Lecture 8: Model transparency and conclusion

Contents

Learning outcomes

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

- Explain why are transparency and interpretability important in ML.
- Use `feature_importances_` attribute of `sklearn` tree-based models, interpret them, and being aware of their main drawbacks.
- Broadly explain how permutation feature importances are calculated and use `sklearn`'s `permutation_importance`.
- Create shapely values associated with tree-based models using the `SHAP` package and understand how to interpret them.
- Generate and interpret the force plot to explain a specific prediction.
- Explain and interpret beeswarm, bar, and scatter plots produced with shapely values.
- Learn to be aware of multicolinearity in your data when interpreting models.

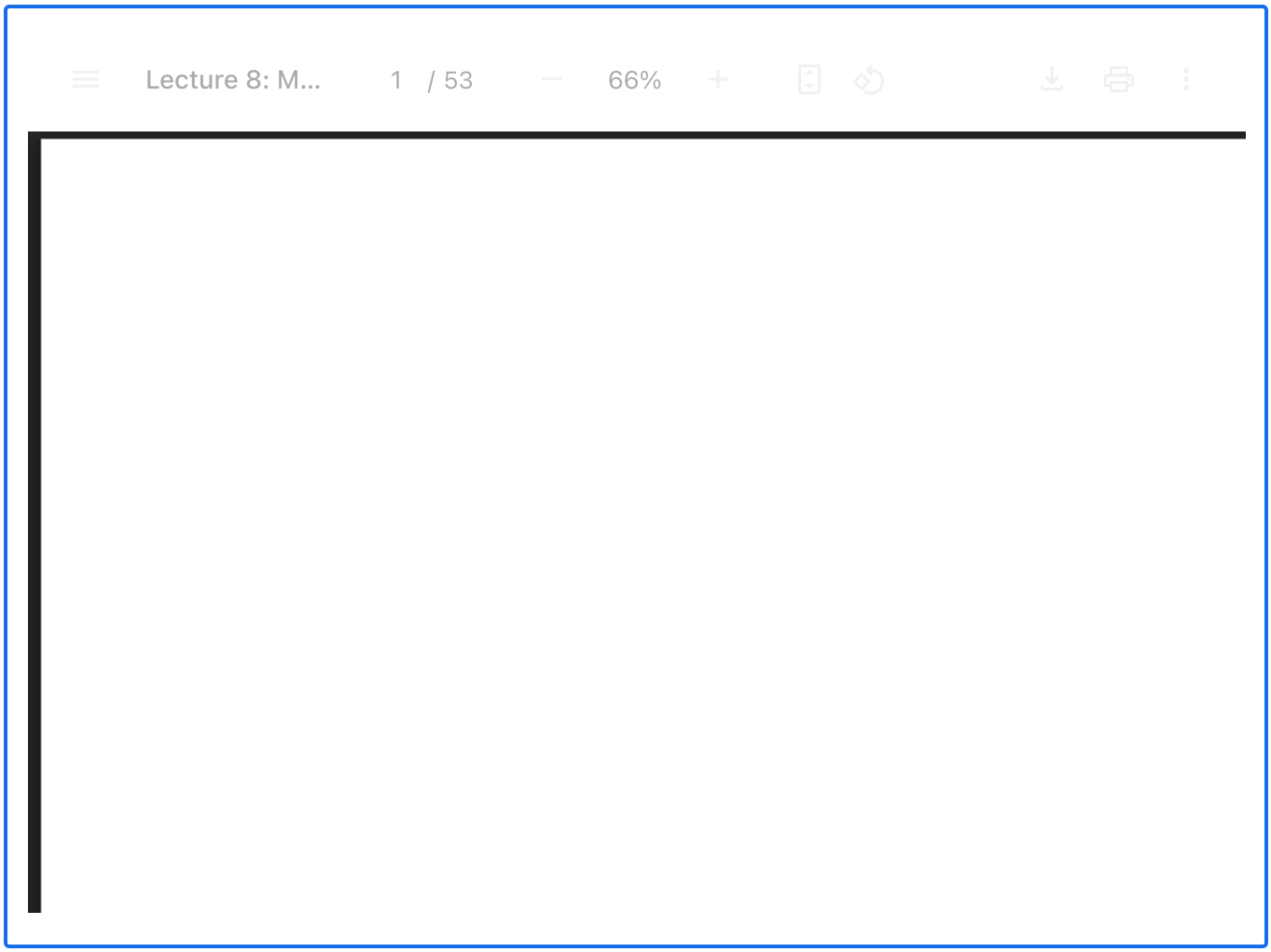
Imports and LO

Imports

```
import os
import sys

sys.path.append("code/.")
import string
from collections import deque

import matplotlib.pyplot as plt

%matplotlib inline
import numpy as np
import pandas as pd
from plotting_functions import *
from sklearn import datasets
from sklearn.compose import ColumnTransformer, make_column_transformer
from sklearn.dummy import DummyClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.impute import SimpleImputer
from sklearn.linear_model import LogisticRegression, Ridge, RidgeCV
from sklearn.model_selection import (
    cross_validate,
    train_test_split,
)
from sklearn.pipeline import Pipeline, make_pipeline
from sklearn.preprocessing import (
    OneHotEncoder,
    OrdinalEncoder,
    PolynomialFeatures,
    StandardScaler,
)
from sklearn.inspection import permutation_importance
from utils import *
```

Lecture slides

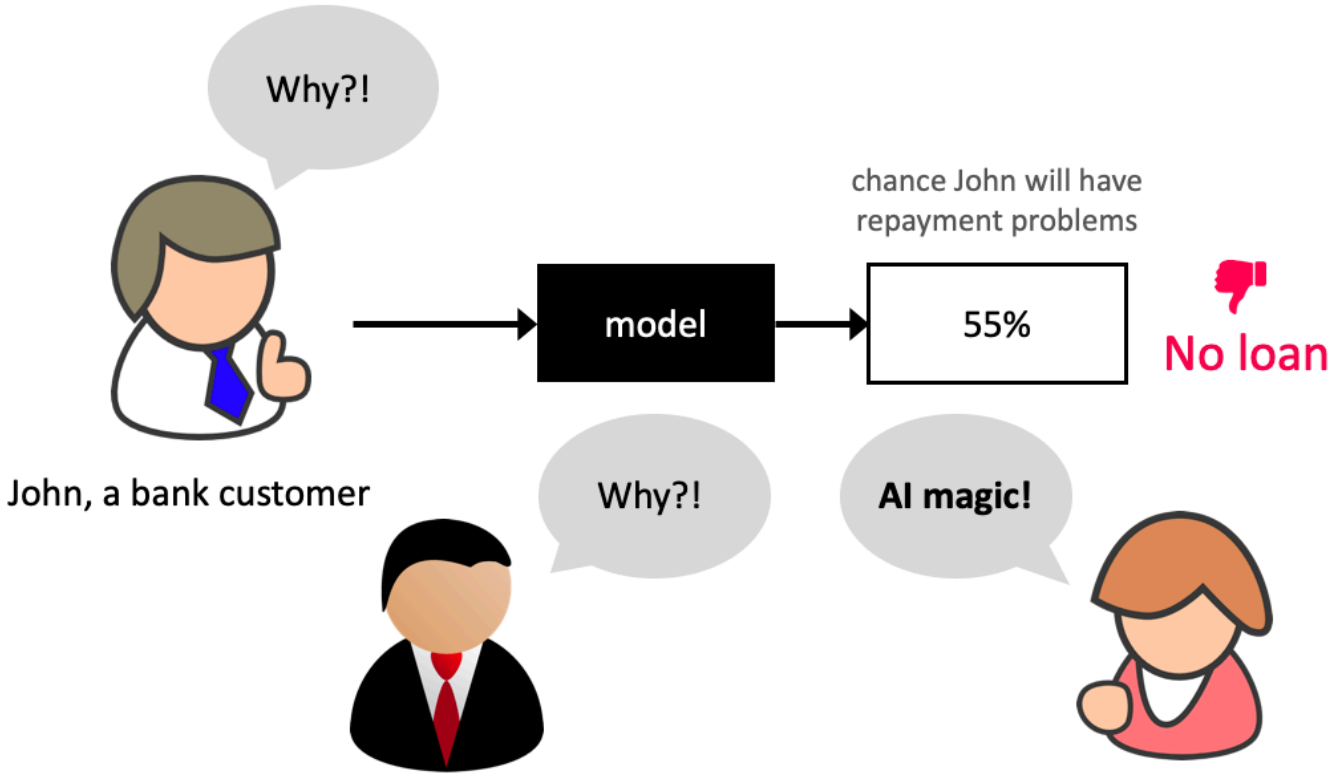
Section 1 slides | Section 2 slides



Transparency and explainability of ML models: Motivation

Why model transparency/interpretability?

- Ability to interpret ML models is crucial in many applications such as banking, healthcare, and criminal justice.
- It can be leveraged by domain experts to diagnose systematic errors and underlying biases of complex ML systems.



Source

What is model interpretability?

- In this lecture, our definition of model interpretability will be looking at **feature importances**, i.e., exploring features which are important to the model.
- There is more to interpretability than feature importances, but it's a good start!
- Resources:
  - Interpretable Machine Learning
  - Yann LeCun, Kilian Weinberger, Patrice Simard, and Rich Caruana: Panel debate on interpretability

Data

- Let's work with the adult census data set from the last lecture.

```
adult_df_large = pd.read_csv("data/adult.csv")
train_df, test_df = train_test_split(adult_df_large, test_size=0.2, random_sta
train_df_nan = train_df.replace("?", np.nan)
test_df_nan = test_df.replace("?", np.nan)
train_df_nan.head()
```

	age	workclass	fnlwgt	education	education.num	marital.status	occu
5514	26	Private	256263	HS-grad	9	Never-married	Craf
19777	24	Private	170277	HS-grad	9	Never-married	
10781	36	Private	75826	Bachelors	13	Divorced	
32240	22	State-gov	24395	Some-college	10	Married-civ-spouse	
9876	31	Local-gov	356689	Bachelors	13	Married-civ-spouse	s

```
numeric_features = ["age", "capital.gain", "capital.loss", "hours.per.week"]
categorical_features = [
    "workclass",
    "marital.status",
    "occupation",
    "relationship",
    "native.country",
]
ordinal_features = ["education"]
binary_features = ["sex"]
drop_features = ["race", "education.num", "fnlwgt"]
target_column = "income"
```

```
education_levels = [
    "Preschool",
    "1st-4th",
    "5th-6th",
    "7th-8th",
    "9th",
    "10th",
    "11th",
    "12th",
    "HS-grad",
    "Prof-school",
    "Assoc-voc",
    "Assoc-acdm",
    "Some-college",
    "Bachelors",
    "Masters",
    "Doctorate",
]
```

```
assert set(education_levels) == set(train_df["education"].unique())
```

```
numeric_transformer = make_pipeline(SimpleImputer(strategy="median"), Standard
tree_numeric_transformer = make_pipeline(SimpleImputer(strategy="median"))

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

ordinal_transformer = make_pipeline(
    SimpleImputer(strategy="constant", fill_value="missing"),
    OrdinalEncoder(categories=[education_levels], dtype=int),
)

binary_transformer = make_pipeline(
    SimpleImputer(strategy="constant", fill_value="missing"),
    OneHotEncoder(drop="if_binary", dtype=int),
)

preprocessor = make_column_transformer(
    ("drop", drop_features),
    (numeric_transformer, numeric_features),
    (ordinal_transformer, ordinal_features),
    (binary_transformer, binary_features),
    (categorical_transformer, categorical_features),
)
```

```
```

```
X_train = train_df_nan.drop(columns=[target_column])
y_train = train_df_nan[target_column]

X_test = test_df_nan.drop(columns=[target_column])
y_test = test_df_nan[target_column]
```

```
# encode categorical class values as integers for XGBoost
from sklearn.preprocessing import LabelEncoder
label_encoder = LabelEncoder()
y_train_num = label_encoder.fit_transform(y_train)
y_test_num = label_encoder.transform(y_test)
```

Do we have class imbalance?

