1. ML - Basics

-> import dat

-> analýza dat → data.describe(), data.info()
→ data.head()
→ data.variable.value_counts(dropna = FALSE)

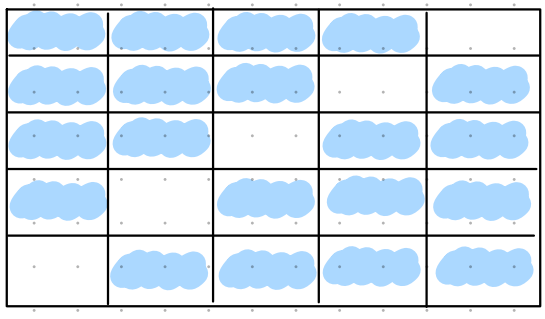
- označit target (jaký to je sloupec v datech)

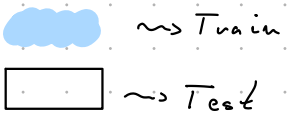
→ Definujeme prediktory → numerical
→ cathegorical

- Data split
  - ↳ training sample, validation sample, test sample, (out of time sample, historical out of time sample)
    - No stratification
      (in time mask)
      data_train, data_test = train_test_split(data[intime_mask], test = 0.4, random_state=2)
    - Stratification
      v každém samplu jsou proporčně podobně zastoupené proměnné jako v originálním datasetu (distribuce se zachová)
      ↳ train_test_split(..., stratify = (variable))
    - masky = train / valid / test
      valid = na vylepšení hyperparametrů (prevent overfitting), learning rate, regularization, ...
      vyhodnocuje se na nich ("batch propagation")
      na validačním setu získáváme nové info

- Cross-Validation
  - k-fold crossvalidation:
    -> použijeme #data na train i valid
    př.: kf = KFold(n_splits = 5), kf.get_n_splits(data)

iterace: 1. 2. 3. 4. 5.





- Data Cleaning
  -> jednoduché jestli tam nejsou nějaké nekonečné hodnoty a kdyžtak nahradit
  -> podezřelá pozorování: např první transakce dřív než poslední
  zkusit lépe porozumět datům

- Missing Data
  - např. v měsících je 0 a nahradíme průměrem
  - v kódu graf jež zobrazí proměnné a zastoupení jejich chybějících hodnot v různých měsících.
    - vypište vždy podíl chybějících hodnot (0 až 1)

- Variable Selection
  -> cols_pred → numerical
  → cathegorical, zadáme jaké prediktory nepoužijeme

- Feature engineering
  Automatic grouping
  -> only training set !
  -> většinou podle kvantilů
  -> add missing as a new variable

- Encode cathegorical predictors
  ! mean target encoding: 1) Pro každou kategorii spočítáme průměrnou hodnotu targetu v dané kategorii, v binární počítáme proporce pro jednotlivé kategorie
  2) výměna cathegory → mean target
  3) Vyřešit proměnné nenacházející se v testu
  -> nahradit celkovým průměrem

- **Logistic regression**
  - -> fit model
  - -> predict probability
  - -> calculate gini for different samples
    - ↳ impurity function = měříme pst. nesprávné klasifikace náhodně vybraného prvku
      - $\in (0, 0.5)$
        - 0 = pure dataset všechna data 1 kategorie
        - 0.5 = podmnožina rozdělena mezi dvěma třídami

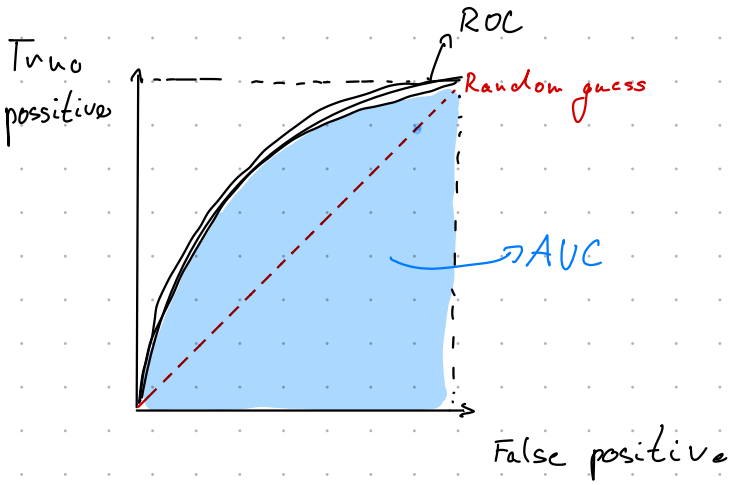
- **Penalty**
  - -> regularization constant
  - -> LASSO coef. jako regularizační konstanta = L1 regularizace
    - ↳ $\min (\|y - X\beta\|_2^2 + \lambda \|\beta\|_1)$

