Data Analysis and Machine Learning, tutorials, 2024-2025

Nicky van Foreest, Aico van Vuuren

October 14, 2024

Contents

1 Tutorial 1 **4**

2 Tutorial 2 **9**

3 Tutorial 3 **14**

4 Tutorial 4 **18**

5 Tutorial 5 **21**

Introduction

General info

Each of these tutorials contains two parts. In the first part, we will use sales data of second hand Toyota Corollas to showcase the machine learning techniques discussed during the lectures and the book related to the course. In the second part, we will analyze geographical data and ultimately build our own simplified version of the routing app Komoot. Here and there I include some algorithmic challenges that I find them sufficiently interesting to work on myself. Some of these challenges are a bit hard. If this is the case, I indicate them as optional. (If you have a good idea how to solve and program the challenges, please let me know.)

The lectures and the book provide ample motivation to understand why we study Toyota case. As for the second part, besides that it is fun to make cool maps that you can share with your family and friend,

1. You will learn to work with the website of `openstreetmap` and with geographical data.
2. Using geographical data will confront you with many challenges on how to deal with big data. If you program sloppily, your algorithms will work on small cases, such as the scale of a village or a city, but very slowly on a scale of a province like Groningen, and not anymore on the scale of country. The data will not fit into the memory, it will takes ages before your programs finish, if at all. Thus, you have to develop strategies on how to work with real big data.
3. You will make visual representations, in the form of maps, of your work. In our Komoot case, this is just fun, but in a professional setting, it can be very important to make good figures. Of course, making figures takes time, but, as they say, a good drawing can save a thousand words.

4. You will learn trees to classify tracks into good and bad tracks. Trees are data structures that allow to very fast classification.

Stated in other terms, the largest part of data science is not concerned with developing the hottest machine learning techniques and tuning super large neural networks. The largest part is concerned with obtaining an understanding of the data (so that you clean it), visual communicating about data and the information you can derive from it, and struggling with databases and other methods to handle large amounts of data.

Spend the first hour of the tutorial on the Toyota case and the second hour on the geographical cases.. Like this it will become clear during the tutorials what I expect from you for each type of topic. You can then finish the rest as homework. Note that there are many questions, but many are real simple, if you use ChatGPT.

In fact, you are expected to use ChatGPT to help you write **Python** code (not R) for the exercises. Be aware: Using ChatGPT for programming offers many advantages (and for these I use it all the time myself):

1. ChatGPT helps by providing instant code snippets, and debugging advice, saving time on research, troubleshooting, and reading manuals.

2. It can explain complex concepts in simple terms, making it a useful tool for beginners or those looking to learn new programming languages or frameworks.

However, there are two points to watch out for (and because of this, I check each and every line of ChatGPT's code to be sure I understand what it does):

1. While ChatGPT can generate useful code, it might not always produce the most efficient or accurate solutions. Users should verify the output before using it in production environments.

2. ChatGPT might not fully understand the broader context of a project, leading to solutions that don't align with the specific requirements or nuances of the user's code base or system.

One topic we don't deal with in these tutorials is text cleaning and text analysis. There are a few reasons.

1. Automatic text analysis is very hard, how to chop up a text into sentences? Just splitting on points is not the rule. To convince you about the complexities consider the phrase: "At 3 p.m., Dr. Smith, who earned his Ph.D. at Harvard Univ., met with Prof. Johnson, M.D., to discuss the latest research published in the Journal of Neuroscience, vol. 23, no. 7, pp. 89-104, while also referencing the works of Dr. Dre, Inc., in the field of audio engineering.".

2. Sentiment analysis is also very interesting, and very hard. Consider: "This food is bad, not good at all." versus "This food is good, not bad at all." Just counting how often words like "good" or "bad" occur is not necessarily a good strategy.

3. In one course we can't do everything.

If you use a Mac, you might consider to install Pycharm to edit and run python code: `https://www.jetbrains.com/pycharm/`.