- There is class imbalance. But without any context, both classes seem equally important.
- Let's use accuracy as our metric.

```
train_df_nan["income"].value_counts(normalize=True)
```

```
income
<=50K    0.757985
>50K     0.242015
Name: proportion, dtype: float64
```

```
scoring_metric = "accuracy"
```

Let's store all the results in a dictionary called `results`.

```
results = {}
```

We are going to use models outside sklearn. Some of them cannot handle categorical target values. So we'll convert them to integers using `LabelEncoder`.

```
y_train_num
```

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

Baseline

```
dummy = DummyClassifier()
results["Dummy"] = mean_std_cross_val_scores(
    dummy, X_train, y_train_num, return_train_score=True, scoring=scoring_metr
)
```

Different models

```
from lightgbm.sklearn import LGBMClassifier
from xgboost import XGBClassifier

pipe_lr = make_pipeline(
    preprocessor, LogisticRegression(max_iter=2000)
)
pipe_rf = make_pipeline(preprocessor, RandomForestClassifier(random_state=123))
pipe_xgb = make_pipeline(
    preprocessor, XGBClassifier(random_state=123, eval_metric="logloss", verbo
)
pipe_lgbm = make_pipeline(preprocessor, LGBMClassifier(random_state=123))
classifiers = {
    "logistic regression": pipe_lr,
    "random forest": pipe_rf,
    "XGBoost": pipe_xgb,
    "LightGBM": pipe_lgbm,
}
```

```
for (name, model) in classifiers.items():
    results[name] = mean_std_cross_val_scores(
        model, X_train, y_train_num, return_train_score=True, scoring=scoring_
    )
```

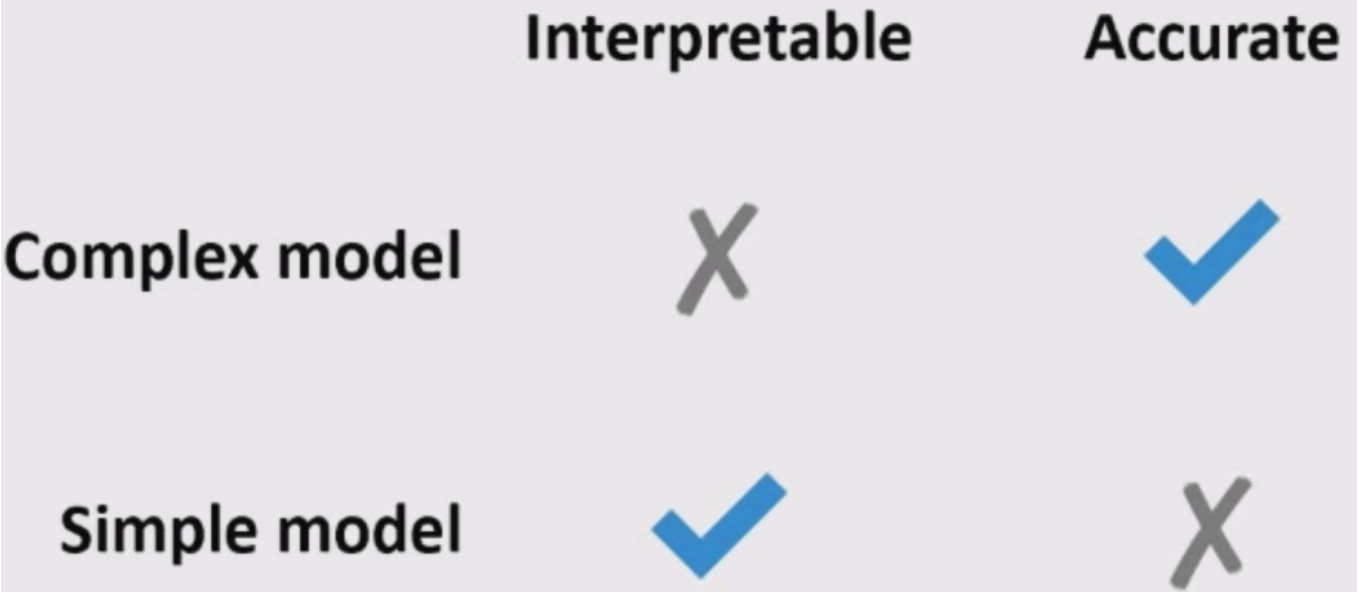
```
[LightGBM] [Info] Number of positive: 5043, number of negative: 15795
[LightGBM] [Info] Auto-choosing row-wise multi-threading, the overhead of testi
You can set `force_row_wise=true` to remove the overhead.
And if memory is not enough, you can set `force_col_wise=true`.
[LightGBM] [Info] Total Bins 450
[LightGBM] [Info] Number of data points in the train set: 20838, number of used
[LightGBM] [Info] [binary:BoostFromScore]: pavg=0.242010 -> initscore=-1.14169
[LightGBM] [Info] Start training from score -1.141692
[LightGBM] [Info] Number of positive: 5043, number of negative: 15795
[LightGBM] [Info] Auto-choosing row-wise multi-threading, the overhead of testi
You can set `force_row_wise=true` to remove the overhead.
And if memory is not enough, you can set `force_col_wise=true`.
[LightGBM] [Info] Total Bins 454
[LightGBM] [Info] Number of data points in the train set: 20838, number of used
[LightGBM] [Info] [binary:BoostFromScore]: pavg=0.242010 -> initscore=-1.14169
[LightGBM] [Info] Start training from score -1.141692
[LightGBM] [Info] Number of positive: 5043, number of negative: 15795
[LightGBM] [Info] Auto-choosing row-wise multi-threading, the overhead of testi
You can set `force_row_wise=true` to remove the overhead.
And if memory is not enough, you can set `force_col_wise=true`.
[LightGBM] [Info] Total Bins 451
[LightGBM] [Info] Number of data points in the train set: 20838, number of used
[LightGBM] [Info] [binary:BoostFromScore]: pavg=0.242010 -> initscore=-1.14169
[LightGBM] [Info] Start training from score -1.141692
[LightGBM] [Info] Number of positive: 5044, number of negative: 15795
[LightGBM] [Info] Auto-choosing row-wise multi-threading, the overhead of testi
You can set `force_row_wise=true` to remove the overhead.
And if memory is not enough, you can set `force_col_wise=true`.
[LightGBM] [Info] Total Bins 451
[LightGBM] [Info] Number of data points in the train set: 20839, number of used
[LightGBM] [Info] [binary:BoostFromScore]: pavg=0.242046 -> initscore=-1.14149
[LightGBM] [Info] Start training from score -1.141494
[LightGBM] [Info] Number of positive: 5043, number of negative: 15796
[LightGBM] [Info] Auto-choosing row-wise multi-threading, the overhead of testi
You can set `force_row_wise=true` to remove the overhead.
And if memory is not enough, you can set `force_col_wise=true`.
[LightGBM] [Info] Total Bins 451
[LightGBM] [Info] Number of data points in the train set: 20839, number of used
[LightGBM] [Info] [binary:BoostFromScore]: pavg=0.241998 -> initscore=-1.14175
[LightGBM] [Info] Start training from score -1.141756
```

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

	fit_time	score_time	test_score	train_score
Dummy	0.005 (+/- 0.001)	0.001 (+/- 0.000)	0.758 (+/- 0.000)	0.758 (+/- 0.000)
logistic regression	0.419 (+/- 0.079)	0.017 (+/- 0.002)	0.849 (+/- 0.005)	0.850 (+/- 0.001)
random forest	12.507 (+/- 0.281)	0.114 (+/- 0.002)	0.847 (+/- 0.006)	0.979 (+/- 0.000)
XGBoost	0.149 (+/- 0.015)	0.020 (+/- 0.001)	0.870 (+/- 0.004)	0.898 (+/- 0.001)
LightGBM	0.167 (+/- 0.004)	0.023 (+/- 0.001)	0.872 (+/- 0.004)	0.888 (+/- 0.000)

- Logistic regression is giving reasonable scores but not the best ones.
- XGBoost and LightGBM are giving us the best CV scores.

- Often simple models (e.g., linear models) are interpretable but not very accurate.
- Complex models (e.g., LightGBM) are more accurate but less interpretable.



Source

Feature importances in linear models

Let's create and fit a pipeline with preprocessor and logistic regression.

```
pipe_lr = make_pipeline(preprocessor, LogisticRegression(max_iter=2000))
pipe_lr.fit(X_train, y_train_num);
```

```
ohe_feature_names = (
    pipe_rf.named_steps["columntransformer"]
    .named_transformers_["pipeline-4"]
    .named_steps["onehotencoder"]
    .get_feature_names_out(categorical_features)
    .tolist()
)
feature_names = (
    numeric_features + ordinal_features + binary_features + ohe_feature_names
)
feature_names[:15]
```

```
['age',
 'capital.gain',
 'capital.loss',
 'hours.per.week',
 'education',
 'sex',
 'workclass_Federal-gov',
 'workclass_Local-gov',
 'workclass_Never-worked',
 'workclass_Private',
 'workclass_Self-emp-inc',
 'workclass_Self-emp-not-inc',
 'workclass_State-gov',
 'workclass_Without-pay',
 'workclass_missing']
```

```
data = {
    "coefficient": pipe_lr.named_steps["logisticregression"].coef_.flatten().t
    "magnitude": np.absolute(
        pipe_lr.named_steps["logisticregression"].coef_.flatten().tolist()
    ),
}
coef_df = pd.DataFrame(data, index=feature_names).sort_values(
    "magnitude", ascending=False
)
```

```
coef_df[:10]
```

	coefficient	magnitude
capital.gain	2.352062	2.352062
marital.status_Married-AF-spouse	1.578684	1.578684
occupation_Priv-house-serv	-1.556352	1.556352
relationship_Own-child	-1.343943	1.343943
marital.status_Never-married	-1.249807	1.249807
native.country_Columbia	-1.219141	1.219141
native.country_Dominican-Republic	-1.128432	1.128432
occupation_Farming-fishing	-1.113381	1.113381
workclass_Without-pay	-1.044948	1.044948
marital.status_Married-civ-spouse	1.036837	1.036837

- Increasing `capital.gain` is likely to push the prediction towards “>50k” income class
- Whereas occupation of private house service is likely to push the prediction towards “<=50K” income.

Can we get feature importances for non-linear models?

Model interpretability beyond linear models

- We will be looking at interpretability in terms of feature importances.
- Note that there is no absolute or perfect way to get feature importances. But it’s useful to get some idea on feature importances. So we just try our best.

We will be looking at two ways to get feature importances.

- `sklearn`’s `feature_importances_` and `permutation_importance`
- SHAP

`sklearn`'s `feature_importances_` and `permutation_importance`

Feature importance or **variable importance** is a score associated with a feature which tells us how "important" the feature is to the model.

Do we have correlated features?

```
X_train_enc = preprocessor.fit_transform(X_train).todense()
corr_df = pd.DataFrame(X_train_enc, columns=feature_names).corr().abs()
```

```
corr_df[corr_df == 1] = 0 # Set the diagonal to 0.
```

- Let's look at columns where any correlation number is > 0.80.
- 0.80 is an arbitrary choice

```
high_corr = [column for column in corr_df.columns if any(corr_df[column] > 0.8
print(high_corr)
```

```
['workclass_missing', 'marital.status_Married-civ-spouse', 'occupation_missing
```

Seems like there are some columns which are highly correlated.

```
corr_df['occupation_missing']['workclass_missing']
```

```
0.9977957422135846
```

```
corr_df['marital.status_Married-civ-spouse']['relationship_Husband']
```

```
0.8937442459553657
```

- When we look at the feature importances, we should be mindful of these correlated features.

- Remember the limitations of looking at simple linear correlations.
- You should probably investigate multi-colinearity with more sophisticated approaches (e.g., variance inflation factors (VIF) from DSCI 561).

`sklearn`'s `feature_importances_` attribute vs `permutation_importance`

- Feature importances can be
  - algorithm dependent, i.e., calculated based on the information given by the model algorithm (e.g., gini importance)
  - model agnostic (e.g., by measuring increase in prediction error after permuting feature values).
- Different measures give insight into different aspects of your data and model.

> Here you will find some drawbacks of using `feature_importances_` attribute in the context of tree-based models.