- **Performance characteristics**
  - **ROC curve**

True positive rate vs. False positive rate

" "

$\frac{TP}{TP+FN}$ $\frac{FP}{FP+TN}$



AUC → blízko 1 ideální
→ blízko 0.5 stejné jako random guess

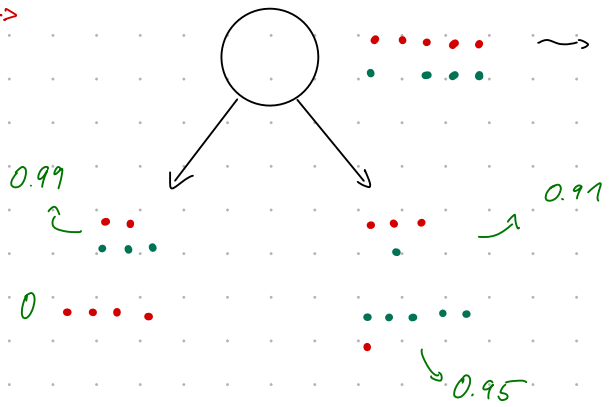
ROC pro binary classifiers obsahuje více treshholdů

2. Trees and Forest

- Load and adjust data
  - DecisionTreeRegression (continuous target - věk, výška, teplota, ...)
  - DecisiosTreeClassification (cathegorical target - barva, 0/1, ...)

CARD

- In each node we iterate over randomly chosen set of predictors and compute gain for each predictors measured using defined measure and assuming different possible splits over this predictor. Using the best predictor we perform split into two subtrees.



→ want to maximize information gain

.. first split

.. second split

⇓

IG = information gain

$IG_1 = 1 - \frac{5}{10} \cdot 0,99 - \frac{5}{10} \cdot 0,91 = \frac{1}{20}$

$IG_2 = 1 - \frac{4}{10} \cdot 0 - \frac{6}{10} \cdot 0,95 = \frac{43}{100}$

$IG_2 > IG_1$ ... we choose G_2

- recode variables Yes/No ⟶ 1/0
  - → missing values fix
- target columns
- predictor columns
  - → check missing values: data.isna().sum()

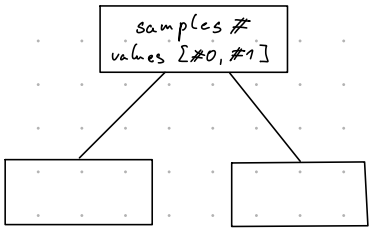
    smažeme všechny řádky, kde je target hodnota missing a

    a u ostatních je nahradíme průměrnou hodnotou

- Split and train

→ choose stratification ... volíme náš TARGET

→ fit model .. DecisionTreeClassifier → check the parameters interpretation "link"



→ predict probability ... pravděpodobnosti umíme převést na 0/1

- u accuracy (jak $p_i > 0,5$ / $p_i < 0,5$)

→ accuracy : 1 - abs(target - prediction) — 1 trefili jsme se; 0 netrefili jsme se

→ check predictors importance and DO PLOTS

CHAID

Short description:

Continuous predictors are assumed to be categorized either in 'nominal' or 'ordinal' (e.g. deciles) format. Target is either 'nominal' or 'continuous'.

We define constants alpha_merge and split_threshold.

In each node we iterate over all categories of all predictors and test if they significantly influence the target value in terms of Chi-square test if target is 'categorical' or F-test if target is continuous. Categories are merged until the significance is below alpha_merge. Now we iterate over such newly categorized predictors and split using the one with the most significant influence on target (using the same measure as above) if split_threshold is triggered by any of them. We obtain possibly split which is non-binary.