Grading

You work in groups of 4 on the tutorials. I (=nvf) will invite each group one week after each tutorial for an interview of 10 to 15 minutes. To each interview you bring a laptop with the figures and/or maps you made, and all the **python** code you developed (with ChatGPT) for that tutorial. During the interview I will ask *each* group member to explain a part or line (that I will select randomly) of your code or the results of your programms. I don't want to see any comment in your code; I expect that each group member can explain, in detail, *all* code that the group developed for last week's tutorial. In other words, I don't accept that you distribute the work of a tutorial among the 4 group members, and that each member understand just his/her part.

The members of the group should agree on one piece of code, and show this to me. The reason is this. In banks, insuranse companies, pension funds, it is required that code is read and validated by (at least) two people, the so-called 4 eyes principle. Thus, you have to practice with reading, understanding, and commenting on, other people's code.

A student receives 2 points for a correct answer, 1 point for a vague answer, and 0 points for a bad or wrong answer ((or no answer at all). Finally, if I have doubt about the correctness of the code, you have show me that it runs and produces the results required for the tutorial. If it doesn't, all group members get 0 points.

Let k be the total number of points obtained by a student for the 5 tutorials, and $p \in \{0,1\}$ depending on whether the student was present or not at the welcome event on September 4. Then the student gets as final grade for the assignment: $1+9(k+0.05p)/(10.05)$.

Useful and/or interesting websites

Here are some sites that I found useful or interesting (or both).

- Using Python for Data analysis
- Making nice geographical maps with Python
- `https://janakiev.com/blog/osm-compare-countries-and-cities/`
- `https://janakiev.com/blog/osm-predict-economic-indicators/`
- Machine Learning Framework for the Estimation of Average Speed in Rural Road Networks with OpenStreetMap Data

1 Tutorial 1

1.1 Toyota Corollas

This tutorial has as aims:

1. Analyze the data in the Toyota file.
2. Learn how to do regression of price on numerical and categorical features.
3. Make a correlation matrix and a heatmap.
4. Come to appreciate how incredibly useful ChatGPT can be for data analysis with Python.

1.1.1 Study the basic data

1. Read the `ToyotaCorolla.csv` with pandas into a dataframe and print the header of the frame. Check what you see.
2. Print the types of the columns, and the range of the values of the columns.
3. We need to find out which columns are categorical, and which are numerical. Here is my question to ChatGPT to give me some useful code: "How can I determine if a numerical feature in my dataset is actually categorical? Explain different methods I can use to identify this."
4. I did not know what the column with name `HP` meant. So, I asked ChatGPT what that could be in the context of Toyota car sales data. Now you know what to ask ChatGPT :-). Check the all unique values of this column to check that ChatGPT is right (at least it was for me). Once you understand the meaning, decide whether `HP` is numerical or categorical.
5. Drop the `Model`, `Id` from the dataframe, and any columns you suspect to be useless to predict prices.
6. Get some high level statistics of the file, such as number of rows, mean and variance per column.
7. Are there any duplicate entries in the dataframe? One or two is perhaps not a real problem for a dataset of this size, but if there are many doubles, we have to remove duplicates. (Again, as a hint, ask ChatGPT how to find duplicates in a pandas dataframe.)

1.1.2 Analyzing the price of cars

1. Make a histogram of the prices.
2. Use `seaborn` to make a kernel density estimate (KDE) plot which is a smoother alternative to a histogram.
3. Plot the data points price as a function of age (use a scatter plot).

4. Make a linear regression of the price-age relation, and plot that in the same scatter plot, so that we can compare the linear relation and the actual data. Tell Chatgpt you want to use `numpy.linalg.solve` for this task. (In a later tutorial we will use functionality of `sklearn`.) In other words, like in example `plotreg2.py` of the book, you build the linear regression yourself.

5. Make a box plot, like Figure 1.8 of the book, to see how price depends on ABS.

6. In the box plot, I get lots of circles outside of the range of the whiskers. Why is that? Ask ChatGPT what this is, and how to remove them.

