Mud card

- **Why would you ever want to train an XGboost reduced-features model if XGboost is good at handling missing values on its own?**
    - I use it in class so you see that the plain XGB test score and the reduced-features XGB test score are pretty similar.
    - You might want to compare how the XGB reduced-features test scores compare to e.g., SVM reduced-features or RF reduced-features.
    - But generally speaking, there is no need to use reduced-features with XGB
- **What are the pros and cons of using a different machine learning algorithms for the reduced-features submodel that has no missing values? In today's lecture it was pointed out that a different function than XGBoost should be used in the reduced_feature_xgb function in the lecture notes. Are there shortcomings in using XGboost here? Or it is a matter of the choice of model**
    - If your dataset has no missing values, there is no need to use reduced-features.
    - If your dataset has missing values, use reduced features and modify the code such that it works with other ML models (like the linear models, SVM, RF, k-nearest neighbor).
- **Why should the XGBoost model perform roughly the same when we just use XGBoost versus when we do the reduced-feature model?**
    - This is not a general conclusion.
    - XGB and reduced-feature-XGB happens to perform similarly on the kaggle hours price dataset but that does not mean that the performance will be similar on any and all dataset!
    - What this indicates is that maybe the missingness in the house price dataset doesn't correlate with the target variable.
    - Again, this is a conclusion specific to this one dataset!
- **Why can't we run sklearn models on gpus? Is there a particular barrier, or are they just not optimized yet?**
    - It is a significant amount of work to write GPU-friendly code and the sklearn developers decided not to do so.
    - the work is warranted for deep learning packages because most of the computations are performed on GPUs.
    - It's less beneficial to run sklearn on GPUs because many of the models and techniques in sklearn are not inherently parallelizable.

Global feature importance metrics

By the end of this module, you will be able to

- perform permutation feature importance calculation
- study the coefficients of linear models
- outlook to other metrics

The supervised ML pipeline

The goal: Use the training data (X and y) to develop a model which can accurately predict the target variable (y_new') for previously unseen data (X_new).

1. Exploratory Data Analysis (EDA): you need to understand your data and verify that it doesn't contain errors

- do as much EDA as you can!

2. Split the data into different sets: most often the sets are train, validation, and test (or holdout)

- practitioners often make errors in this step!
- you can split the data randomly, based on groups, based on time, or any other non-standard way if necessary to answer your ML question

3. Preprocess the data: ML models only work if X and Y are numbers! Some ML models additionally require each feature to have 0 mean and 1 standard deviation (standardized features)

- often the original features you get contain strings (for example a gender feature would contain 'male', 'female', 'non-binary', 'unknown') which needs to transformed into numbers
- often the features are not standardized (e.g., age is between 0 and 100) but it needs to be standardized

4. Choose an evaluation metric: depends on the priorities of the stakeholders

- often requires quite a bit of thinking and ethical considerations

5. Choose one or more ML techniques: it is highly recommended that you try multiple models

- start with simple models like linear or logistic regression
- try also more complex models like nearest neighbors, support vector machines, random forest, etc.

6. Tune the hyperparameters of your ML models (aka cross-validation)

- ML techniques have hyperparameters that you need to optimize to achieve best performance
- for each ML model, decide which parameters to tune and what values to try
- loop through each parameter combination
  - train one model for each parameter combination
  - evaluate how well the model performs on the validation set
- take the parameter combo that gives the best validation score
- evaluate that model on the test set to report how well the model is expected to perform on previously unseen data

7. Interpret your model: black boxes are often not useful

- check if your model uses features that make sense (excellent tool for debugging)
- often model predictions are not enough, you need to be able to explain how the model arrived to a particular prediction (e.g., in health care)

Motivation

- debugging ML models is tough
  - a model that runs without errors/warning is not necessarily correct
- how do you know that your model is correct?
  - check test set predictions
    - in regression: check points with a large difference between true and predicted values
    - in classification: confusion matrix, check out FPs and FNs
  - inspect your model
    - especially useful for non-linear models
    - metrics to measure how much a model depends on a feature is one way to inspect your model

Global feature importance metrics

By the end of this module, you will be able to

- **perform permutation feature importance calculation**
- study coefficients of linear models
- outlook to other metrics