Additional parameters:

- the above procedure is repeated only until the depth of the tree is not larger then max_depth
- if we have less then min_parent_node_size in given node, no splitting is performed
- if splitting would result in new node with less then min_child_node_size observations, no splitting is performed
- if is_exhaustive is true then all possible merging combinations of given predictor's categories are examined (according to https://www.researchgate.net/publication/332591728_Classification_of_air_traffic_control_scenarios_using_decision_trees_insights_from_a_field_stu note that the significance is adjusted using Bonferroni method)

→ No need to fix missing values (jen smazat chybějící u target)

→ recode Yes/No → 1/0

→ target and predictors — categorical / continuous

↳ do categorization if continuous

→ fit model ... need to know if it is "ordinal" or "nominal"

↳ choose parameters

→ prediction ... test data need to be categorized in same way as train data

↳ need to be specified ... method is not involved

- Random Forest - manually
  - → multiple random decision trees
  - → distinct random seed
    - ↳ in bonus task 7
  - → if TARGET is missing we drop the values
  - → Predictor missing values have to be handled .. e.g. missing as new category

1. One-hot encoding → pro každou kategoriální proměnnou vytvoří sloupec a značí 0/1
   - ↳ for categorical predictors

př.:

	Col_red	Col_blue	Col_green
Red	1	0	0
Green	0	0	1
Blue	0	1	0
Red	1	0	0

⊕
- provides numerical format
- doesn't assume ordinal relationships between categories = suitable for nominal data

⊖
- lead to high dimensionality
- sparsity ("řídkost") = resulting binary vectors contain many zeros (ineffective in terms of storage)

1. Mean target encoding → nahrazuje kategorickou proměnnou průměrnou hodnotou cílové proměnné pro danou kategorii

Klient	Region	Default	Mean Target Encoded Region
1	North	0	0.333
2	South	1	0.667
3	North	0	0.333
4	West	1	0.5
5	South	0	0.667
6	North	1	0.333
7	West	0	0.5
8	South	1	0.667

North: $(0+0+1)/3 = 0.333$

South: $(1+0+1)/3 = 0.667$

West: $(1+0)/2 = 0.5$

→ train separately on different encoding ... "sklearn" = documentation

→ Evaluate on
- MAE (mean absolute error) $= \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i|$
- MAPE (mean absolute precentage error) $= \frac{100}{n}\sum_{i=1}^{n}\left|\frac{y_i - \hat{y}_i}{y_i}\right|$

↳ look at importance

3. XGBoost

-> each new tree corrects the errors of previous one

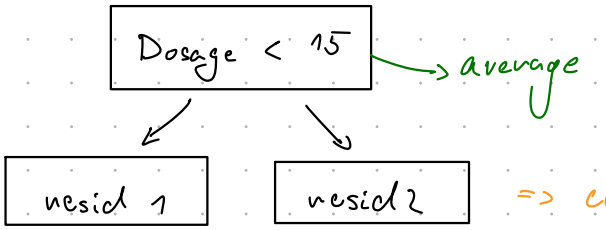
-> we need to set initial prediction

1. start with leaf [all residuals] => modelujemy residua

similarity score $= \frac{\sum \text{residuals}^2}{\text{Numbers of res} + \alpha}$ -> regularization parameter

↳ reduce prediction sensitivity

↳ prevents overfitting



=> count similarity score of each node

↳ Gain = $\text{Left}_{\text{similarity}} + \text{Right}_{\text{sim}} - \text{Root}_{\text{sim}}$

=> it is for trash hold Dosage < 15

-> calculate for new split (Dosage < 22,5), ...

Want to maximize GAIN

2. spliting gain the nodes to get Max gain = Max similarity in split

-> Gain - γ < 0 => remove branch; if we don't remove branch

↳ choosen we remove root

-> learning rate default = 0.3

Procedure

- -> load data
- -> define Target and predictors
  - numerical
  - categorical
- -> Data split
  - test
  - train
  - valid

  } Define masks
- -> Data processing:
  - replace infinity
  - Encode categorical predictors (one-hot, mean-target)
- -> XGBoost model:
  - Early stopping
    - if we don't improve in n-iteration: STOP
- -> Plot AUC
- -> Predictors importance (plot) by total gain
- -> Final model ... selection of optimal predictors before fitting
  - ↳ cross-validation score (xgb.train, xgb.XGBClassifier)
- -> Model Validation
- -> Permutation importance →

Permutation importance is a technique used to measure the importance of features in a predictive model. It quantifies the impact of each feature on the model's performance by observing the change in performance when the feature's values are randomly shuffled. This method provides an insight into how much a model depends on each feature.