7. Are Diesel cars more expensive on average?

Hopefully you get the point: Data analysis starts with lots of simple tests and plots.

1.1.3 Correlation matrix

Here is what I asked ChatGPT: 'I have a pandas data frame with prices (numerical values) and age (also numerical), but also with categorical variables such as color. How to make a correlation matrix and a heapmap for such mixed cases?'

1. Formulate my question in your own words, and ask ChatGPT for an answer. Read its suggestions and code carefully, and tinker with it until the code does what you want it to do.

2. Figure out how remove the correlation of price with itself, as this is uninformative.

3. Make a heatmap of the correlation between all features.

4. What seem to be the most important features to predict the price of a car?

5. In my heatmap the numbers fall on top of each other. If that happens with you too, repair this to make the heatmap look nicer.

6. Another idea to beautify the presentation is to use a pass on a mask (with `np.triu`) as an argument to `sns.heatmap`; ask ChatGPT for the details.

1.2 Plotting garbage containers on a map

This part of the tutorial has as aims:

1. learn how to make an html page that contains a map with the locations of garbage containers in the city of Groningen. (I find the map pretty cool, actually.)

2. Work a bit with a sql database.

3. Use openstreetmap to look up geographical information such as the latitude and longitude of nodes.

1.2.1 Become familiar with the data

The sqlite database `waste_for_students.db`, available on Brightspace, contains the locations of all underground waste containers in the city of Groningen, and the times at which the drum is opened for each container during a period of three months.

1. Find a program to view the data in the sqlite database. (I use `sqlitebrowser` on Linux. There must be a windows and mac equivalent.)
2. What tables does the database contain?
3. Which is the one with the locations of the containers?
4. Check the content of the relevant table. Do you understand the format and the type of the data?
5. How many container locations does the table contain?

1.2.2 Plotting the container locations

1. Make a python program that uses pandas to read the sqlite database. BIG HINT: I typed this at the ChatGPT prompt: "I want to read a sqlite database with pandas.". Then I got an answer from ChatGPT, and I read the code *carefully*. With the sqlitebrowser (the previous step), you have seen which tables the database contains, and which table you need to read specifically. Then read the code again to figure out where you should type the name of the table.
2. We will use the python library `folium` to make a map that you can view in a browser. You should know that a `folium` map needs a center and a zoom factor. The zoom is easy, just run the program with different zoom factors until you find something useful. (Once you view the map in the browser, this remark will be completely clear.) However, finding a good center is a bit more involved. I use `https://www.openstreetmap.org/#map=9/52.626/7.057`. Then I click somewhere in the middle of the province, and look up a node. Then, by querying the node, I find its coordinates, and those I copied into my code and use that as center. Now you do the same.
3. In step one you loaded the location. Now extend your program to use the python library `folium` to plot a marker on the map of Groningen at each container's location. Write the map to an html file. You can view the file with your browser by opening it. BIG HINT: think of a good prompt for ChatGPT. If that prompt does not answer your question, improve your prompt based on the answer of ChatGPT. I suppose you get the idea now: HENCEFORTH ASK CHATGPT FOR EACH AND EVERY QUESTION AND READ ITS ANSWERS.
4. Include the id of the containers in the markers made by folium, so that when clicking on the container, a text box appears with the container id.
5. Show the station name when hovering over the station icon, instead of having to click on the icon to see the name. Hovering is often much nicer for a user.
6. As there are about 1000 container locations, the map becomes somewhat crowded with markers. Find some way to cluster the markers on the map. If you do this in the right way, you get a heatmap of the container density (num containers per km^2) for free. (With

your and ChatGPT's help I discovered several ways to make heatmaps: `MarkerCluster`, `FastMarkerCluster`, and `Heatmap`. You can try just one, or all three, if you like.)

7. Find the id of the container nearest to your house.

8. Walk to your selected container, and check whether the id is correct, :-)

9. Try to summarize what you skills just learned. Then ask ChatGPT for what purpose you can use these skills. (I first thought about making a list of problems that you can tackle with your just acquired skills, such as plotting the density of crime rates in city districts. However, I suddenly realized that ChatGPT can also answer this, and better perhaps.)

