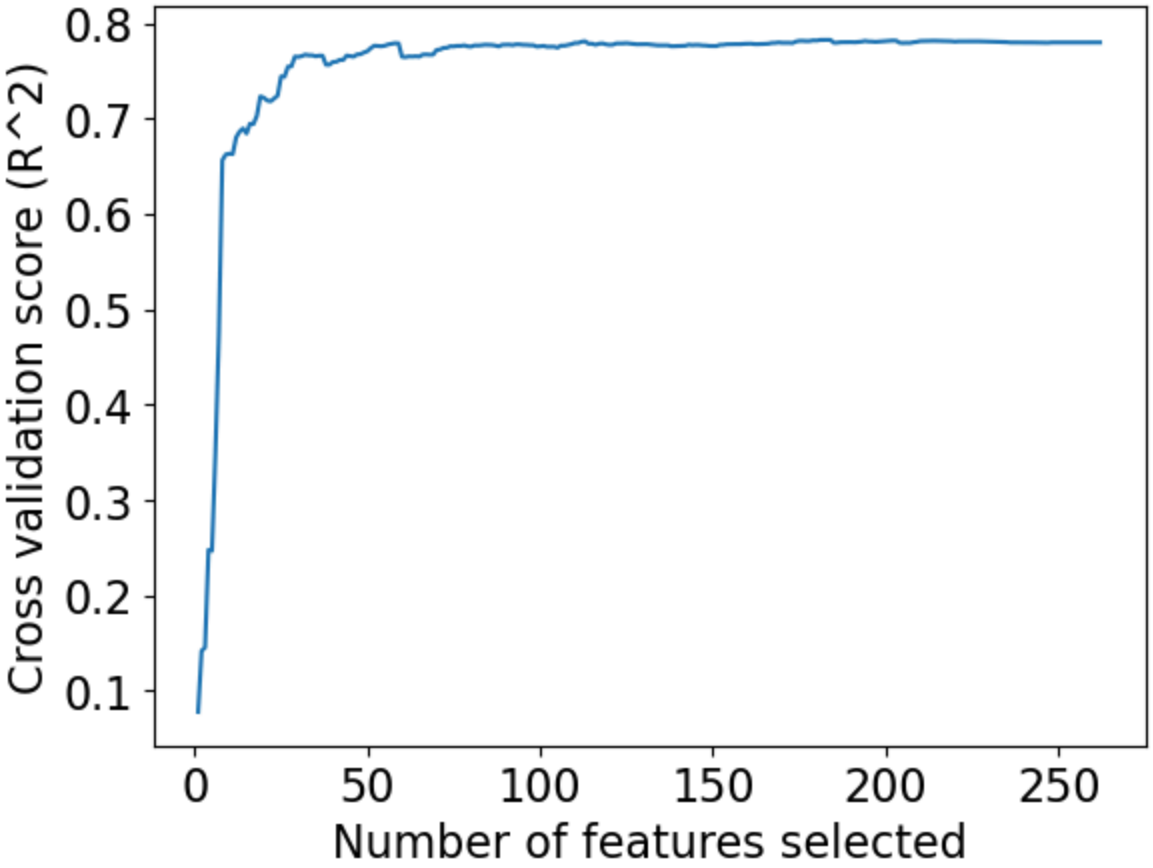
```
182
```

Let's examine number of features selected and corresponding cross-validation scores.

```
```

```
min_features_to_select = 1  # Minimum number of features to consider
pipe_rfecv = make_pipeline(
    preprocessor,
    RFECV(estimator=Ridge(), step=1, min_features_to_select=min_features_to_se
)
pipe_rfecv.fit(X_train, y_train)
plot_cross_validated_rfe(pipe_rfecv)
```

```
Optimal number of features : 182
```



- Our model is giving us the best results at number of features = 73.
- But in our case when we pass these 73 selected features to `RandomForestRegressor`, which is a non-linear model, the scores get worse compared to 120 features or model-based selection. That said, we have reduced the dimensionality a lot.
- This is similar to using cross-validation to tune a hyperparameter.
- Indeed, the number of selected features is a hyperparameter.