Permutation feature importance

- model agnostic, you can use it with any supervised ML model
- steps:
  - train a model and calculate a test score :)
  - randomly shuffle a single feature in the test set
  - recalculate the test score with the shuffled data
  - model score worsens because the shuffling breaks the relationship between feature and target
  - the larger the difference, the more important the feature is

In [1]:

```
import numpy as np
import pandas as pd
from sklearn.svm import SVC
from sklearn.ensemble import RandomForestClassifier
from sklearn.pipeline import make_pipeline
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import train_test_split
from sklearn.model_selection import StratifiedKFold
from sklearn.preprocessing import StandardScaler
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import OneHotEncoder
import matplotlib.pylab as plt

df = pd.read_csv('data/adult_data.csv')
label = 'gross-income'
y = df[label]
df.drop(columns=[label],inplace=True)
X = df
ftr_names = X.columns
print(X.head())
print(y)
```

```
   age          workclass  fnlwgt   education  education-num  \
0   39          State-gov   77516   Bachelors             13
1   50   Self-emp-not-inc   83311   Bachelors             13
2   38            Private  215646     HS-grad              9
3   53            Private  234721        11th              7
4   28            Private  338409   Bachelors             13

       marital-status         occupation    relationship   race      sex  \
0       Never-married       Adm-clerical   Not-in-family  White     Male
1  Married-civ-spouse    Exec-managerial         Husband  White     Male
2            Divorced  Handlers-cleaners   Not-in-family  White     Male
3  Married-civ-spouse  Handlers-cleaners         Husband  Black     Male
4  Married-civ-spouse     Prof-specialty            Wife  Black   Female

   capital-gain  capital-loss  hours-per-week  native-country
0          2174             0              40   United-States
1             0             0              13   United-States
2             0             0              40   United-States
3             0             0              40   United-States
4             0             0              40            Cuba
0         <=50K
1         <=50K
2         <=50K
3         <=50K
4         <=50K
          ...
32556     <=50K
32557      >50K
32558     <=50K
32559     <=50K
32560      >50K
Name: gross-income, Length: 32561, dtype: object
```

In [2]:

```
def ML_pipeline_kfold(X,y,random_state,n_folds):
    # create a test set
    X_other, X_test, y_other, y_test = train_test_split(X, y, test_size=0.2, random_state = random_state)
    # splitter for _other
    kf = StratifiedKFold(n_splits=n_folds,shuffle=True,random_state=random_state)
    # create the pipeline: preprocessor + supervised ML method
    cat_ftrs = ['workclass','education','marital-status','occupation','relationship','race','sex','native-country']
    cont_ftrs = ['age','fnlwgt','education-num','capital-gain','capital-loss','hours-per-week']
    # one-hot encoder
    categorical_transformer = Pipeline(steps=[
        ('onehot', OneHotEncoder(sparse_output=False,handle_unknown='ignore'))])
    # standard scaler
    numeric_transformer = Pipeline(steps=[
        ('scaler', StandardScaler())])
    preprocessor = ColumnTransformer(
        transformers=[
            ('num', numeric_transformer, cont_ftrs),
            ('cat', categorical_transformer, cat_ftrs)])
    pipe = make_pipeline(preprocessor,SVC())
    #pipe = make_pipeline(preprocessor,RandomForestClassifier())
    # the parameter(s) we want to tune
    param_grid = {'svc__C': [0.01, 0.1, 1, 10, 100],
                  'svc__gamma': [0.01, 0.1, 1, 10, 100]}
    # param_grid = {
    #               'randomforestclassifier__max_depth': [1, 3, 10, 30, 100], # no upper bound so the values are eve
    #               'randomforestclassifier__max_features': [0.25, 0.5,0.75,1.0] # linearly spaced because it is bet
    #               }

    # prepare gridsearch
    grid = GridSearchCV(pipe, param_grid=param_grid,cv=kf, return_train_score = True,n_jobs=-1,verbose=True)
    # do kfold CV on _other
    grid.fit(X_other, y_other)
    return grid, X_test, y_test
```

Be careful, SVM is used on a relatively large dataset

In [3]:

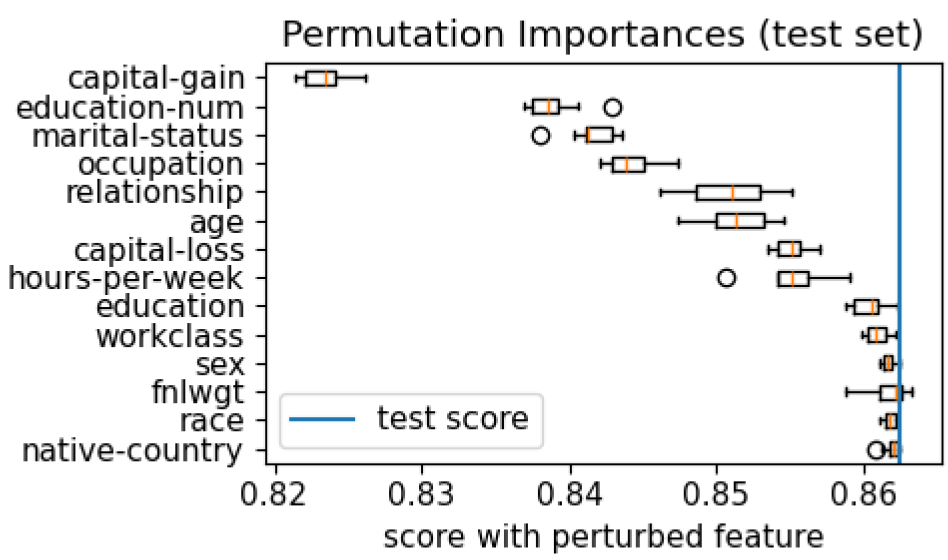
```
model, X_test, y_test = ML_pipeline_kfold(X,y,42,4)
print(model.best_score_)
print(model.score(X_test,y_test))
print(model.best_params_)

# save the output so I can use it later
import pickle
file = open('results/grid.save', 'wb')
pickle.dump((model,X_test,y_test),file)
file.close()
```

```
Fitting 4 folds for each of 25 candidates, totalling 100 fits
0.8545377764127764
0.8624289881774911
{'svc__C': 1, 'svc__gamma': 0.1}
```

In [4]:
```python
import pickle
file = open('results/grid.save', 'rb')
model, X_test, y_test = pickle.load(file)
file.close()

np.random.seed(42)

nr_runs = 10
scores = np.zeros([len(ftr_names),nr_runs])

test_score = model.score(X_test,y_test)
print('test score = ',test_score)
print('test baseline = ',np.sum(y_test == ' <=50K')/len(y_test))
# loop through the features
for i in range(len(ftr_names)):
    print('shuffling '+str(ftr_names[i]))
    acc_scores = []
    for j in range(nr_runs):
        X_test_shuffled = X_test.copy()
        X_test_shuffled[ftr_names[i]] = np.random.permutation(X_test[ftr_names[i]].values)
        acc_scores.append(model.score(X_test_shuffled,y_test))
    print('   shuffled test score:',np.around(np.mean(acc_scores),3),'+/-',np.around(np.std(acc_scores),3))
    scores[i] = acc_scores
```

```
test score =  0.8624289881774911
test baseline =  0.7587901120835252
shuffling age
   shuffled test score: 0.851 +/- 0.002
shuffling workclass
   shuffled test score: 0.861 +/- 0.001
shuffling fnlwgt
   shuffled test score: 0.862 +/- 0.001
shuffling education
   shuffled test score: 0.86 +/- 0.001
shuffling education-num
   shuffled test score: 0.839 +/- 0.002
shuffling marital-status
   shuffled test score: 0.842 +/- 0.002
shuffling occupation
   shuffled test score: 0.844 +/- 0.002
shuffling relationship
   shuffled test score: 0.851 +/- 0.003
shuffling race
   shuffled test score: 0.862 +/- 0.0
shuffling sex
   shuffled test score: 0.862 +/- 0.0
shuffling capital-gain
   shuffled test score: 0.823 +/- 0.001
shuffling capital-loss
   shuffled test score: 0.855 +/- 0.001
shuffling hours-per-week
   shuffled test score: 0.855 +/- 0.002
shuffling native-country
   shuffled test score: 0.862 +/- 0.001
```

In [10]:
```python
sorted_indcs = np.argsort(np.mean(scores,axis=1))[::-1]
plt.rcParams.update({'font.size': 11})
plt.figure(figsize=(5,3))
plt.boxplot(scores[sorted_indcs].T,tick_labels=ftr_names[sorted_indcs],vert=False)
plt.axvline(test_score,label='test score')
plt.title("Permutation Importances (test set)")
plt.xlabel('score with perturbed feature')
plt.legend()
plt.tight_layout()
plt.show()
```