1.3 Why *not* compute $(X'X)^{-1}$

This part is homework.

Recall that for linear regression we seem to compute β with the formula $\beta = (X'X)^{-1}X'y$. Copy the next code, run it, and see what happens.

```python
import numpy as np


def hilbert_matrix(n):
    # make Hilbert matrix of size n x n
    A = np.zeros((n, n))
    for i in range(n):
        for j in range(n):
            A[i, j] = 1 / (i + j + 1)
    return A


n = 10
A = hilbert_matrix(n)
b = np.ones(n)
A_inv = np.linalg.inv(A)
x = A_inv @ b
print(A @ x)

n = 15
A = hilbert_matrix(n)
b = np.ones(n)
A_inv = np.linalg.inv(A)
x = A_inv @ b
print(A @ x)

# This  is the good way to solve A x = b
x = np.linalg.solve(A, b)
print(A @ x)
```

Check that, after inverting A we get solution x with negative values; but that must be impossible because A and b are strictly positive. Conclusion:

1. The numpy solver that solves $Ax = b$ by suitable methods works reliably.

2. A seemingly small change, from $n = 10$ to $n = 15$, can give drastically different results. So, we should not rely on naive intuition.

3. Inverting a matrix is terrible, even when the matrix A is small.

More generally, do **NOT** invert a matrix to solve an equation of the type $Ax = b$.

1. Solving $Ax = b$ has an algorithmic complexity of $O(n^2)$, whereas solving A^{-1} is $O(n^3)$.

2. The computation of A^{-1} can be numerically unstable.

3. Even when A is sparse, A^{-1} can be dense. You can go from an $O(n)$ size matrix to an $O(n^2)$ size matrix. The former can fit in memory, the other not.

4. Check out page: `https://www.johndcook.com/blog/2010/01/19/dont-invert-that-matrix/`.

In most general terms, use toolboxes for standard problems, don't believe that standard algorithms are simple to get right. Coding is tough.

1.4 Analyzing Spotify data

This part is homework. What I learned from this case is that conclusions that seem to follow from small datasets no longer hold for larger sets. I downloaded these data sets:

- `https://www.kaggle.com/datasets/maharshipandya/-spotify-tracks-dataset`
- `https://www.kaggle.com/datasets/lehaknarnauli/spotify-datasets`

It might be that the links have been removed by the user. I found these datasets by googling on "Spotify kaggle datasets". It the above links don't work, just google.

1. For the small set, regress song popularity on tempo to see a correlation.

2. Make a correlation matrix of the features based on the data of the small set.

3. Do the same two steps for the big set.

This comparison leads to interesting conclusions. For instance, the data of the small set suggests that song popularity increases with tempo, but in the large set, I cannot find this relation. This leads to a somewhat awkward problem: when to believe conclusions from data? What is, actually, good data? I don't know the answer BTW.

Here are two nice sites:

1. Predicting the success of songs on Spotify.

2. Spotify data analysis.

.

The rest is of this section is optional.

1. You can download data about your own play history on Spotify (see the web on how to do this). Perhaps you find it interesting to apply the tools you learn in this course to your own data.

2. Suppose you would like to share your spotify play history with your friends, how to prune this list if it contains songs you feel ashamed of listening to (for instance, Taylor Swift :-)). It's an interesting data analysis problem to classify songs into two types: Willing (or not) to tell others that you listen to such songs.

2 Tutorial 2

In the first hour of the tutorial we work on the Toyota case, in the second on a geographical challenge.

In this tutorial, and the rest, we will use following python packages:

- `numpy`
- `sklearn`
- `matplotlib`
- `seaborn`
- `folium`
- `osmium`. (It is called `pyosmium`, but to install it, use `pip install osmium`).

If you don't what a package does, ask ChatGPT.