`sklearn` `feature_importances_`

- Do we have to use Ridge for feature selection?

- Many `sklearn` models have `feature_importances_` attribute.
- For tree-based models it's calculated based on impurity (gini index or information gain).
- For example, you can use `RandomForestClassifier` for feature selection. It will use `feature_importances_` attribute instead of coefficients.

Search and score methods

- Not based on feature importances.
- Can work for models which do not have feature importances.
- Define a **scoring function** $f(S)$ that measures the quality of the set of features S.
- Now **search** for the set of features S with the best score.

General idea of search and score methods

- Example: Suppose you have three features: A, B, C
  - Compute **score** for $S = \{\}$
  - Compute **score** for $S = \{A\}$
  - Compute **score** for $S = \{B\}$
  - Compute **score** for $S = \{C\}$
  - Compute **score** for $S = \{A, B\}$
  - Compute **score** for $S = \{A, C\}$
  - Compute **score** for $S = \{B, C\}$
  - Compute **score** for $S = \{A, B, C\}$
- Return S with the best score.
- How many distinct combinations we have to try out?
- This is intractable. So we go with greedy search methods such as forward or backward selection.

Forward or backward selection

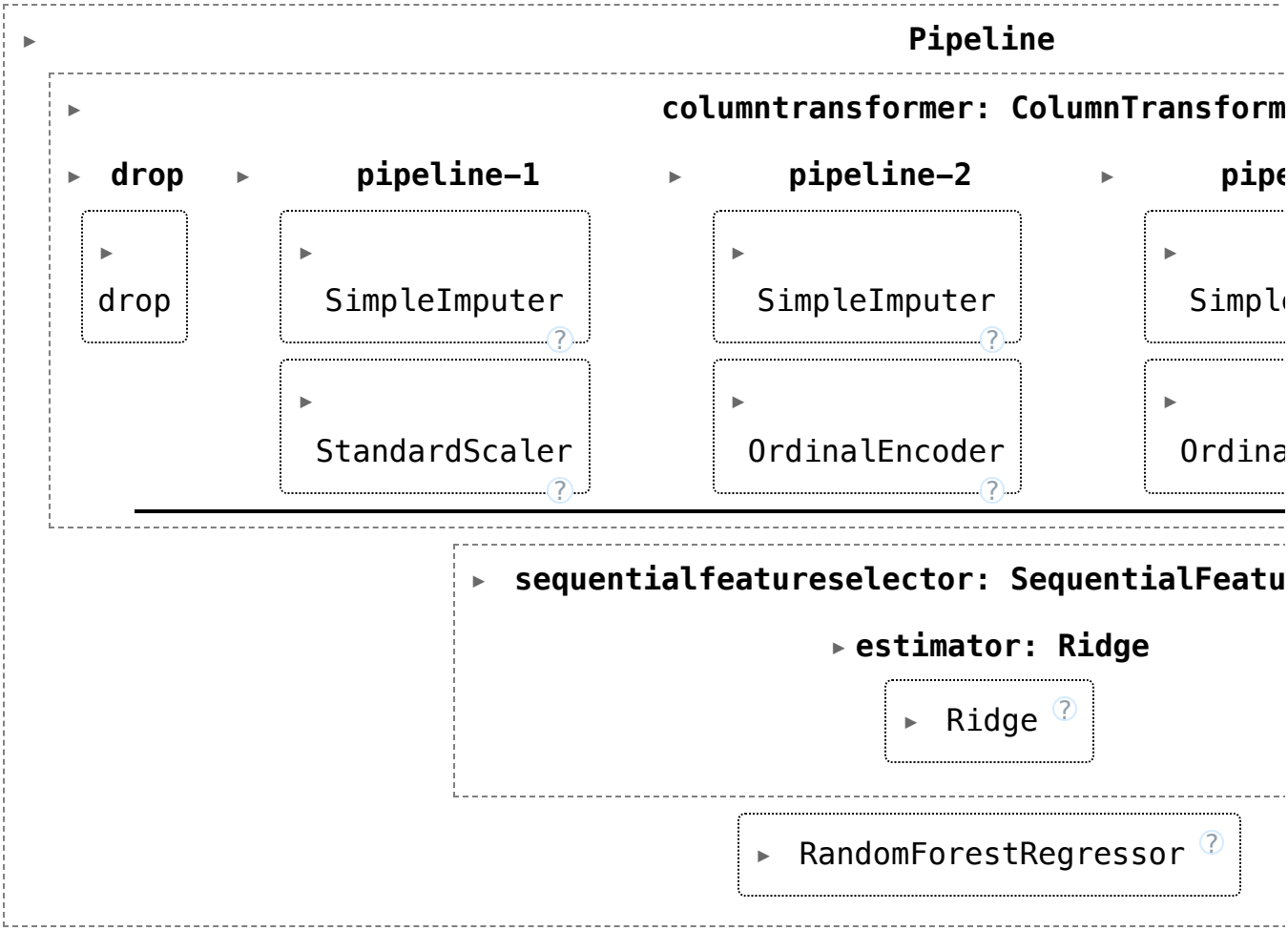
- Also called wrapper methods
- Shrink or grow feature set by removing or adding one feature at a time
- Makes the decision based on whether adding/removing the feature improves the CV score or not