How Permutation Importance Works

1. **Train the Model**: Train the model using the original dataset.
2. **Measure Baseline Performance**: Evaluate the model's performance on a validation set using a chosen metric (e.g., accuracy, mean squared error, etc.) to get a baseline score.
3. **Permute Feature Values**: For each feature, permute (shuffle) its values while keeping the values of other features unchanged. This breaks the relationship between the feature and the target variable.
4. **Evaluate Permuted Performance**: Re-evaluate the model's performance on the validation set using the permuted dataset.
5. **Calculate Importance**: Compute the importance of the feature by measuring the decrease in performance from the baseline. A larger decrease indicates higher importance.

Interpretation

- **High Importance**: If permuting a feature's values significantly decreases the model's performance, the feature is considered important.
- **Low Importance**: If permuting a feature's values does not affect the model's performance much, the feature is considered less important.

- -> Plots
  - Partial dependence
  - Individual conditional expectation
  - SHAP
    - helps interpret model predictions

1. NN_tensenflow_intro

Optimizer Adam

Regularizace L_1, L_2

- Simple NN classification

→ models with tf.keras

models: sequential, functional, model_subclassing

↳ most used

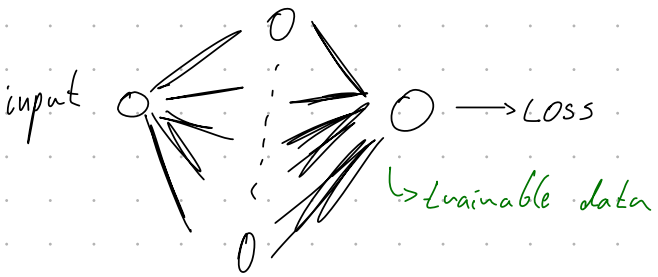
- sequential model

→ multiple inputs/outputs; non-linear topology

```
# Specification A)

model = tf.keras.Sequential([
    tf.keras.layers.InputLayer([X_train.shape[1],]), # Create input layer with 'input data' neurons
    tf.keras.layers.Dense(10, activation="relu"), # Create hidden layer with 10 neurons and ReLU activation
    tf.keras.layers.Dense(1, activation="sigmoid"), # Create output layer with one neuron and sigmoid activation
])

model.summary()
```



↳ trainable data

↳ hidden layers např: 10 x

→ layer 0: matice 10 x 8 ← zadat

↓ volíme

→ data: trainable
non-trainable

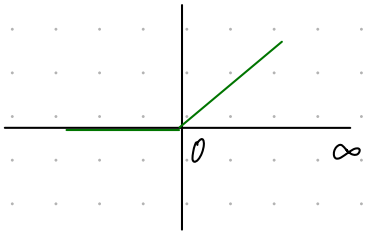
→ vždy: input, hidden layers, output

→ activation functions:

hlavní role v získávání outputu neuronu na základě inputu

zavádějí nelinearitu v síti, díky které se může model učit

- ReLU = common for hidden layers



↳ range $(0, \infty)$

→ output and inputs directly otherwise it outputs zero

→ negative inputs mapped to zero and possitive values remain unchanged

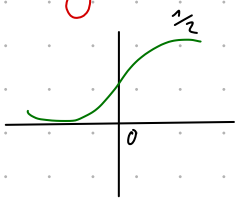
↳ can improve computational efficiency

→ derivative: 1 for positive
0 for negative ... simplifies back propagation

⊕ • computational efficiency
• helps mitigate (zmírňuje) vanishing gradient problem

⊖ • "dying ReLU" → when lots of inputs are negative

Sigmoid = output layer of binary classification



↳ range $(0, 1)$

$\Rightarrow 2\,\text{sigmoid}(2x) = \tanh(x)$

→ map any real-valued number to $(0, 1)$ making it useful for binary classication, where output represents the probability

→ smooth curve, which is differentiable everywhere

⊕ • useful for models where outputs are probability
• provides smooth gradients, which can help in the learning proces

⊖ • vanishing gradient problem, very small grad. during backpropagation
• not-zero centered outputs which can affect the efficiency of optimalization proces

- Functional model

→ definovaný rekurzivně

1. LOSS: použijeme Binary crossentropy

měřením vzdálenosti $H(p,q) = -E[\log q]$

crossentropy:

- output modelu je pravděpodobnostní rozdělení
- predikuje pravděpodobnostní rozdělení
- Loss: $\text{crossentropy}(y_{TRUE}, y_{PRED}) = -E_{y_{TRUE}} \log(y_{PRED})$

$$= \sum_{i=1}^{n} y_{TRUE}^{i} \log(y_{PRED}^{i})$$

y_{TRUE} jak daleko od y_{PRED}

2. $\theta_{new} = \theta_n - \alpha \frac{\partial L}{\partial \theta_n}$ + vybraný optimizer