2.1 Toyota

In this part of the tutorial, we will develop code to:

1. Split the data into training and testing sets.
2. Fit a linear regression model using both numerical and categorical features. You can use one hot encoding for this. Read about the option `drop='first'` and decide whether to use that option or not.

For a deeper discussion on the theory and implications of over-fitting, please refer to the relevant sections in the book and the lectures. I find the explanation on `sklearn` on cross validation very helpful. BTW, even when using ChatGPT to build code, I check the actual documentation to see what these all these functions do exactly. ChatGPT does make mistakes...

2.1.1 Simple regression

We start with using `sklearn` to regress price on age.

1. Use `sklearn` to make a train and test set of the Toyota data.
2. Then, make a pipe line with a standard scaler and a regressor.
3. Fit the model, and make a plot of predicted prices for the test set and the actual prices in the test set. What can you conclude from the figure?
4. Compute the score of the pipeline (Again, just ask ChatGPT how to do this).
5. Now use a 5 fold cross validation to obtain an array of R2 scores. You can use the `sklearn` function `cross_val_score` with `cv=5` for instance. Print this array, its mean and its std. Interpret the outcome. BTW, ensure you can explain what happens under the hood. ChatGPT gives very clean code, but the code is very high level.

6. The analysis of the previous step can result in largely varying scores of the estimators. Why is that? (Hint, use `Kfold` and read the manual to see it can shuffle the data.) Feed a `Kfold` object into `cross_val_score`; ask ChatGPT how to do this. Do the scores improve? Why is that?)

7. Evaluate the model's performance on the test data to see if it generalizes well.

2.1.2 Regression on multiple features.

1. Now perform a regression of prices on both age, ABS and color (or some other categorical feature). Does ABS change the impact of age by a lot? Realize that now you deal with a a mix of feature types, i.e., numerical and categorical.

2. Add some further features to the feature list such kilometers and the number of doors to the regression and see whether the accuracy decreases. Ensure to include a few categorical variables, and handle them correctly.

3. Then, make a figure (with two sub figures). One shows the predicted price as a function of the actual price. The second plots the relative error of the predicted price as a function of the actual price.

4. Do a cross validation for this model too.

5. As a somewhat meta question, is it reasonable to expect to be able to predict the price very accurately? Actually, I don't think so, because the price of a second hand car is the result of a bidding process, and some people simply are more insistent than others. There must be a lot of variability. What is your stance on this matter?

What is your opinion about this code? What is conceptually wrong?

```python
preprocessor = ColumnTransformer(
    transformers=[
        ('num', StandardScaler(), ['Age_08_04', 'ABS', 'KM']),
        ('cat', OneHotEncoder(), ['Color', 'Fuel_Type'])
    ])
```

2.2 Plotting benches in Groningen

The aims of this tutorial:

1. Learn that data *exploration* is a key factor in data analysis. In this case we explore OpenStreetMap.

2. To work with data in OpenStreetMap files and make maps of this.

3. You might encounter interesting algorithmic challenges, like how to align the stations such that the routes are printed nicely. If the alignment is not correct, you'll end up with a figure with arbitrary lines between stations. Surely, these random lines don't follow the actual rail tracks.

2.2.1 OpenStreetMap website

There are many interesting things to do with the data in OpenStreetMap.

1. Go to `https://www.openstreetmap.org/#map=19/53.21654/6.55079`
2. Right click on some place on the map and select `query features` with a right click of the mouse. (When you use another language in your browser, the words `query features` will be different. Just explore a bit.) At the left, you'll see a vertical bar with information.
3. Try to understand what features are available and what they mean; just click on a few links in the left bar.
4. Find a bench near your house, and get its latitude and longitude. Find out its tags by querying the features. For instance, a bench is of type `amenity`.
5. Explore this website a bit more, to see what is available. For instance, query on a canal, or forest, or something else that you find interesting. The aim here is that you see what type of information is available.