Given $X = \{x_1, x_2, \ldots, x_n\}$ and y

1. Start with an empty set of features $S = \{\}$
2. Initialize the score (e.g., score = ∞, if score is validation error)
3. For each possible feature $x_j, 1 \leq j \leq n$
    - Compute the scores of features in S combined with x_j
4. If no x_j improves the score or desired number of selected features is reached, stop.
5. Otherwise add feature x_j to S which improves the score the most, update the score, and go back to step 2.

How do we use these methods in `sklearn`?

```
from sklearn.feature_selection import SequentialFeatureSelector

pipe_forward = make_pipeline(
    preprocessor,
    SequentialFeatureSelector(Ridge(), direction="forward"),
    RandomForestRegressor(random_state=42),
)

# results['rf_forward_fs'] = mean_std_cross_val_scores(pipe_forward, X_train,

pipe_forward.fit(X_train, y_train)
```

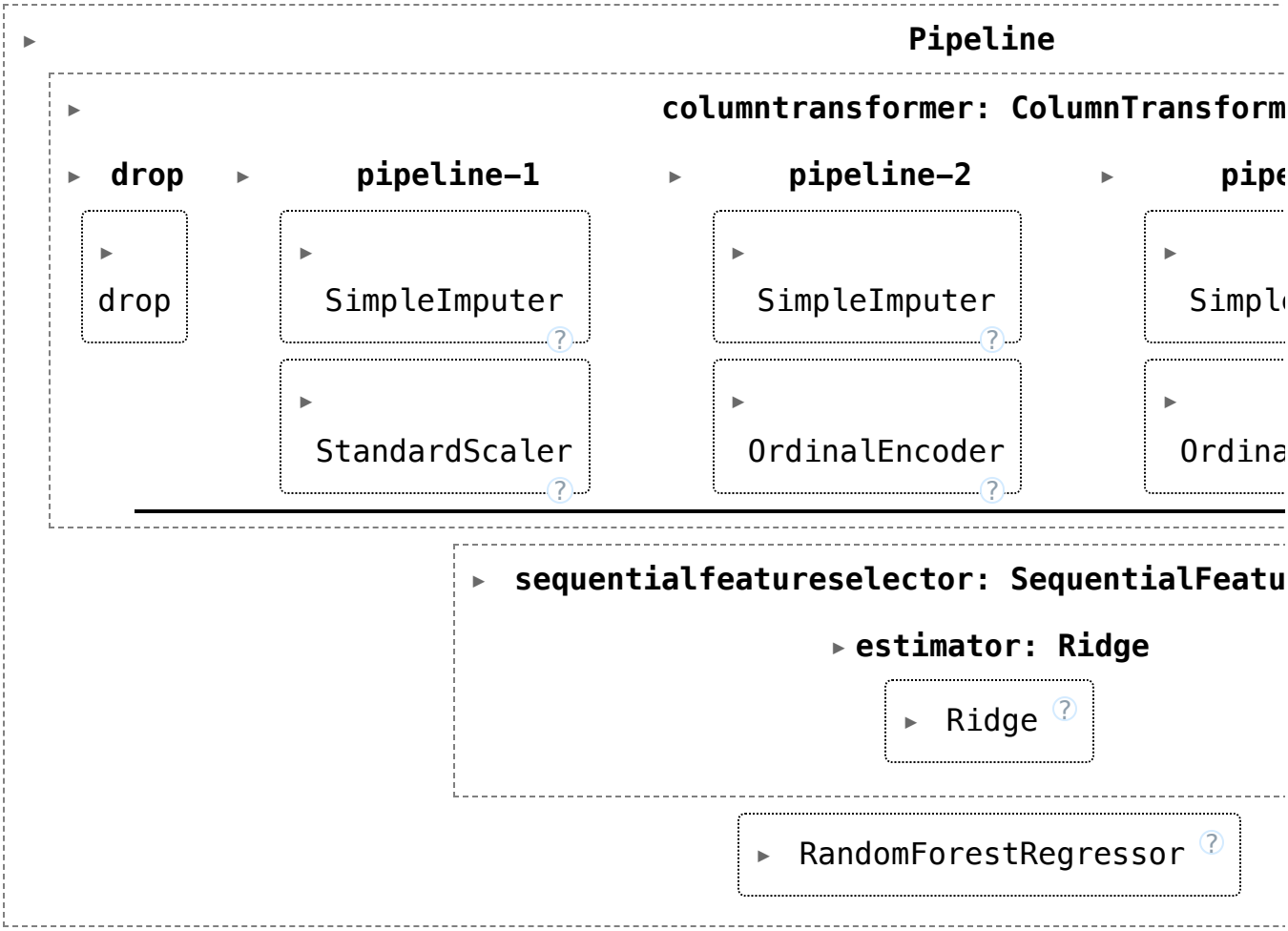



```
print("Train score: {:.3f}".format(pipe_forward.score(X_train, y_train)))
print("Test score: {:.3f}".format(pipe_forward.score(X_test, y_test)))
```

```
Train score: 0.978
Test score: 0.902
```

- VERY SLOW on this dataset and the scores did not improve.
- By default it's going to select $d/2$ features.

```
from sklearn.feature_selection import SequentialFeatureSelector

pipe_backward = make_pipeline(
    preprocessor,
    SequentialFeatureSelector(Ridge(), direction="backward"),
    RandomForestRegressor(random_state=42),
)