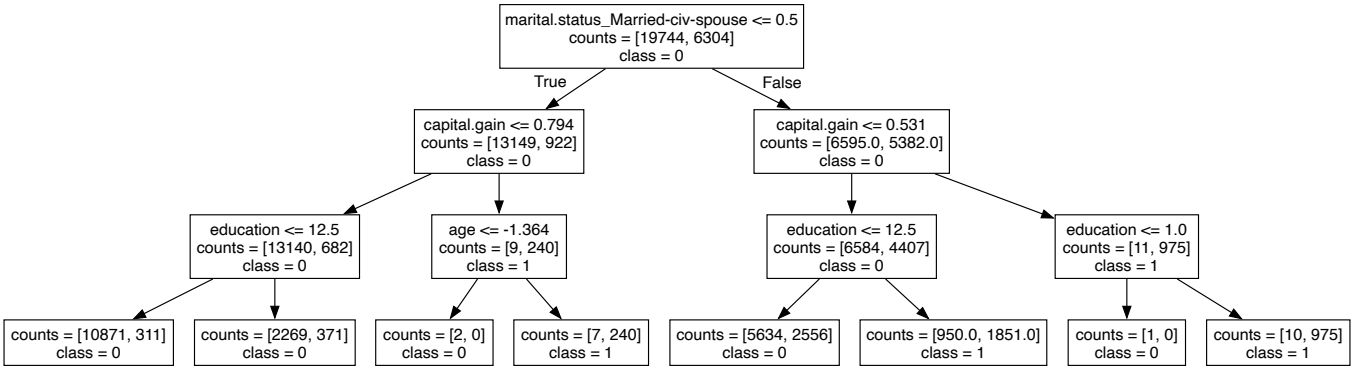
Decision tree feature importances

```
pipe_dt = make_pipeline(preprocessor, DecisionTreeClassifier(max_depth=3))
pipe_dt.fit(X_train, y_train_num);
```

```
data = {
    "Importance": pipe_dt.named_steps["decisiontreeclassifier"].feature_import
}
pd.DataFrame(data=data, index=feature_names,).sort_values(
    by="Importance", ascending=False
)[:10]
```

	Importance
marital.status_Married-civ-spouse	0.543351
capital.gain	0.294855
education	0.160727
age	0.001068
native.country_Ireland	0.000000
native.country_India	0.000000
native.country_Hungary	0.000000
native.country_Hong	0.000000
native.country_Honduras	0.000000
native.country_Holand-Netherlands	0.000000

```
display_tree(feature_names, pipe_dt.named_steps["decisiontreeclassifier"], cou
```

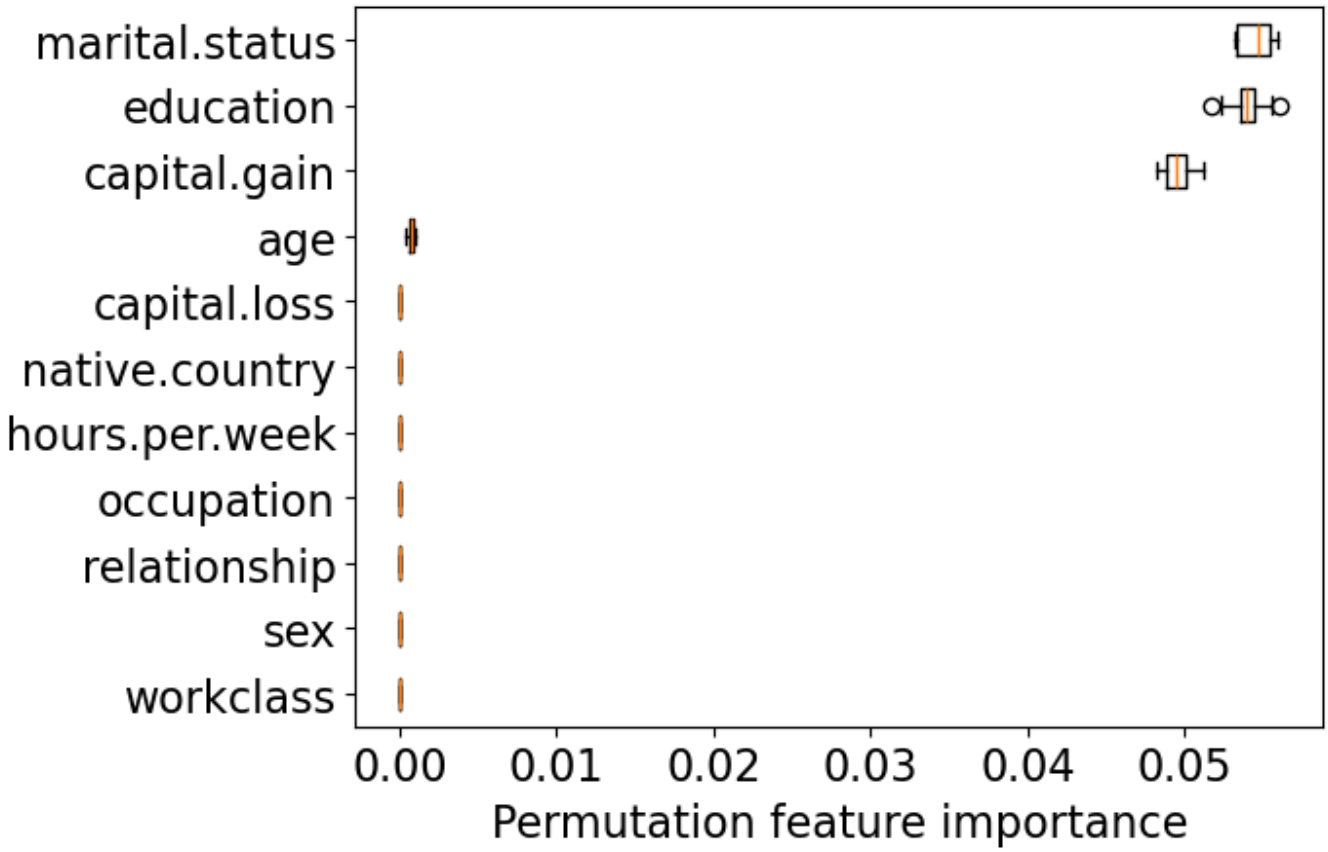


Let's explore permutation importance.

- For each feature this method evaluates the impact of permuting feature values

```
def get_permutation_importance(model):
    X_train_perm = X_train.drop(columns=["race", "education.num", "fnlwgt"])
    result = permutation_importance(model, X_train_perm, y_train_num, n_repeat
    perm_sorted_idx = result.importances_mean.argsort()
    plt.boxplot(
        result.importances[perm_sorted_idx].T,
        vert=False,
        tick_labels=X_train_perm.columns[perm_sorted_idx],
    )
    plt.xlabel('Permutation feature importance')
    plt.show()
```

```
get_permutation_importance(pipe_dt)
```



Decision tree is primarily making all decisions based on three features: marital.status, education, and capital.gain.

Let's create and fit a pipeline with preprocessor and random forest.

Random forest feature importances

```
pipe_rf = make_pipeline(preprocessor, RandomForestClassifier(random_state=2))
pipe_rf.fit(X_train, y_train_num);
```

Which features are driving the predictions the most?

```
data = {
    "Importance": pipe_rf.named_steps["randomforestclassifier"].feature_import
}
rf_imp_df = pd.DataFrame(
    data=data,
    index=feature_names,
).sort_values(by="Importance", ascending=False)
```

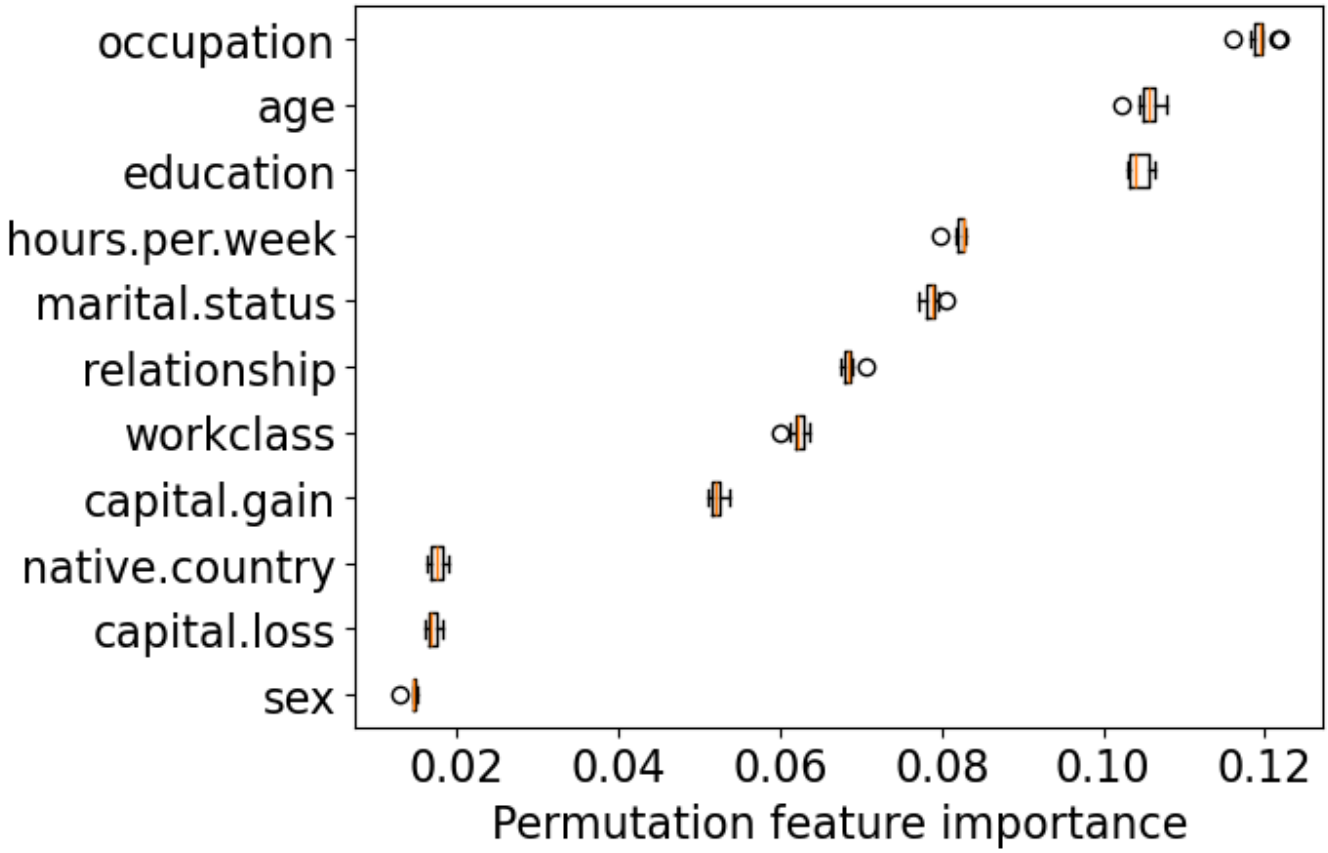
```
rf_imp_df[:8]
```

	Importance
age	0.230412
education	0.122210
hours.per.week	0.114521
capital.gain	0.113815
marital.status_Married-civ-spouse	0.077887
relationship_Husband	0.044242
capital.loss	0.038287
marital.status_Never-married	0.025661

```
np.sum(pipe_rf.named_steps["randomforestclassifier"].feature_importances_)
```

```
0.9999999999999998
```

```
get_permutation_importance(pipe_rf)
```



Random forest is using more features in the model compared to decision trees.

Key point

- Unlike the linear model coefficients, `feature_importances_` **do not have a sign**!
  - They tell us about importance, but not an "up or down".
  - Indeed, increasing a feature may cause the prediction to first go up, and then go down.
  - This cannot happen in linear models, because they are linear.

How can we get feature importances for non `sklearn` models?