↳ learning rate

3. Metrika

jak měříme Accuracy — AUC, Binary Accuracy, ⋮

Batch = máme batchsize = 64 a projdeme #64-ice = 1epocha

↳ nejčastější problém = OVERFITTING

↳ můžeme použít L_2-regularizaci, která pomáhá potlačit overfitting, nebo hovoříme parametru

- Define

```
inputs = tf.keras.Input(shape=(X_train.shape[1],))

hidden = tf.keras.layers.Dense(10)(inputs)
hidden = tf.keras.activations.relu(hidden)
hidden = tf.keras.layers.Dense(1)(hidden)
outputs = tf.keras.activations.sigmoid(hidden)

model = tf.keras.Model(inputs=inputs, outputs=outputs, name='Model')
```

```
model.summary()
```

- Model Compilation

```
# compile the model with selected optimizer, loss and metrics
model.compile(
        optimizer=tf.optimizers.Adam(), # Several other possibilities for optimizers
        loss=tf.losses.BinaryCrossentropy(), # Select the proper loss for the task
        metrics=[tf.keras.metrics.AUC(), tf.keras.metrics.BinaryAccuracy()], # Select the proper metrics for the task
)
```

- Train model with default setting

```
# train the model with default setting
model.fit(X_train, y_train, batch_size=64, epochs=100)
```

- Evaluate model

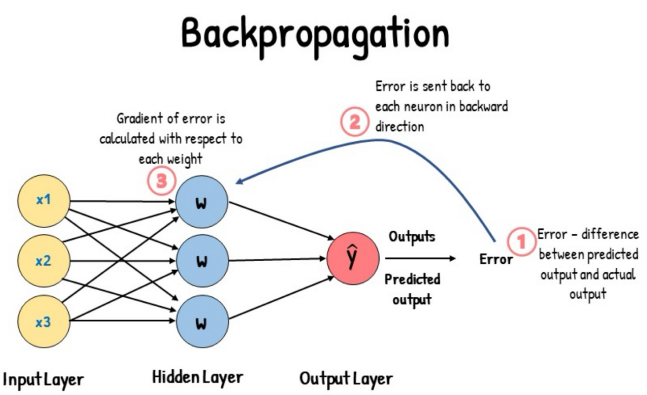
```
# Evaluate the model and predict for the test data
model.evaluate(X_test, y_test)
test_pred = model.predict(X_test)
```

→ add early stopping and regularization

→ Grid search and Tensorboard

obviously :

2. NN_backpropagation



→ forward and backward propagation were done
→ we defined true weights
→ use gradients

- Vanishing gradient
  = occurs when the gradients used to update the weights during backpropagation became exceedingly small, effectively preventing the weights from changing and the network from learning
  ↳ gradient loss function → repeated aplication of the chain rule, which multiplies gradients layer by layer from the output back to input
  → by activation function sigmoid or tanh can occur since they produce small gradients

  consequences:
  slow learning: small gradients => small weights updates
  - poor convergence: may fail to converge
  layer impact: early layers tend to be more affected

  solutions:
  - Activation function ReLU
  - Weight inicialization
  - Batch normalization = stabilize the learning process
  - Residual Networks = shortcut connections

Chain rule in NN

= used to calculate gradients during backpropagation

mathematical:

consider 2 functions: f, g: $g = g(u)$ and $u = f(x) \Rightarrow h(x) = g(f(x))$

↳ $\frac{dh}{dx} = \frac{dg}{du} \cdot \frac{df}{dx}$

→ Multiclass classification

3. CNN_intro

= images

- Convolution for border detection
  - -> RGB Channels
  - -> convert to grayscale .. preprogramed function
  - -> detect borders .. and apply convolution
    - ↳ masks = convolutional kernels or filters that are used to detect specific features within image

1. Sobel filters

Detecting horizontal edges:

$$G_x = \begin{bmatrix} -1 & 0 & 1 \\ 2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}$$

Detecting vertical edges:

$$G_y = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}$$

2. Prewitt filters

Horizontal prewitt filter

$$h_x = \begin{bmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{bmatrix}$$

Vertical prewitt filter

$$h_y = \begin{bmatrix} -1 & -1 & -1 \\ 0 & 0 & 0 \\ 1 & 1 & 1 \end{bmatrix}$$

Masks and CNN

1. Convolution operation
   - ↳ mask is convoluted over the input image, it involves sliding the filter across the image and performing element-wise implementation and summation to produce feature map
2. Activation function
   - ↳ ReLU is applied to introduce non-linearity
3. Pooling
   - ↳ pooling layers such as max pooling are applied to reduce the spatial dimention (prostorové rozměry) of the feature maps

-> we are changing maps

- Cats vs. dogs

→ image processing → resize, ..