2.2.2 OpenStreetMap data files

1. Download the `trompbrug_area.osm` file from Brightspace.
2. Open this file with an editor like notepad, and study its content. For instance, search on the word `bench`, to see what information is associated with a bench node.
3. Ask ChatGPT how information about how `node`, `way` and `relation` are organized in osm files.
4. Then search on `<way` and `<relation` in the `osm` file, and check the explanation of ChatGPT. The aim of this step is that you understand how geographic data is organized in such openstreetmap files. The pbf file contains the same information, but highly compressed; the pbf is friendly for computers, the osm file for humans. In my code I tend to use the pbf files, because they are (much) smaller. (I don't know which file format loads faster, or is faster to work with.)

2.2.3 Plotting benches with folium

1. Install `pip install osmium`. This is a module that allows you to read relevant information of a pbf file. In the past I tried to work with `osmnx` or `pyrosm`, but I did not find them easy to use (which may very well be my fault). Let me repeat: I expect you to install and use the `osmium` module.
2. Download the file `groningen-latest.osm.pbf` from `geofabrik`.
3. Tell ChatGPT that you want to use the `osmium` module from python to make a `handler` to read the locations of benches. (If things go well, ChatGPT will tell you to use a `SimpleHandler` class). With this handler, you can obtain a list of the gps coordinates of all benches in the province of Groningen.

4. Plot the benches with `folium`. In the previous tutorial you learned how to do this for garbage containers.
5. Can you make a heatmap of the density of benches?
6. Try other amenties, like pictic tables. The tag to look for is `leisure` with value `picnic_table`. Here is an example.

Optional challenges: plot all cafes, restaurants in Groningen city. Prof Van Vuuren used exactly these tools to study whether this is relation between the density of amenities (e.g., cafes) and house prices.

2.3 Plotting train stations

On a nice summer's day we would like to take a long walk, on some (for us) unknown part of the Netherlands. Why not walk from one train station to some nearby, other, station? Now, if for some reason we like to walk from Coevorden to Hoogeveen (a 26 km walk km), and we like to the take the train back, we need to travel via Zwolle, which is a considerable detour. If we don't want this, we need to select stations that lie on the same track. So, to select two nice stations for our walk, we are going to make an html page with all train stations and the railway tracks that connect them.

2.3.1 Steps

1. Figure out how to obtain the node coordinates of all railway stations. (If you instructed Chat-GPT well, it will tell you to use a `SimpleHandler` class).
2. Plot the stations on a html page with `folium`.
3. Extend the map with the train tracks.
4. Make nice icons for the stations, and add these icons to your map.
5. Add station names to the markers. Enable hovering if you like.
6. Make the size of the box containing the station name dependent on the length of the station name.
7. Compare your map with map offered by Openstreetmap. Hopefully you find your map much more helpful for the goal of finding directly connected stations.

2.3.2 Filtering the railway data

The data in geographical pbf files is highly compressed (with lots of smart tricks). In expanded form they can take many gigabites. To save time, I run a program called `osmium` to obtain specific information from a big source file and write the filtered data to a smaller file. The source files can be downloaded from `geofabrik`; just google on `map, geofabrik` or something similar to get to the site.

Here are some examples on how to use `osmium` (if you plan to make your own files). This is not necessary, however, the files you need for the tutorials are on Brightspace.

These are the scripts I used to filter the large file `groningen-latest.osm.pbf` to make small files for the railways.

```
osmium tags-filter /home/nicky/tmp/data/groningen-latest.osm.pbf \
    nwr/railway=station \
    nwr/route=railway \
    -o /home/nicky/tmp/data/railway_data.osm.pbf \
    --overwrite
```

Now I filter on `route=train` instead of on `route/railway/`. (It requires a bit of modification (but not much) to see how that changes the map.) BTW, it is not always clear what tags will give the best results; just experiment and use the query feature on OpenStreetMap to see what tags and features might be useful for the task.

```
osmium tags-filter /home/nicky/tmp/data/groningen-latest.osm.pbf \
    nwr/railway=station \
    nwr/route=train \
    -o /home/nicky/tmp/data/railway_data.osm \
    --overwrite

osmium tags-filter /home/nicky/tmp/data/netherlands-latest.osm.pbf \
    nwr/railway=station \
    nwr/route=train \
    -o /home/nicky/tmp/data/railway_data.osm.pbf \
    --overwrite
```

With this last file, you can plot all stations of the Netherlands.