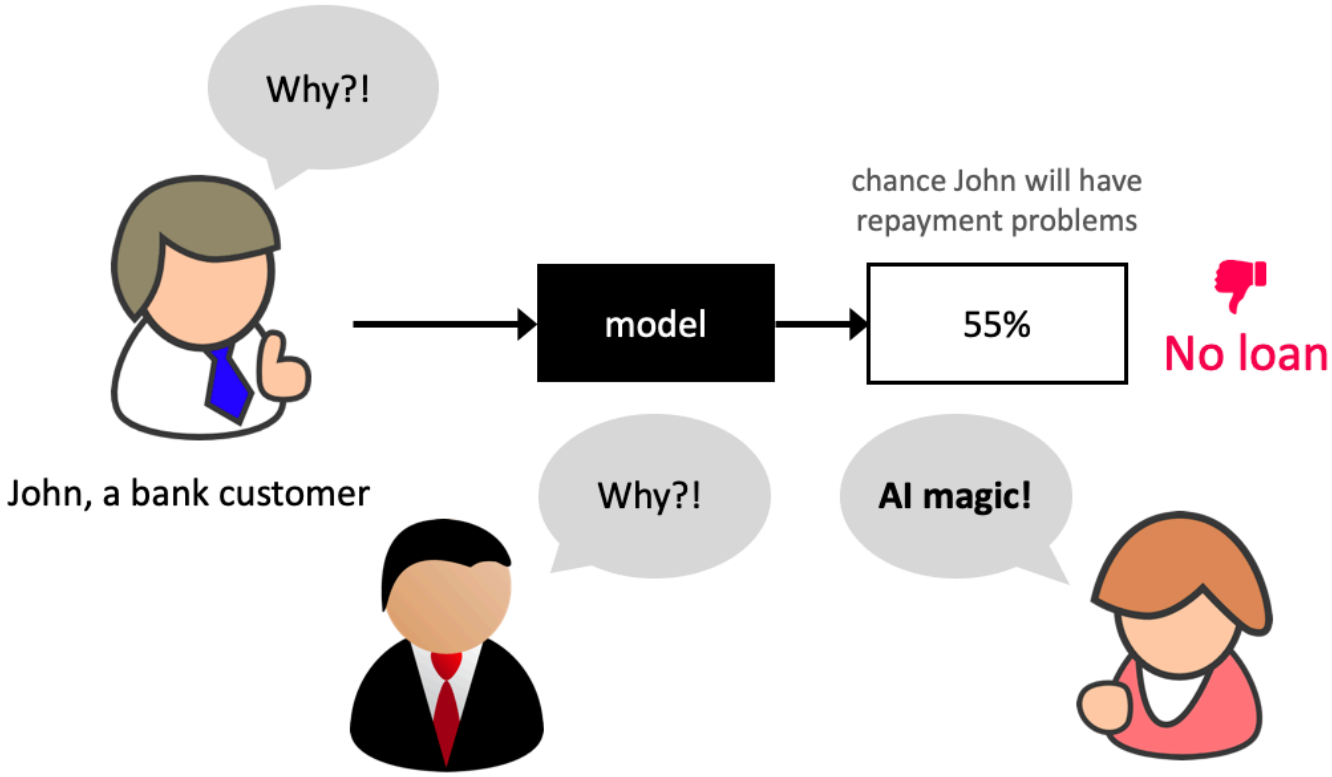
- These values tell us globally about which features are important.
- But what if you want to explain a *specific* prediction.
- Some fancier tools can help us do this.

Break

SHAP (SHapley Additive exPlanations)

introduction

Explaining a prediction



Source

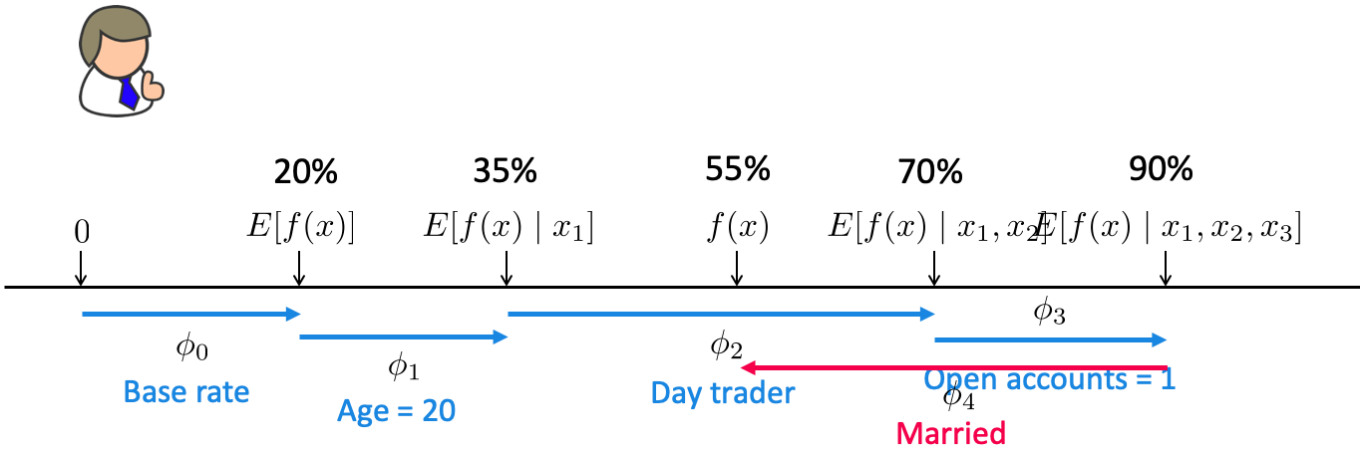
SHAP (SHapley Additive exPlanations)

- Based on the idea of shapely values. A shapely value is created for each example and each feature.
- Can explain the prediction of an example by computing the contribution of each feature to the prediction.
- Great visualizations
- Support for different kinds of models; fast variants for tree-based models
- Original paper: Lundberg and Lee, 2017

Our focus

- How to use it on our dataset?

- How to generate and interpret plots created by SHAP?
- We are not going to discuss how SHAP works.



Source

- Start at a base rate (e.g., how often people get their loans rejected).
- Add one feature at a time and see how it impacts the decision.

SHAP on LGBM model

- Let's try it out on our best performing LightGBM model.
- You should have `shap` in the course conda environment

Let's create the test dataframes with our transformed features.

```
# We need access to the transformed features and their feature names
X_test_enc = pd.DataFrame(
    data=preprocessor.transform(X_test).toarray(),
    columns=feature_names,
    index=X_test.index,
)
X_test_enc
```

	age	capital.gain	capital.loss	hours.per.week	education	sex	wo
14160	-0.701161	-0.147166	-0.21768	-0.042081	12.0	1.0	
27048	-1.437140	-0.147166	-0.21768	-2.069258	12.0	1.0	
28868	-0.774759	-0.147166	-0.21768	-0.042081	12.0	0.0	
5667	-0.259573	-0.147166	-0.21768	0.363354	6.0	1.0	
7827	-1.363542	-0.147166	-0.21768	-0.852952	12.0	0.0	
...	...	...	...	...	...	...	
1338	-0.848357	-0.147166	3.64915	2.390530	8.0	1.0	
24534	1.801168	-0.147166	-0.21768	-0.042081	8.0	1.0	
18080	1.285983	-0.147166	-0.21768	-0.042081	14.0	1.0	
10354	-1.216346	-0.147166	-0.21768	-1.663822	10.0	1.0	
24639	0.476406	-0.147166	-0.21768	1.985095	3.0	1.0	

6513 rows × 85 columns

Let's get SHAP values for train and test data.

```
import shap

# We need to fit the model before getting the importance
# (since they are based on making predictions with feature combinations)
pipe_lgbm.fit(X_train, y_train)

# Create a shap explainer object
lgbm_explainer = shap.TreeExplainer(
    pipe_lgbm.named_steps['lgbmclassifier'],
)
lgbm_explanation = lgbm_explainer(X_test_enc)
```

```
[LightGBM] [Info] Number of positive: 6304, number of negative: 19744
[LightGBM] [Info] Auto-choosing row-wise multi-threading, the overhead of testi
You can set `force_row_wise=true` to remove the overhead.
And if memory is not enough, you can set `force_col_wise=true`.
[LightGBM] [Info] Total Bins 463
[LightGBM] [Info] Number of data points in the train set: 26048, number of used
[LightGBM] [Info] [binary:BoostFromScore]: pavg=0.242015 -> initscore=-1.14166
[LightGBM] [Info] Start training from score -1.141665
```

Next, let's view the structure of the explanation object.

- `.values` are the actual shap values
- `.base_values` the expected value, E[f(x)], e.g. the predicted training data average in regression or proportion in classification
- `.data` the preprossed training data, i.e. `X_train_enc`

```
lgbm_explanation
```

```
.values =
array([[ 3.42314187e-02, -3.73942531e-01, -9.86760863e-02, ...,
         1.03183239e-03,  0.00000000e+00, -6.73362470e-04],
       [-3.00721661e+00, -2.12310564e-01, -4.62851000e-02, ...,
         8.30250692e-04,  0.00000000e+00,  7.84364014e-04],
       [-3.14108692e-01, -3.34083871e-01, -7.74339439e-02, ...,
         9.14510768e-04,  0.00000000e+00,  5.93783490e-04],
       ...,
       [ 8.38256268e-01, -2.65235113e-01, -9.41261613e-02, ...,
         1.03907710e-03,  0.00000000e+00,  6.30413770e-04],
       [-1.42941927e+00, -2.54388740e-01, -5.38819273e-02, ...,
         8.99856165e-04,  0.00000000e+00,  1.67690948e-03],
       [ 6.28595509e-01, -2.93821813e-01, -8.80047978e-02, ...,
         1.03559384e-03,  0.00000000e+00,  1.82672177e-03]])

.base_values =
array([-2.33641142, -2.33641142, -2.33641142, ..., -2.33641142,
       -2.33641142, -2.33641142])

.data =
array([[-0.70116088, -0.14716638, -0.21767954, ...,  0.        ,
         0.        ,  0.        ],
       [-1.43714007, -0.14716638, -0.21767954, ...,  0.        ,
         0.        ,  0.        ],
       [-0.7747588 , -0.14716638, -0.21767954, ...,  0.        ,
         0.        ,  0.        ],
       ...,
       [ 1.28598292, -0.14716638, -0.21767954, ...,  0.        ,
         0.        ,  0.        ],
       [-1.21634631, -0.14716638, -0.21767954, ...,  0.        ,
         0.        ,  0.        ],
       [ 0.47640581, -0.14716638, -0.21767954, ...,  0.        ,
         0.        ,  0.        ]])
```

- For each example, each feature, and each class we have a SHAP value.
- SHAP values tell us how to fairly distribute the prediction among features.
- For classification it's a bit confusing. It gives SHAP matrix for all classes.

- Let's stick to shap values for class 1, i.e., income > 50K.

SHAP plots

- Let's try to explain predictions on a couple of examples from the test data.
- I'm sampling some examples where target is <=50K and some examples where target is >50K.

```
y_test_reset = y_test.reset_index(drop=True)
y_test_reset
```

```
0       <=50K
1       <=50K
2       <=50K
3       <=50K
4       <=50K
        ...
6508    <=50K
6509    <=50K
6510     >50K
6511    <=50K
6512     >50K
Name: income, Length: 6513, dtype: object
```

```
l50k_ind = y_test_reset[y_test_reset == "<=50K"].index.tolist()
g50k_ind = y_test_reset[y_test_reset == ">50K"].index.tolist()

ex_l50k_index = l50k_ind[10]
ex_g50k_index = g50k_ind[10]
```

Explaining a prediction

Imagine that you are given the following test example.

```
X_test_enc.iloc[ex_l50k_index]
```

```
age                                0.476406
capital.gain                      -0.147166
capital.loss                       4.649658
hours.per.week                    -0.042081
education                          8.000000
                                     ...
native.country_Trinadad&Tobago     0.000000
native.country_United-States       1.000000
native.country_Vietnam             0.000000
native.country_Yugoslavia          0.000000
native.country_missing             0.000000
Name: 345, Length: 85, dtype: float64
```

You get the following hard prediction, which you are interested in explaining.

```
pipe_lgbm.named_steps["lgbmclassifier"].predict(X_test_enc)[ex_l50k_index]
```

```
'<=50K'
```

You can first look at `predict_proba` output to get a better understanding of model confidence.

```
pipe_lgbm.named_steps["lgbmclassifier"].predict_proba(X_test_enc)[ex_l50k_inde
```

```
array([0.99240562, 0.00759438])
```

- The model seems quite confident. But if we want to know more, for example, which feature values are playing a role in this specific prediction, we can use SHAP waterfall plots.
- Remember that we have SHAP values per feature per example. We'll use these values to create SHAP plots.