# results['rf_backward_fs'] = mean_std_cross_val_scores(pipe_forward, X_train,
pipe_backward.fit(X_train, y_train)
```




```
print("Train score: {:.3f}".format(pipe_backward.score(X_train, y_train)))
print("Test score: {:.3f}".format(pipe_backward.score(X_test, y_test)))
```

```
Train score: 0.977
Test score: 0.890
```

Other ways to search

- Stochastic local search
  - Inject randomness so that we can explore new parts of the search space
  - Simulated annealing
  - Genetic algorithms

Warnings about feature selection

- A feature's relevance is only defined in the context of other features.
  - Adding/removing features can make features relevant/irrelevant.
- If features can be predicted from other features, you cannot know which one to pick.
- Relevance for features does not have a causal relationship.
- Don't be overconfident.
  - The methods we have seen probably do not discover the ground truth and how the world really works.
  - They simply tell you which features help in predicting y_i for the data you have.

General advice on finding relevant features

- Try RFE, forward selection or other feature selection methods (e.g., simulated annealing, genetic algorithms)
- Talk to domain experts; they probably have an idea why certain features are relevant.
- Don't be overconfident.
  - The methods we have seen probably do not discover the ground truth and how the world really works.
  - They simply tell you which features help in predicting y_i.

❓❓ Questions for you

Select all the statements below which are True.