2.4 Plotting fuel stations

Read the problem, to see another application of what you just learned. The coding is optional, just do it if you find this fun.

1. A student of mine needed to find out good potential locations for hydrogen fuel stations in the Netherlands. It is clear that hydrogen fuel stations need to be spread (building one next to another makes no sense.) Use OpenStreetMap to find a hydrogen fuel station. Then check all the labels and tags that describe the fuel station.

2. Plot the fuel stations on the map of the Netherlands.

3. Good potential locations are other, regular, fuel stations. Plot all regular fuel stations, in red say. Re-plot the hydrogen fuel stations in green.

4. With the map we can visually check whether the potential locations are indeed what we need.

5. The last step is select a set of regular stations whose area is sufficiently large to place an extra hydrogen fuel station, and are spread over the Netherlands. You can skip this step, but for your interest we made a mixed integer problem for this.

3 Tutorial 3

3.1 Toyota prices

Besides using ChatGPT, I recommend to read the manual of `sklearn` on lasso and ridge. The `sklearn` function LassoCV is remarkably powerful, so powerful in fact that it takes time to understand what is going on under the hood. You should read the documentation of this function!

- Make a graph of the regularization curves of the feature weights when using Ridge as a regularization method. Figure 6.3 of the book provides an illustration.
- What can you learn about the features from the regularization curves?
- Make a similar graph when using Lasso regression.
- What difference do you see between two types of regularization?
- What are the most important features to predict car prices?
- If all went OK, ChatGPT advised you in the previous steps to use a standard scaler. Remove this scaler from the pipeline, and make the regularization curves again, for instance for lasso. Do you get bad results? (Hopefully, yes, so that you see the need to scale the data.)
- Now use cross validation with Lasso regression to ensure that your model generalizes well to unseen data by preventing over-fitting and enabling the selection of the best regularization strength alpha. Warning: the code offered by `sklearn` is very high level. Ensure to understand what each function does.
- Print the model scores for each fold and each alpha. Compute the average over the folds for each alpha, and print the results.
- Compute the score of the model with the best alpha on the test set.

While playing with ChatGPT, I discovered two ways of scaling a bunch of numbers $a_i \in \mathbb{R}$. One is like this. Let μ and σ be the mean and std of the series $\{a_i\}_{i=1}^n$. Then, take $\tilde{a}_i = (a_i - \mu)/\sigma$. Another is this, let m and M be the min and max of the series. Then, set $\tilde{a}_i = (a_i - m)/(M - m)$. I don't know which type of scaling is better.

3.2 Geography

In this tutorial we'll make a plot of the boundary of the province of Groningen. This is useful because once we have the boundary of some area, we can compute lots of interesting geographical or societal statistics, so as the population density (of some insect or animal), the percentage of farm land or forest. Next, you will use a graph with nodes and edges to store and organize roads. Third, in one question you will learn that dealing with real big data brings its own challenges.

3.2.1 Getting the data

Here is the osmium code I ran to make the file. If you like, you can make your own files. Otherwise, you can find the file for this tutorial on brightspace.

I ran this osmium code to select the relation with the right id. The option `-r` says that I want to recursively read the ways and nodes related to the indicated relation.

```
osmium getid -r ~/tmp/data/groningen-latest.osm.pbf r47826 \
       -o ~/tmp/data/groningen-boundary.osm --overwrite
```