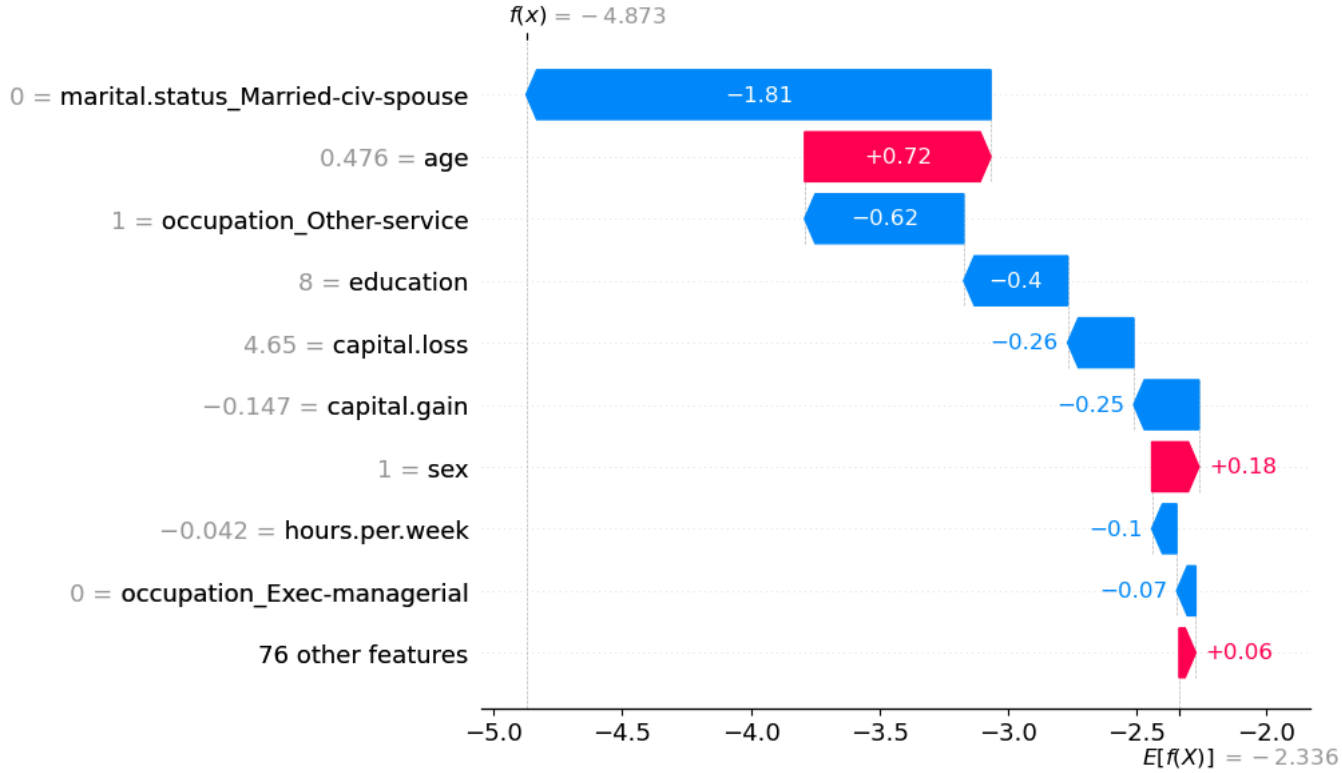
```
pd.DataFrame(
    lgbm_explanation[ex_l50k_index, :].values,
    index=feature_names,
    columns=["SHAP values"],
).sort_values("SHAP values")
```

	SHAP values
marital.status_Married-civ-spouse	-1.805103
occupation_Other-service	-0.618201
education	-0.403715
capital.loss	-0.256666
capital.gain	-0.253426
...	...
relationship_Own-child	0.024354
occupation_Machine-op-inspct	0.026332
marital.status_Never-married	0.053894
sex	0.183267
age	0.723502

85 rows × 1 columns

```
# load JS visualization code to notebook
shap.initjs()
```

```
shap.plots.waterfall(lgbm_explanation[ex_l50k_index, :])
```

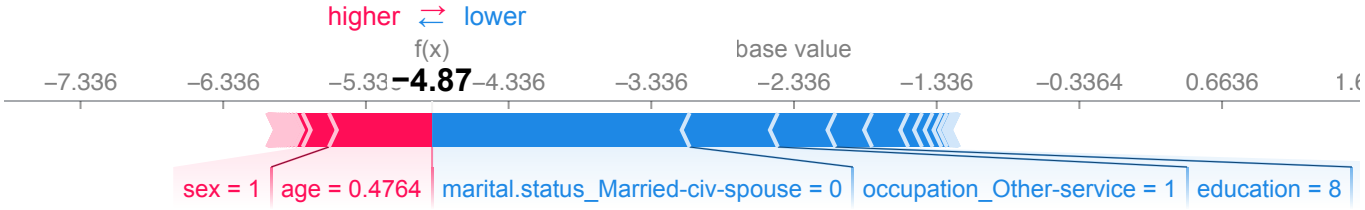


The reason that the shap values and the expected value above are not 0-1 is that LGBM uses logloss rather than proabilities as its default loss metric, which is not as interpretable. We could try to do something like this when we set up the explainer, but there are some scenarios where this does not work due to limitations in the SHAP API, so we will stick, so we will stick to the raw log odds here, although probabilities would be easier to interpret and communicate.

```
lgbm_explainer = shap.TreeExplainer(
    pipe_lgbm.named_steps["lgbmclassifier"],
    data=X_test_enc,
    model_output='predict_proba'
)
```

A force plots is another way of showing this info, but it a bit more busy and can be harder to interpret.

```
shap.plots.force(lgbm_explanation[ex_l50k_index, :])
```



- The raw model score is much smaller than the base value, which is reflected in the prediction of <= 50k class.
- sex = 1.0, scaled age = 0.48 are pushing the prediction towards higher score.
- education = 8.0, occupation_Other-service = 1.0 and marital.status_Married-civ-spouse = 0.0 are pushing the prediction towards lower score.

```
pipe_lgbm.named_steps["lgbmclassifier"].classes_
```

```
array(['<=50K', '>50K'], dtype=object)
```

We can get the raw model output (the log odds) by passing `raw_score=True` in `predict`.

```
pipe_lgbm.named_steps["lgbmclassifier"].predict(X_test_enc, raw_score=True)
```

```
array([-1.76270194, -7.61912405, -0.45555535, ...,  1.13521135,
       -6.62873917, -0.84062193])
```

What's the raw score of the example above we are trying to explain?

- The score matches with what we see in the force plot.
- The base score above is the mean raw score prediction on the training data. Our example has a lower raw score compared to the base raw score and the force plot tries to explain which feature values are bringing this score to a lower value.

```
lgbm_explainer.expected_value  # The base raw score
```

```
-2.3364114233677307
```

```
# Same as the mean prediction on the training data
pipe_lgbm.predict(X_train, raw_score=True).mean()
```

```
-2.336411423367732
```

Note: a nice thing about SHAP values is that the feature importances sum to the prediction:

```
lgbm_explanation[ex_l50k_index, :].values.sum() + lgbm_explainer.expected_valu
```

```
-4.8727229084399575
```

Now let's try to explain another prediction.

- The hard prediction here is 1.
- From the `predict_proba` and raw score output it seems like the model is not too confident about the prediction.

```
pipe_lgbm.named_steps["lgbmclassifier"].predict(X_test_enc)[ex_g50k_index]
```

```
'>50K'
```

```
# X_test_enc.iloc[ex_g50k_index]
```

```
pipe_lgbm.named_steps["lgbmclassifier"].predict_proba(X_test_enc)[ex_g50k_inde
```

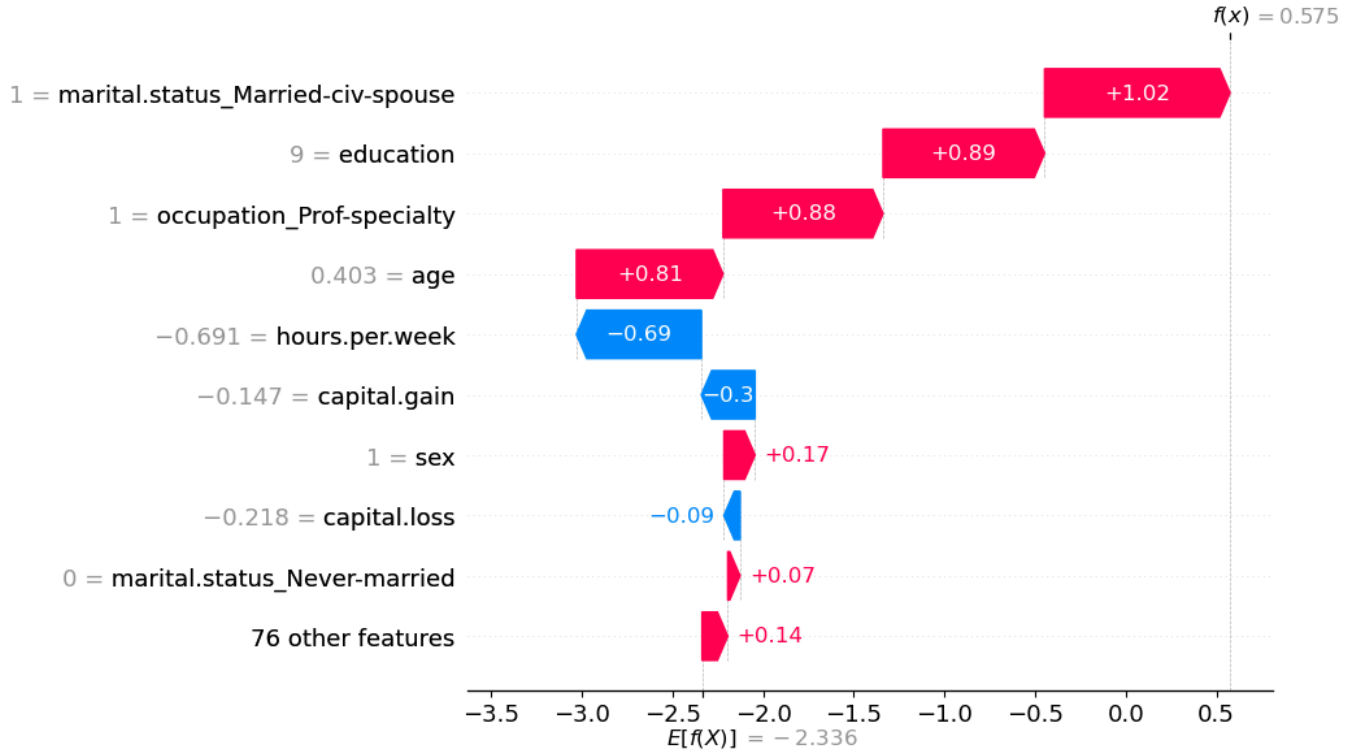
```
array([0.35997929, 0.64002071])
```

What's the raw score for this example?

```
pipe_lgbm.named_steps["lgbmclassifier"].predict(X_test_enc, raw_score=True)[ex
```

```
0.5754540510801829
```

```
shap.plots.waterfall(lgbm_explanation[ex_g50k_index, :])
```