- (A) Simple association-based feature selection approaches do not take into account the interaction between features.
- (B) You can carry out feature selection using linear models by pruning the features which have very small weights (i.e., coefficients less than a threshold).
- (C) Forward search is guaranteed to find the best feature set.
- (D) RFE and backward selection are basically the same.
- (E) The order of features removed given by `rfe.ranking_` is the same as the order of original feature importances given by the model.

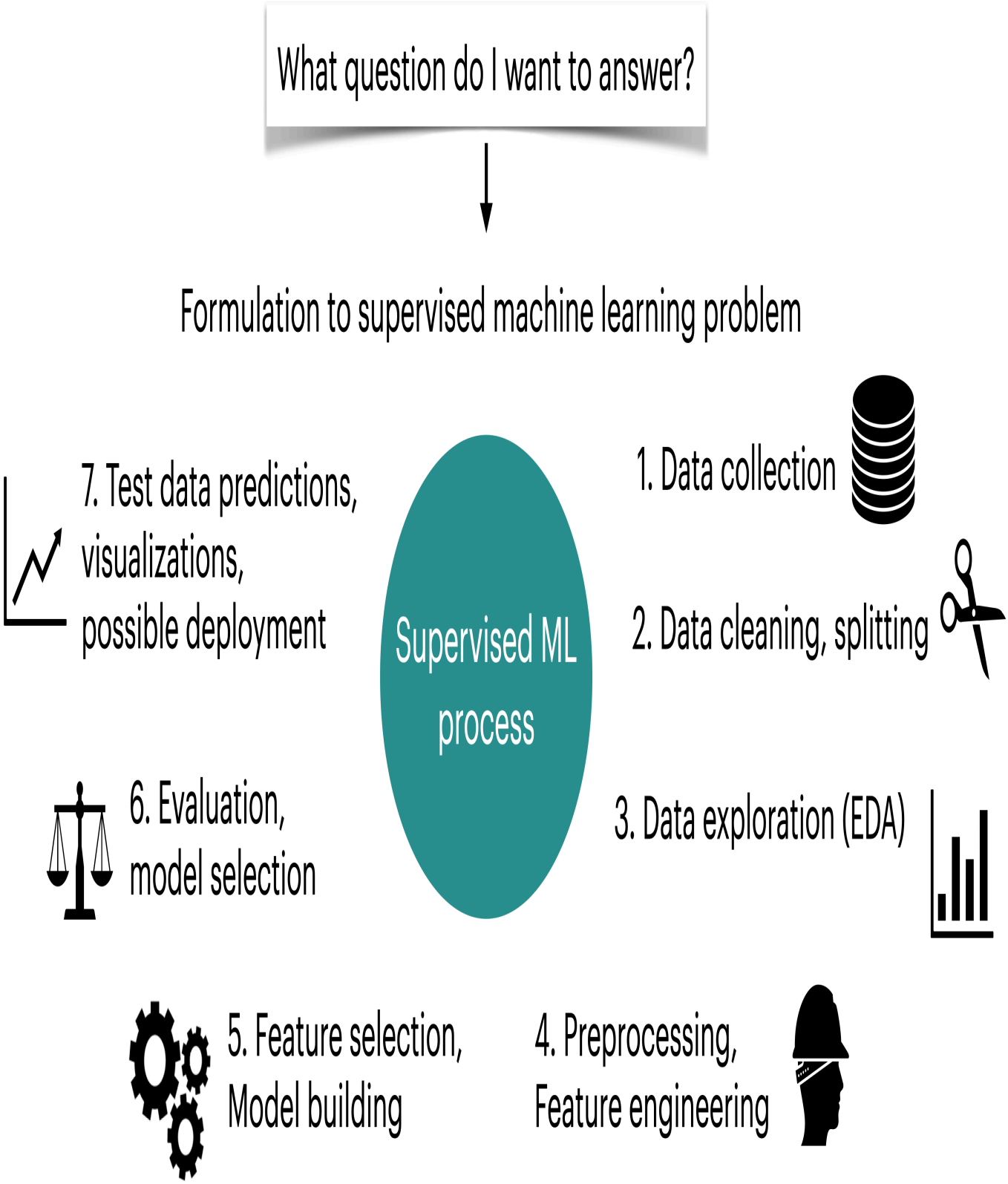
V's answers: A, B

Summary

What did we learn today?