Check out sklearn's permutation importance!

https://scikit-learn.org/stable/modules/generated/sklearn.inspection.permutation_importance.html

https://scikit-learn.org/stable/modules/permutation_importance.html#permutation-importance

Cons of permutation feature importance

- strongly correlated features
    - if one of the features is shuffled, the model can still use the other correlated feature
    - both features appear to be less important than they actually are
    - solution:
        - check the correlation matrix plot
        - remove all but one of the strongly correlated features
- no feature interactions
    - one feature might appear unimportant but combined with another feature could be important
    - solution:
        - permute two features to measure how important feature pairs are
        - this can be computationally expensive

Quiz

Global feature importance metrics

By the end of this module, you will be able to

- perform permutation feature importance calculation
- **study the coefficients of linear models**
- outlook to other metrics

Coefficients of linear models

- the coefficients of linear and logistic regression can be used as a measure of feature importance **ONLY IF** all features have the same mean (usually 0) and the same standard deviation (usually 1)
    - all features meaning that the one-hot encoded and ordinal features as well!
- then the absolute value of the coefficients can be used to rank them

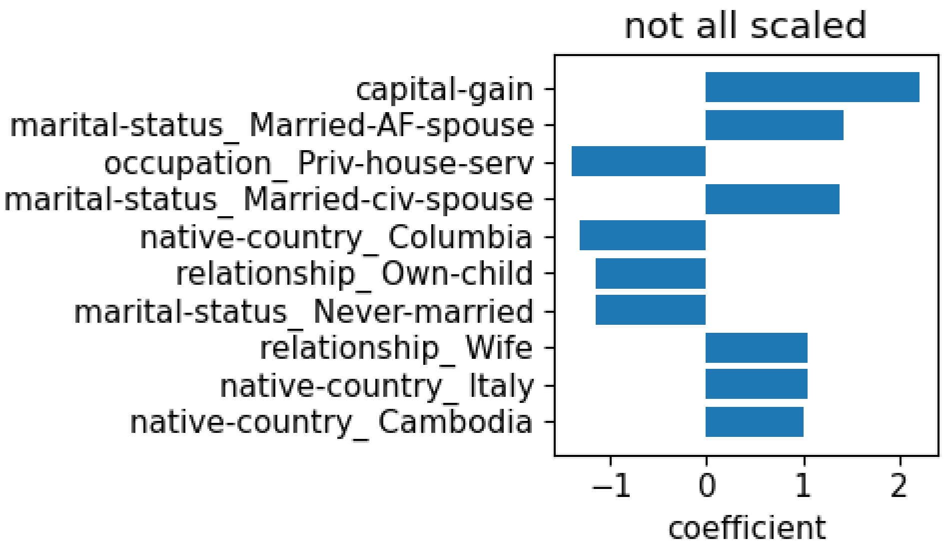
Let's rewrite the kfold CV function a bit

In [6]:
```
from sklearn.linear_model import LogisticRegression
def ML_pipeline_kfold_LR1(X,y,random_state,n_folds):
    # create a test set
    X_other, X_test, y_other, y_test = train_test_split(X, y, test_size=0.2, random_state = random_state)
    # splitter for _other
    kf = StratifiedKFold(n_splits=n_folds,shuffle=True,random_state=random_state)
    # create the pipeline: preprocessor + supervised ML method
    cat_ftrs = ['workclass','education','marital-status','occupation','relationship','race','sex','native-country']
    cont_ftrs = ['age','fnlwgt','education-num','capital-gain','capital-loss','hours-per-week']
    # one-hot encoder
    categorical_transformer = Pipeline(steps=[
        ('onehot', OneHotEncoder(sparse_output=False,handle_unknown='ignore'))])
    # standard scaler
    numeric_transformer = Pipeline(steps=[
        ('scaler', StandardScaler())])
    preprocessor = ColumnTransformer(
        transformers=[
            ('num', numeric_transformer, cont_ftrs),
            ('cat', categorical_transformer, cat_ftrs)])
    pipe = make_pipeline(preprocessor,LogisticRegression(penalty='l2',solver='lbfgs',max_iter=10000000))
    # the parameter(s) we want to tune
    param_grid = {'logisticregression__C': [0.01, 0.1, 1, 10,100]}
    # prepare gridsearch
    grid = GridSearchCV(pipe, param_grid=param_grid,cv=kf, return_train_score = True,n_jobs=-1)
    # do kfold CV on _other
    grid.fit(X_other, y_other)
    feature_names = cont_ftrs + \
                list(grid.best_estimator_[0].named_transformers_['cat'][0].get_feature_names_out(cat_ftrs))
    return grid, np.array(feature_names), X_test, y_test
```