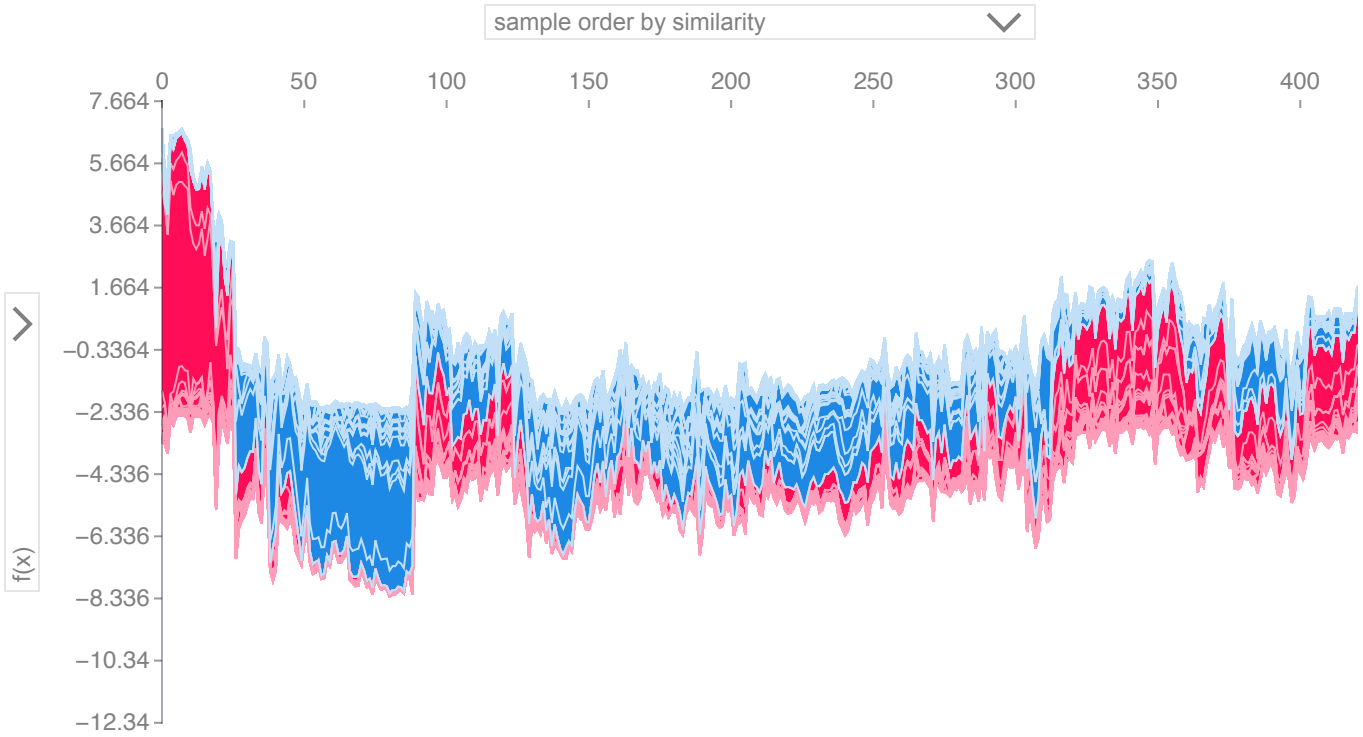


Observations:

- The base value, i.e., the average raw score is -2.336.
- We see the forces that drive the prediction.
- That is, we can see the main factors pushing it from the base value (average over the dataset) to this particular prediction.
- Features that push the prediction to a higher value are shown in red.
- Features that push the prediction to a lower value are shown in blue.

We can also see clusters of of samples with similar shap values by using the force plot on all the observations instead of just a single one.

```
# We sample the data because the plot is slow for thousands of rows
shap.plots.force(lgbm_explanation[:, :].sample(500))
```



Global feature importance using SHAP

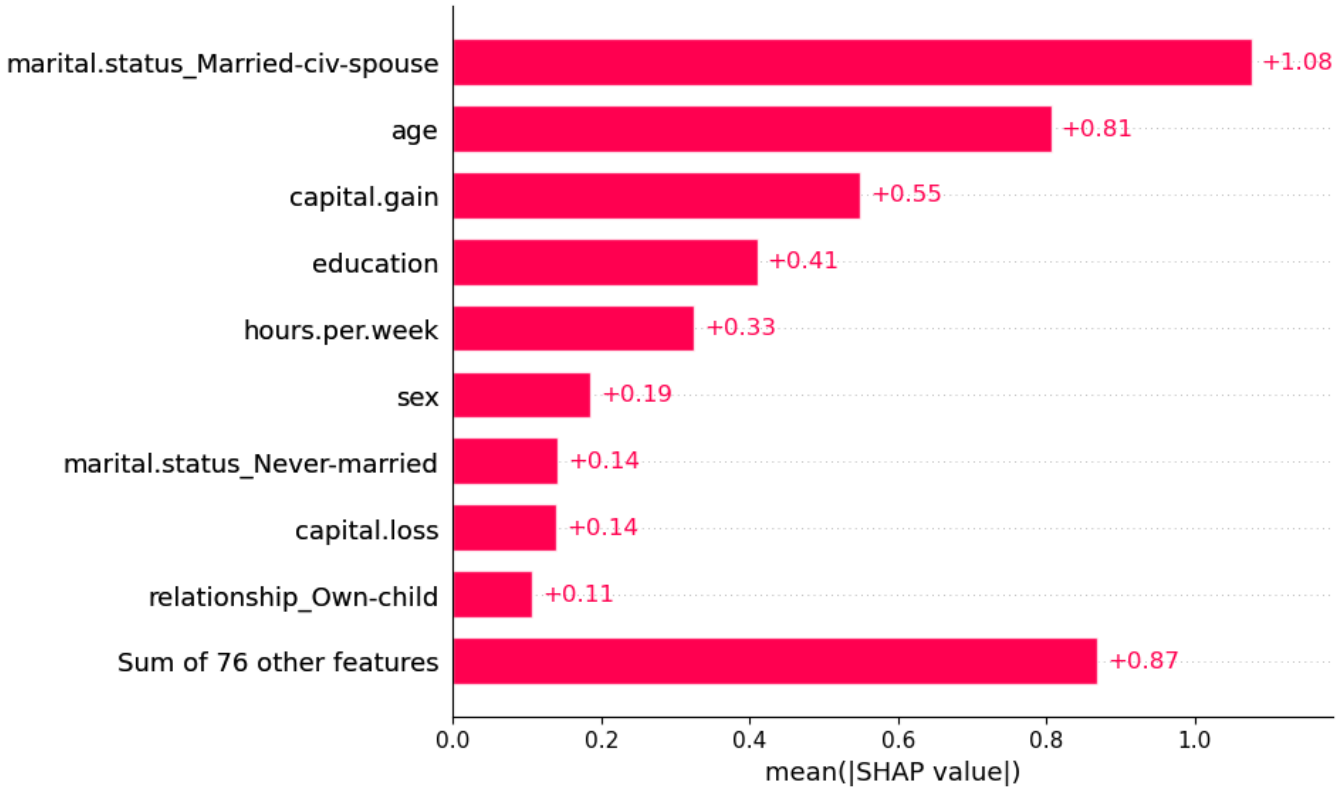
Let's look at the average SHAP values associated with each feature.

```
pd.Series(
    np.abs(lgbm_explanation[:, :].values).mean(axis=0),
    index=feature_names
).sort_values()
```

```
native.country_Yugoslavia             0.000000
workclass_Without-pay                 0.000000
native.country_Japan                  0.000000
marital.status_Married-AF-spouse      0.000000
native.country_Laos                   0.000000
                                          ...
hours.per.week                        0.325179
education                             0.410481
capital.gain                          0.549726
age                                   0.806692
marital.status_Married-civ-spouse     1.076705
Length: 85, dtype: float64
```

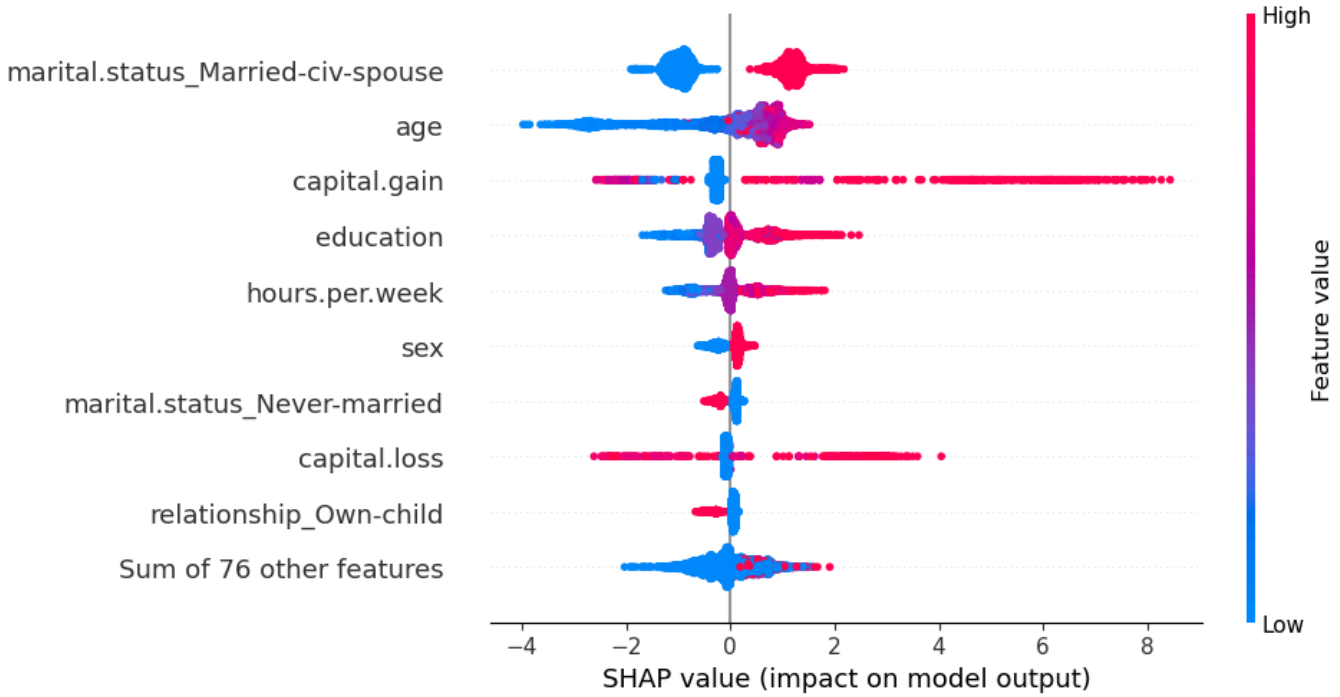
This average value can be thought of as a global feature importance and we can visualize it with a barplot in shap.

```
shap.plots.bar(lgbm_explanation[:, :])
```



Creating a beeswarm plot is more informative as it tells us about both the average effect (the sort order), but also how the feature affects the shap value. We can e.g. see that although `capital.loss` does not have a high average effect, there are many observations where it has a really high effect on the prediction (i.e. it matters a lot, but just to some individuals).

```
shap.plots.beeswarm(lgbm_explanation[:, :])
```