- Feature importances
- Limitations of correlation based methods.
- How to interpret coefficients of different types of features
- Why do we need feature selection?
- Model based feature selection
- Recursive feature elimination
- Forward/backward selection

- We started with building a preliminary supervised machine learning pipeline in DSCI 571.
- Now we are refining the pipeline by adding more components in it.



Coming up

- Model selection with complexity penalty
- Regularization

(Optional) Problems with feature selection

- The term 'relevance' is not clearly defined.
- What all things can go wrong with feature selection?
- Attribution: From CPSC 340.

Example: Is "Relevance" clearly defined?

- Consider a supervised classification task of predicting whether someone has particular genetic variation (SNP)

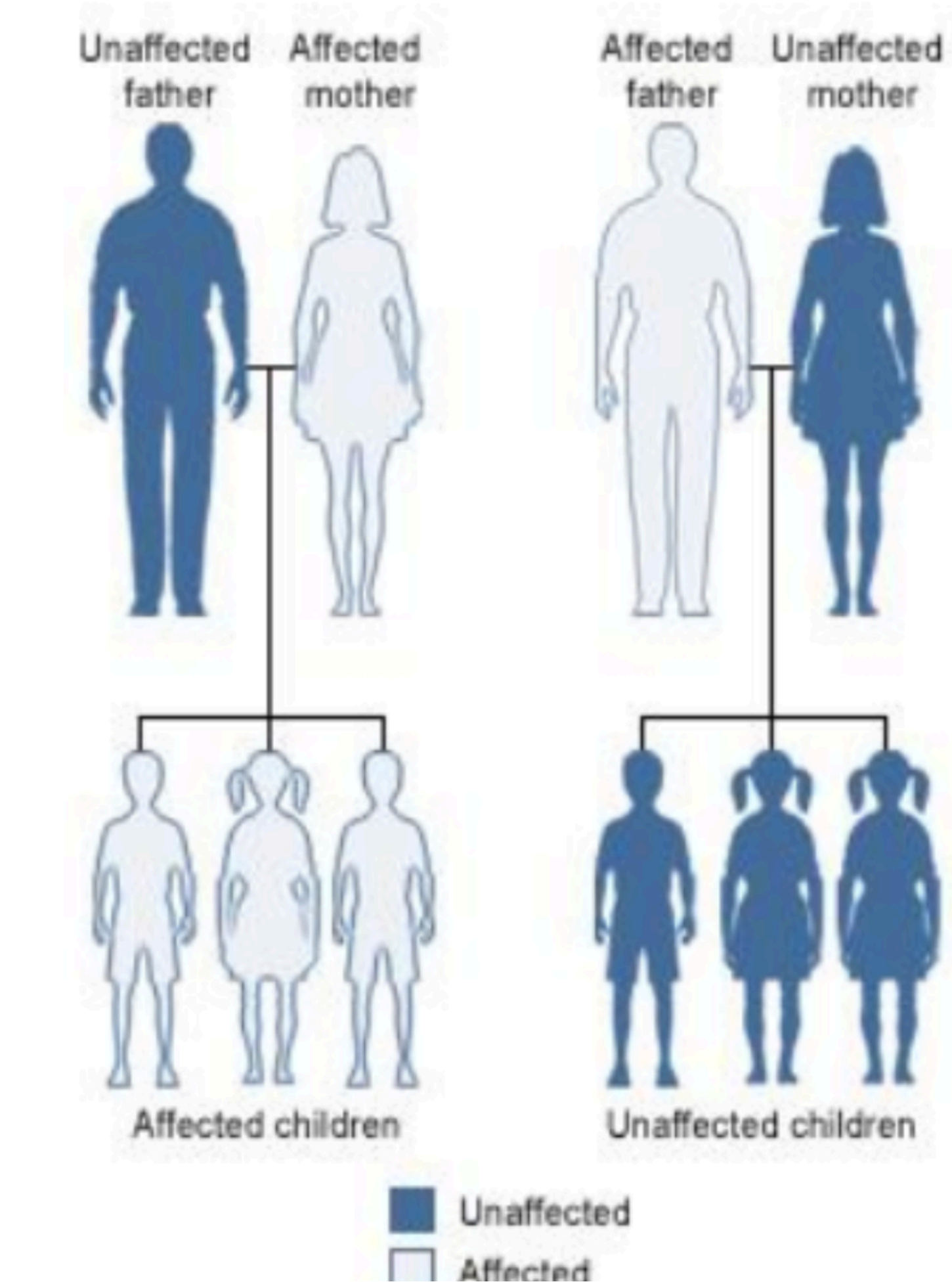
sex	biological dad	biological mom
F	0	1
M	1	0
F	0	0
F	1	1