In [7]:
```
grid, feature_names, X_test, y_test = ML_pipeline_kfold_LR1(X,y,42,4)
print('test score:',grid.score(X_test,y_test))
coefs = grid.best_estimator_[-1].coef_[0]
sorted_indcs = np.argsort(np.abs(coefs))

plt.figure(figsize=(5,3))
```

```python
plt.rcParams.update({'font.size': 11})
plt.barh(np.arange(10),coefs[sorted_indcs[-10:]])
plt.yticks(np.arange(10),feature_names[sorted_indcs[-10:]])
plt.xlabel('coefficient')
plt.title('not all scaled')
plt.tight_layout()
plt.savefig('figures/LR_coefs_notscaled.png',dpi=300)
plt.show()
```

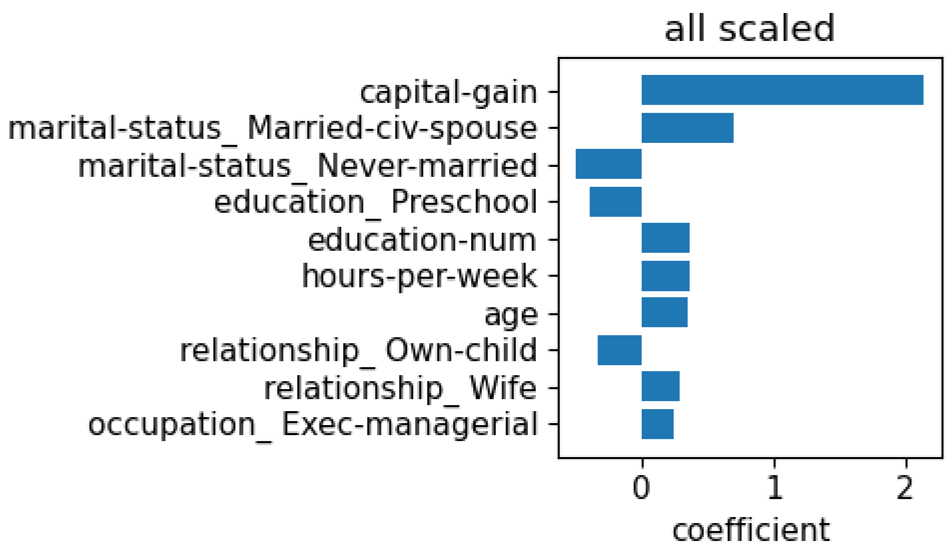
```
test score: 0.8581298940580377
```

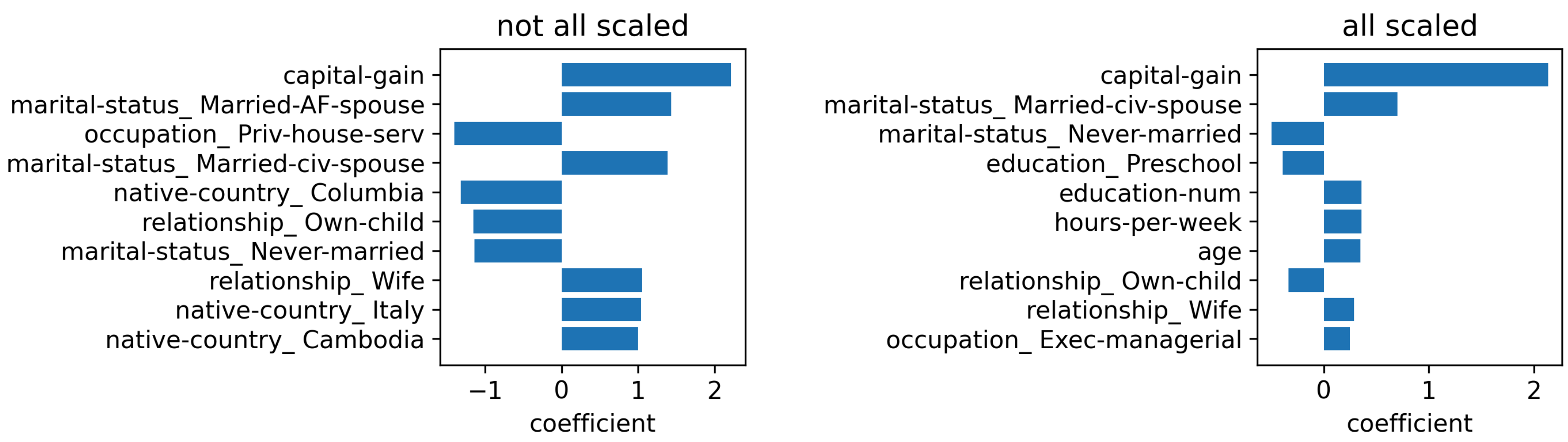


In [8]:
```python
from sklearn.linear_model import LogisticRegression
def ML_pipeline_kfold_LR2(X,y,random_state,n_folds):
    # create a test set
    X_other, X_test, y_other, y_test = train_test_split(X, y, test_size=0.2, random_state = random_state)
    # splitter for _other
    kf = StratifiedKFold(n_splits=n_folds,shuffle=True,random_state=random_state)
    # create the pipeline: preprocessor + supervised ML method
    cat_ftrs = ['workclass','education','marital-status','occupation','relationship','race','sex','native-country']
    cont_ftrs = ['age','fnlwgt','education-num','capital-gain','capital-loss','hours-per-week']
    # one-hot encoder
    categorical_transformer = Pipeline(steps=[
        ('onehot', OneHotEncoder(sparse_output=False,handle_unknown='ignore'))])
    # standard scaler
    numeric_transformer = Pipeline(steps=[
        ('scaler', StandardScaler())])
    preprocessor = ColumnTransformer(
        transformers=[
            ('num', numeric_transformer, cont_ftrs),
            ('cat', categorical_transformer, cat_ftrs)])
    final_scaler = StandardScaler()