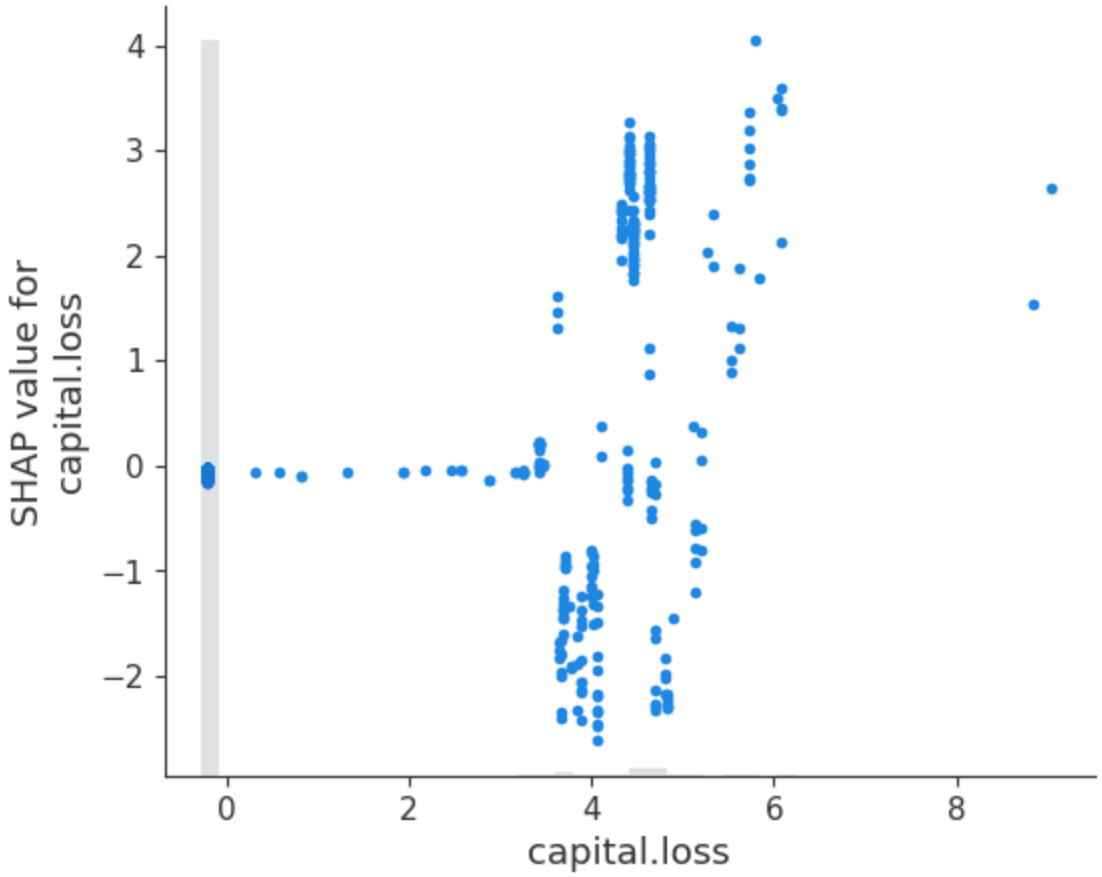


We could also see that:

- Presence of the marital status of Married-civ-spouse seems to have bigger SHAP values for class 1 and absence seems to have smaller SHAP values for class 1.
- Higher levels of education seem to have bigger SHAP values for class 1 whereas smaller levels of education have smaller SHAP values.

We could have a closer look at the relationship for `capital.loss` by creating a scatter plot showing the character of the relationship between the variable and the shap value (e.g. linear/non-linear). Here we can see that there seems to a big effect for any value that is not 0, but likely there are many zeros that drown out the average importance of the feature.

```
shap.plots.scatter(lgbm_explanation[:, 'capital.loss'])
```



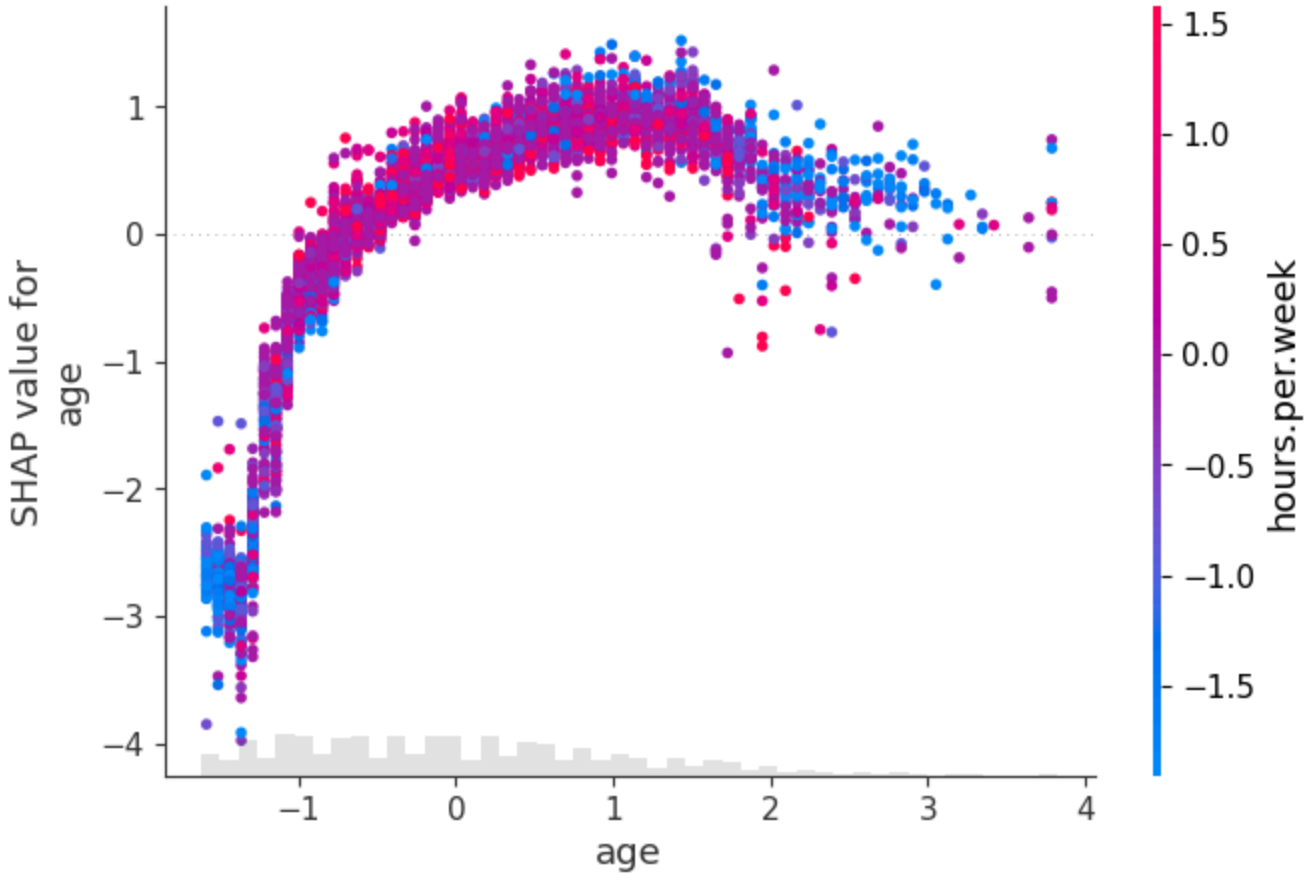
We could also color by another variable, to see interactions, e.g. how age and education impact the shap value.

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

```
array(['pipeline-1__age', 'pipeline-1__capital.gain',
       'pipeline-1__capital.loss', 'pipeline-1__hours.per.week',
       'pipeline-2__education', 'pipeline-3__sex_Male',
       'pipeline-4__workclass_Federal-gov',
       'pipeline-4__workclass_Local-gov',
       'pipeline-4__workclass_Never-worked',
       'pipeline-4__workclass_Private',
       'pipeline-4__workclass_Self-emp-inc',
       'pipeline-4__workclass_Self-emp-not-inc',
       'pipeline-4__workclass_State-gov',
       'pipeline-4__workclass_Without-pay',
       'pipeline-4__workclass_missing',
       'pipeline-4__marital.status_Divorced',
       'pipeline-4__marital.status_Married-AF-spouse',
       'pipeline-4__marital.status_Married-civ-spouse',
       'pipeline-4__marital.status_Married-spouse-absent',
       'pipeline-4__marital.status_Never-married',
       'pipeline-4__marital.status_Separated',
       'pipeline-4__marital.status_Widowed',
       'pipeline-4__occupation_Adm-clerical',
       'pipeline-4__occupation_Armed-Forces',
       'pipeline-4__occupation_Craft-repair',
       'pipeline-4__occupation_Exec-managerial',
       'pipeline-4__occupation_Farming-fishing',
       'pipeline-4__occupation_Handlers-cleaners',
       'pipeline-4__occupation_Machine-op-inspct',
       'pipeline-4__occupation_Other-service',
       'pipeline-4__occupation_Priv-house-serv',
       'pipeline-4__occupation_Prof-specialty',
       'pipeline-4__occupation_Protective-serv',
       'pipeline-4__occupation_Sales',
       'pipeline-4__occupation_Tech-support',
       'pipeline-4__occupation_Transport-moving',
       'pipeline-4__occupation_missing',
       'pipeline-4__relationship_Husband',
       'pipeline-4__relationship_Not-in-family',
       'pipeline-4__relationship_Other-relative',
       'pipeline-4__relationship_Own-child',
       'pipeline-4__relationship_Unmarried',
       'pipeline-4__relationship_Wife',
       'pipeline-4__native.country_Cambodia',
       'pipeline-4__native.country_Canada',
       'pipeline-4__native.country_China',
       'pipeline-4__native.country_Columbia',
       'pipeline-4__native.country_Cuba',
       'pipeline-4__native.country_Dominican-Republic',
       'pipeline-4__native.country_Ecuador',
       'pipeline-4__native.country_El-Salvador',
       'pipeline-4__native.country_England',
       'pipeline-4__native.country_France',
       'pipeline-4__native.country_Germany',
       'pipeline-4__native.country_Greece',
       'pipeline-4__native.country_Guatemala',
       'pipeline-4__native.country_Haiti',
       'pipeline-4__native.country_Holand-Netherlands',
```

```
        'pipeline-4__native.country_Honduras',
        'pipeline-4__native.country_Hong',
        'pipeline-4__native.country_Hungary',
        'pipeline-4__native.country_India',
        'pipeline-4__native.country_Iran',
        'pipeline-4__native.country_Ireland',
        'pipeline-4__native.country_Italy',
        'pipeline-4__native.country_Jamaica',
        'pipeline-4__native.country_Japan',
        'pipeline-4__native.country_Laos',
        'pipeline-4__native.country_Mexico',
        'pipeline-4__native.country_Nicaragua',
        'pipeline-4__native.country_Outlying-US(Guam-USVI-etc)',
        'pipeline-4__native.country_Peru',
        'pipeline-4__native.country_Philippines',
        'pipeline-4__native.country_Poland',
        'pipeline-4__native.country_Portugal',
        'pipeline-4__native.country_Puerto-Rico',
        'pipeline-4__native.country_Scotland',
        'pipeline-4__native.country_South',
        'pipeline-4__native.country_Taiwan',
        'pipeline-4__native.country_Thailand',
        'pipeline-4__native.country_Trinadad&Tobago',
        'pipeline-4__native.country_United-States',
        'pipeline-4__native.country_Vietnam',
        'pipeline-4__native.country_Yugoslavia',
        'pipeline-4__native.country_missing'], dtype=object)
```

```
shap.plots.scatter(
    lgbm_explanation[:, 'age'],
    lgbm_explanation[:, 'hours.per.week'],

)
```