SNP
1
0
0
1

- True model: You almost have the same value as your biological mom.

Is "Relevance" clearly defined?

- True model: You almost have the same value for SNP as your biological mom.
  - (SNP = biological mom) with very high probability
  - (SNP != biological mom) with very low probability


Unaffected father
Affected mother
Affected children
Affected father
Unaffected mother
Unaffected children
Unaffected
Affected

Is “Relevance” clearly defined?

- What if “mom” feature is repeated?
- Should we pick both? Should we pick one of them because it predicts the other?
- Dependence, collinearity for linear models
  - If a feature can be predicted from the other, don’t know which one to pick.

sex	biological dad	biological mom	mom2	SNP
F	0	1	1	1
M	1	0	0	0
F	0	0	0	0
F	1	1	1	1

Is “Relevance” clearly defined?

- What if we add (maternal) “grandma” feature?
- Is it relevant?
  - We can predict SNP accurately using this feature
- Conditional independence
  - But grandma is irrelevant given biological mom feature
  - Relevant features may become irrelevant given other features

sex	biological dad	biological mom	grandma	SNP
F	0	1	1	1
M	1	0	0	0
F	0	0	0	0
F	1	1	1	1

Is “Relevance” clearly defined?

- What if we do not know biological mom feature and we just have grandma feature
- It becomes relevant now.
  - Without mom feature this is the best we can do.
- General problem (“taco Tuesday” problem)
  - Features can become relevant due to missing information

sex	biological dad	grandma
F	0	1
M	1	0
F	0	0
F	1	1

SNP
1
0
0
1

Is “Relevance” clearly defined?

- Are there any relevant features now?
- They may have some common maternal ancestor.
- What if mom likes dad because they share SNP?
- General problem (Confounding)
  - Hidden features can make irrelevant features relevant.

sex	biological dad
F	0
M	1
F	0
F	1

SNP
1
0
0
1

Is “Relevance” clearly defined?

- Now what if we have “sibling” feature?
- The feature is relevant in predicting SNP but not the cause of SNP.
- General problem (non causality)
  - the relevant feature may not be causal

sex	biological dad	sibling	SNP
F	0	1	1
M	1	0	0
F	0	0	0
F	1	1	1

Is “Relevance” clearly defined?

- What if you are given “baby” feature?
- Now the sex feature becomes relevant.
  - “baby” feature is relevant when sex == F
- General problem (context specific relevance)
  - adding a feature can make an irrelevant feature relevant

sex	biological dad	baby	SNP
F	0	1	1
M	1	1	0
F	0	0	0
F	1	1	1

Warnings about feature selection

- A feature is only relevant in the context of other features.
  - Adding/removing features can make features relevant/irrelevant.
- Confounding factors can make irrelevant features the most relevant.
- If features can be predicted from other other features, you cannot know which one to pick.
- Relevance for features does not have a causal relationship.
- Is feature selection completely hopeless?
  - It is messy but we still need to do it. So we try to do our best!

Relevant resources

- Genome-wide association study
- sklearn feature selection
- PyData: A Practical Guide to Dimensionality Reduction Techniques