    pipe = make_pipeline(preprocessor,final_scaler,LogisticRegression(penalty='l2',solver='lbfgs'))
    # the parameter(s) we want to tune
    param_grid = {'logisticregression__C': [0.01, 0.1, 1, 10,100]}
    # prepare gridsearch
    grid = GridSearchCV(pipe, param_grid=param_grid,cv=kf, return_train_score = True,n_jobs=-1)
    # do kfold CV on _other
    grid.fit(X_other, y_other)
    feature_names = cont_ftrs + \
                list(grid.best_estimator_[0].named_transformers_['cat'][0].get_feature_names_out(cat_ftrs))

    return grid, np.array(feature_names), X_test, y_test
```

In [9]:
```python
grid, feature_names, X_test, y_test = ML_pipeline_kfold_LR2(X,y,42,4)
print('test score:',grid.score(X_test,y_test))
coefs = grid.best_estimator_[-1].coef_[0]
sorted_indcs = np.argsort(np.abs(coefs))

plt.figure(figsize=(5,3))
plt.rcParams.update({'font.size': 11})
plt.barh(np.arange(10),coefs[sorted_indcs[-10:]])
plt.yticks(np.arange(10),feature_names[sorted_indcs[-10:]])
plt.xlabel('coefficient')
plt.title('all scaled')
plt.tight_layout()
plt.savefig('figures/LR_coefs_scaled.png',dpi=300)
plt.show()
```

```
test score: 0.857976354982343
```





Global feature importance metrics

By the end of this module, you will be able to

- perform permutation feature importance calculation
- study the coefficients of linear models
- **outlook to other metrics**

- SVM:
  - SVC.coef_ and SVR.coef_ can be used as a metric of feature importance if **all** features are standardized
  - for linear SVMs only!
- random forest:
  - RandomForestRegressor.feature_importances_ and RandomForestClassification.feature_importances_
  - gini importance or mean decrease impurity, see here and here
- XGBoost:
  - five different metrics are implemented, see here and here

Quiz

Mudcard

In [ ]:
```

```