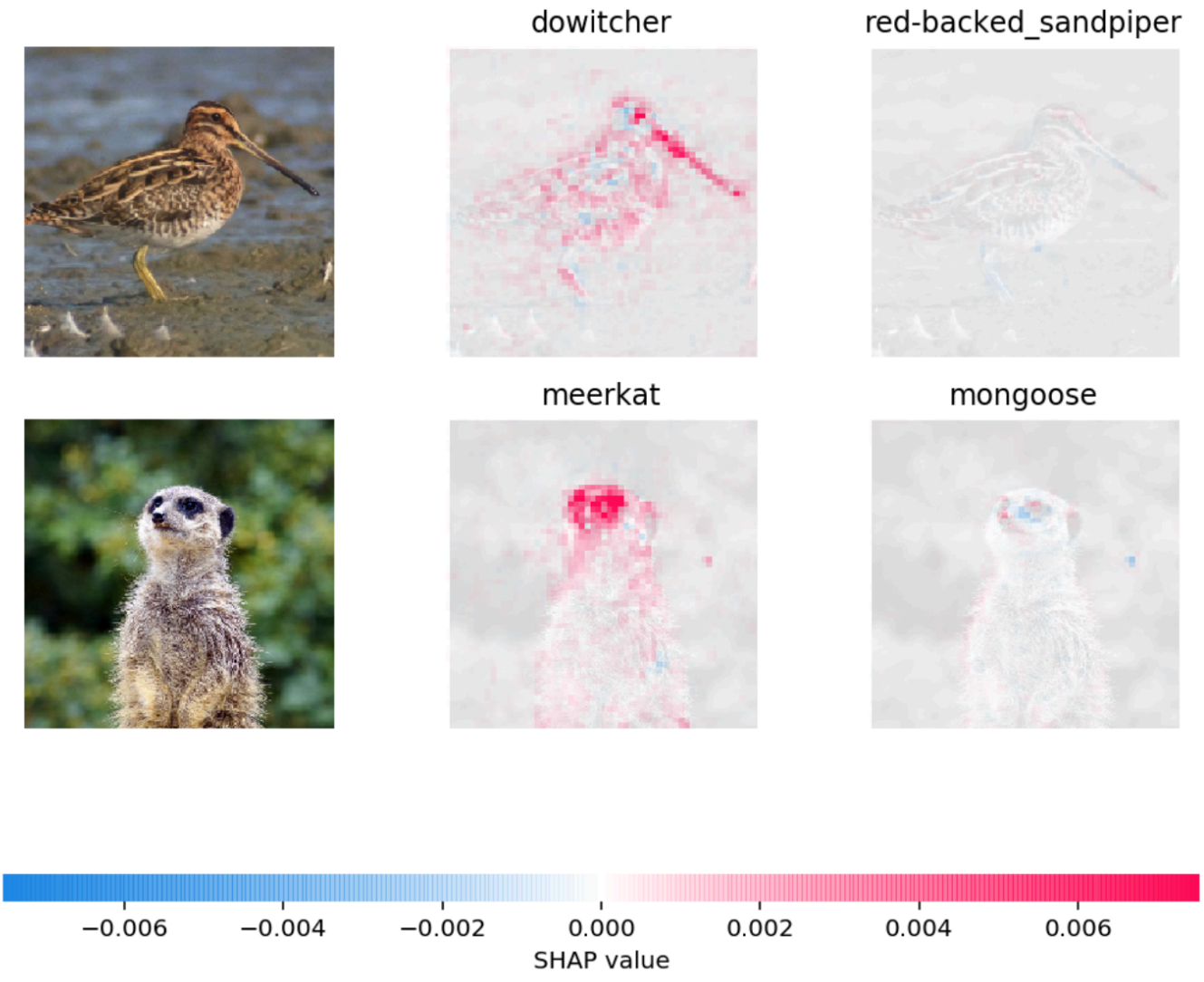
The plot above shows effect of `age` feature on the prediction.

- Each dot is a single prediction for examples above.
- The x-axis represents values of the feature age (scaled).
- The y-axis is the SHAP value for that feature, which represents how much knowing that feature's value changes the output of the model for that example's prediction.
- Lower values of age have smaller SHAP values for class "<50K".
- Similarly, higher values of age also have a bit smaller SHAP values for class ">50K", which makes sense.
- There is some optimal value of age between scaled age of 1 which gives highest SHAP values for for class ">50K".
- The color corresponds to a second feature (hours per week feature in this case) that may have an interaction effect with the feature we are plotting. It's hard to see for most of the plot but, really young and old people seem to spend less hours per week in general.

Here, we explore SHAP's TreeExplainer. It also provides explainer for different kinds of models.

- TreeExplainer (supports XGBoost, CatBoost, LightGBM)
- DeepExplainer (supports deep-learning models)
- KernelExplainer (supports kernel-based models)
- GradientExplainer (supports Keras and Tensorflow models)

- Can also be used to explain text classification and image classification
- Example: In the picture below, red pixels represent positive SHAP values that increase the probability of the class, while blue pixels represent negative SHAP values the reduce the probability of the class.



Source

Other tools

- lime is another package.

If you're not already impressed, keep in mind:

- So far we've only used sklearn models.
- Most sklearn models have some built-in measure of feature importances.
- On many tasks we need to move beyond sklearn, e.g. LightGBM, deep learning.
- These tools work on other models as well, which makes them extremely useful.

Why do we want this information?

Possible reasons:

- Identify features that are not useful and maybe remove them.
- Get guidance on what new data to collect.
  - New features related to useful features -> better results.
  - Don't bother collecting useless features -> save resources.
- Help explain why the model is making certain predictions.
  - Debugging, if the model is behaving strangely.
  - Regulatory requirements.
  - Fairness / bias. See this.
  - Keep in mind this can be used on **deployment** predictions!

Here are some guidelines and important points to remember when you work on a prediction problem where you also want to understand which features are influencing the predictions.

- Examine multicoliniarity in your dataset using methods such as VIF or correlation clustering.
- If you observe high correlations in your dataset, either get rid of redundant features or be mindful of these correlations during interpretation.
- Be mindful that feature relevance is not clearly defined. Adding/removing features can change feature importance/unimportance. Also, feature importances do not give us causal relationships. See this optional section from Lecture 4.

- Most of the models we use in ML are regularized models. With L2 regularization, the feature importances are distributed evenly among correlated features. With L1 regularization, one of the correlated features gets a high importance and the other gets a lower importance.
- Don't be overconfident. Always take feature importance values with a grain of salt.

What did we learn today?

- Why interpretation?
- Interpretation in terms of feature importances
- Interpretation beyond linear models with permutation importances
- Making sense of SHAP plots

❓❓ Questions for you

iClicker Exercise 8.1

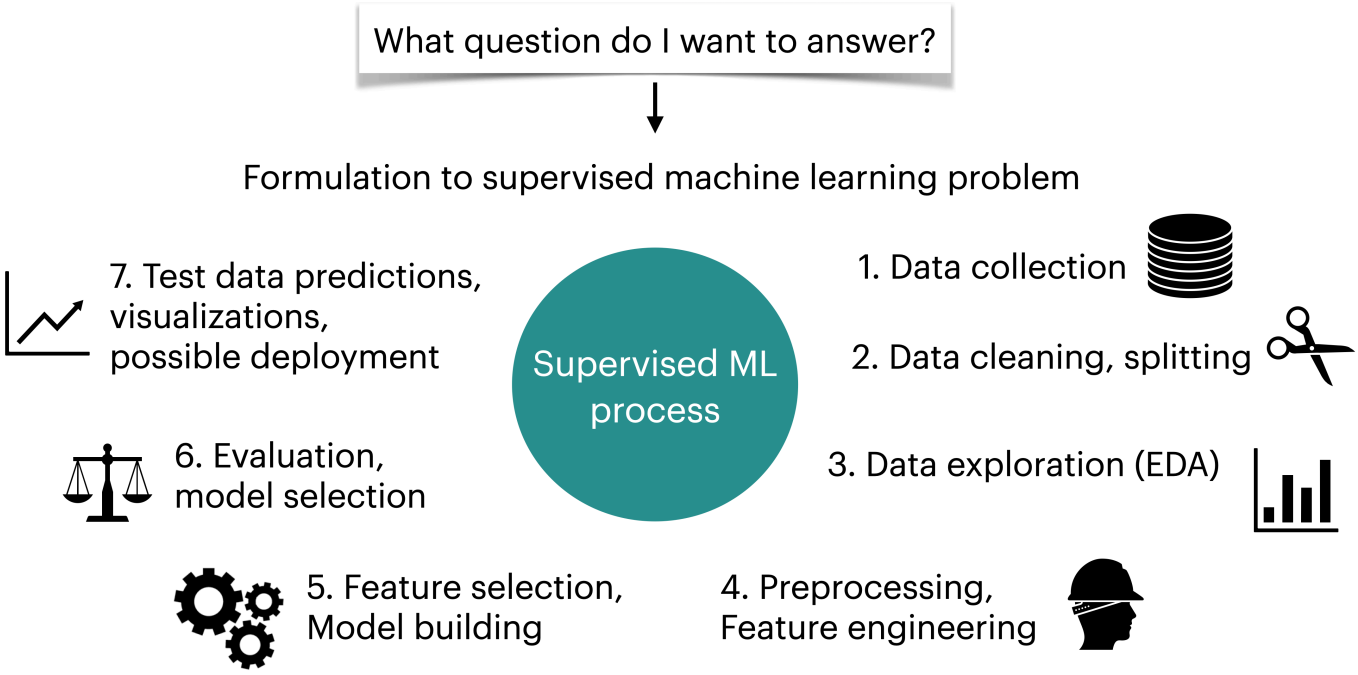
Select all the statements which are True.

- (A) You train a random forest on a binary classification problem with two classes [neg, pos]. A value of 0.580 for feat1 given by `feature_importances_` attribute of your model means that increasing the value of feat1 will drive us towards positive class.
- (B) Scikit learn's permutation importances can be used to get feature importances for non `sklearn` models.
- (C) With SHAP you can only explain predictions on the training examples.

▶ Click to see the solution

- In what scenarios interpretability can be harmful?

Final comments



If you're curious about deployment, here are my notes from CPSC 330 on this topic.

Some key takeaways

Some useful guidelines:

- Do train-test split right away and only once
- Don't look at the test set until the end
- Don't call `fit` on test/validation data
- Use pipelines
- Use baselines
- Do not be overconfident about your models. They do not tell us how the world works.

Difference between Statistics and Machine Learning

- There is no clear boundary.
- But loosely
  - in statistics the emphasis is on inference (usually framed as hypothesis testing)

- in machine learning the emphasis is on prediction and generalization. We are primarily interested in using our model on new unseen examples.

Recipe to approach a supervised learning problem with tabular data

Understanding the problem

1. Have a long conversation with the stakeholder(s) who will be using your pipeline.
2. Have a long conversation with the person(s) who collected the data.
3. Think about
    - what do you want to accomplish?
    - whether you really need to use machine learning for the problem or not
    - how do you want to frame the problem as a supervised machine learning problem? What do you want to predict?
    - how do you plan to measure your success?
    - is there a baseline?
    - is there any operating point?
    - what will it buy the stakeholders if you improve over their current baseline?
4. Think about the ethical implications - are you sure you want to do this project? If so, should ethics guide your approach?

Initial analysis, EDA, preprocessing

1. Random train-test split with fixed random seed; do not touch the test data until Step 16.
2. Exploratory data analysis, outlier detection.
3. Choose a scoring metric -> higher values should make you & your stakeholders happier.
4. Feature engineering –> Create new features which might be useful to solve the prediction problem. This is typically a time-consuming process. Also, sometimes you get an idea of good features only after looking at your model results. So it's an iterative process.
5. Fit a baseline model, e.g. `DummyClassifier` or `DummyRegressor`.
6. Create a preprocessing pipeline.

7. (Optional) Incorporate feature selection.

Model building

1. Try a linear model, e.g. `LogisticRegression` for classification or `Ridge`; tune hyperparameters with CV.
2. Try other sensible model(s), e.g. LightGBM; tune hyperparameters with CV.
3. For each model, look at sub-scores from the folds of cross-validation to get a sense of "error bars" on the scores.
4. Pick a few reasonable models. Best CV score is a reasonable metric, though you may choose to favour simpler models.
5. Carry out hyperparameter optimization for these models, paying attention to whether you are not susceptible to optimization bias.
6. Try averaging and stacking to examine whether you improve the results over your best-performing models.

Model transparency and interpretation

1. Look at feature importances.
2. (optional) Perform some more diagnostics like confusion matrix for classification, or "predicted vs. true" scatterplots for regression.
3. (optional) Try to calibrate the uncertainty/confidence outputted by your model.
4. Test set evaluation.
5. Model transparency. Take a random sample from the test data and try to explain your predictions using SHAP plots.
6. Question everything again: validity of results, bias/fairness of trained model, etc.
7. Concisely summarize your results and discuss them with stakeholders.
8. (optional) Retrain on all your data.
9. Deployment & integration.

PS: the order of steps is approximate, and some steps may need to be repeated during prototyping, experimentation, and as needed over time.

Coming up

- We haven't talked about deep learning and working with images yet. It's coming up in block 5 (DSCI 572).
- Supervised learning is quite successful but many well-known people in the field believe that it's fundamentally limited, as it doesn't seem to be how humans learn. We'll learn about unsupervised learning in block 5 (DSCI 563).
- Forecasting and time series is another important area in machine learning. You'll be learning about it in block 5 (DSCI 574)
- Finally, we'll talk about sequential models for text data in block 6 (DSCI 575)

Farewell

That's all, folks. We made it! Good luck with the quiz next week. Happy holidays and I'll see you in block 5!