→ Data augmentation
- random flip
- random rotation
- random zoom

} we need to recognize each time

- Training CNN
  - Define the architecture of the model

→ convolution blocks, hidden layers, ...

↳ activation function: "soft max"

- softmax → activation function for output layer of NN for classification problems, it converts the raw outputs scores into probabilities that sum to 100%.

1. prob $\Sigma = 1$
2. exponentation
3. normalization

example: Raw scores (logits) produced by NN are $z = [2, 1, 0.1]$

- exp: $e^z = [e^2, e^1, e^{0.1}] = [7.389, 2.718, 1.105]$
- Σ: $\Sigma e^z \approx 11.212$
- devide $\frac{e^z}{\Sigma_z}$: $\frac{7.385}{11.212} = 0.659$; ... => result $[0.659, 0.242, 0.099]$

=> NN can output probs. for multiclass classification tasks

  - Compile the model
  - Create batches from the dataset

↳ Note on what should be the numbers that we see during the training: "For training loss, keras does a running average over the batches. For validation loss, a conventional average over all the batches in validation data is performed. The training accuracy is the average of the accuracy values for each batch of training data during training." !

  - fit model
  - make predictions

↳ we get set of probabilities and choose the argmax so we get the indexis where the argmax is attained

→ Bigger CNN with regularization

- Define: Convolution blocks and hidden layers
- Create Batches with label smoothing
- Compile the model

→ Deeper network with regularization

4. RNN – intro

→ text classification

- Define the network
- Define hyperparameters
  Fix seed and set threads (ulá'hna)
- Check data

⟶ target .. word strings

- Train the model
  → long-casting
  → train move times
- Large movie review dataset

Hyperparameters

aproach for optimal hyperparameters:

1. Grid Search = every combination from search space is evaluated
2. Randomized Grid Search = param. value is selected from distribution of possible values. # of eval. combinations can be restricted independently on # of parametens and their possible values
3. Grid Search with halving
4. Randomized grid search with halving = first bot of comb. are evaluated using small amount of resources
   In next itteration only comb. with biggest potential are considered and evaluated again with more resources

   L> params. of halving process: resource, n_candidates, min_resources, factor

- Hyperopt
  -> library for serial and paralled optimalization uses Bayesian Optimalization principles.
  We need to describe:
  1. The objective function to minimize
  2. Search space
  3. Database to be used for storing results
  4. Search Algorithm

- Genetic Algorithm

- Hyper-parametens optimation with NN
  -> Keras tuner by Random Search
  L> early stopping can be used
  -> Grid Search with tensorflow
  -> Tensorboard

Clustering

= how to cluster data into homogennous groups

- generate dataset
- create clasters using hieravchial clustering
  - -> vizualization of results for difevent algorithm paravneters
  - -> vizualization of decision tvee
- create clusters using k-means clustering
- predict continuous target using k-nearest neighbour algorithm
- Hierarchial clustering
  - -> bottom up approach: each observation stavts in its own cluster and clusters ave successively merged togethev. In each step ve find two clusters with the lowest measure of disimilavity and merge them until either
    - -> only "n-clusters" is produced

Přístup zdola nahoru: každé pozorování začíná ve vlastním shluku a shluky se postupně slučují. V každém kroku najdeme dva shluky s nejnižší mírou nepodobnosti a sloučíme je, dokud nevznikne pouze „n - shluků".

- Distances
  - L1
  - enclidian

" vizualization results for different metrics

- Dantogram
  - Tree-like structure that fully represents the hierarchical models
  - every leaf represents the single data print and every mode represents the cluster that contains all data points in its subtree
- K-means

in 2 steps:
- assign the nearest centroid for every data point
- adjust centroids that they represents "their" datapoints

stopping time: # of iterations on distance tvesh hold

=> 2 phases in every ittevation:
- find the neavest neighbovn
- adjust centvoids

- K-nearest neighbovs