3.2.2 Plotting the geographical boundary of the province of Groningen

1. Use the website of OpenStreetMap to find the boundary between Drenthe and Groningen.
2. Select some point on or nearby the boundary, and query on the features.
3. Search in the bar on left hand on the name 'Groningen'. There you'll see something like 'state boundary'.
4. If the above does not work, choose a node nearer to the boundary between the two provinces.
5. Click one the link of the state boundary to get a `relation` that corresponds to the boundary of the province of Groningen. Behind the relation you can see the id of this relation; it's 47826.
6. Read the osmium code below to see how I made an `osm` file that contains the relevant information for this relation. For your convenience I put the `osm` file on brightspace.
7. Open this file with an editor, like notepad, and check its contents. It's astonishing how much information is involved in just marking the boundary of Groningen.
8. In python, use an `osmium` handler to read out the ways and nodes of this relation.
9. Find an algorithm to compute the center of the map (recall, a `folium map` needs a center and a zoom) based on the coordinates of the nodes of the relation.
10. Develop an algorithm to plot the boundary on a map.
11. It might happen that you get a very strange set of lines. The reason is that the ways do not connect in the right direction. (If this does not happen, you're lucky.) To repair this, I put all nodes in a graph (with `networkx`). Then by plotting the edges of the graph, the problem is resolved.
12. Compute the total length in km of the boundary.
13. To see why a filtering step with `osmium` is necessary, let the handler read the relation of the boundary of Groningen from the file `netherlands-latest.osm.pbf` instead of the file I made available. You'll can get a coffee before the computations are finished and all your RAM will be consumed.

Some remarks:

1. With this tool we can make maps of the boundaries of countries, cities, city districts!

Here is are some real nice challenges. They are optional, because they require ingenuity and some not trivial algorithmic thinking.

1. Try to fill in the boundary with a blue color and opacity of 0.3. With this technique, we can show all provinces of the Netherlands with their own color.

2. Can you compute the area of the province of Groningen?
3. What is the population density?
4. How to restrict the area to land (Part of the province lies in the Waddenzee. The see is not considered to be habitable land.)
5. For the mathematically interested: you can try to use Green's theorem to rewrite a surface integral in terms of a loop integral.
6. Can you apply Green's theorem while taking into the account the curvature of the earth?

3.2.3 Distances between waste container and depot

This is a continuation of the waste container project.

Once the planner has decided which containers have to emptied on a certain day, the planner needs to make good routes for the garbage trucks. And for this, the planner needs a distance matrix containing the distances between any two containers (for our project with the municipality we needed this distance matrix). Let's make just one element of this distance matrix.

1. Look up the location of the depot in the sqlite database, i.e., the database containing the locations of the containers.
2. An easy exercise (for ChatGPT): for your garbage container of choice, compute the distance as the crow flies between your container and the depot.

Optional: If you like a more difficult challenges, use the Open Source Routing Machine (OSRM) to compute the distance over the *road*, rather than through air, between:

1. your container and the depot.
2. All individual containers
3. Take one-way streets into account.

3.2.4 Cafes along Pieterpad

This is optional. When working with large graphs, several 100K of nodes, `networkx` becomes noticebly slow. I spent some time setting things up with `igraph`, which works much faster.

1. Filter, with `osmium` all amenities marked as cafe, restaurant, or similar, from the provinces Groningen, Drenthe, Overijssel, Gelderland, and Limburg.
2. Make a graph (not a figure, but a graph with nodes and edges) with the amenities as nodes, and add edges between all nodes that lie less than 30 km apart on the map.
3. Plot this graph on the map of the Netherlands.
4. Can you find the shortest path from Pieterburen (in Groningen) to the Pietersberg (Limburg) along these amenities?
5. Can you find the shortest path from Pieterburen (in Groningen) to the Pietersberg (Limburg) along these amenities such that the distance between any two amenities along the path never exceeds 7 km, say? (So, every one and a half hour you can stop for a coffee.)
6. What is the shortest possible edge length between two cafes so that there is still a path possible between Pieterburen and the Pietersberg? For instance, there is no such path if the largest distance between two cafes must be less than 500 meters.

4 Tutorial 4

In this tutorial we will use trees for price prediction, but also for as a smart data structure to find nearest neighboring points. Trees are very versatile!

4.1 Toyota prices with random forests

In this part, we will develop a random forest to predict the prices of Toyota cars.

1. Do we need a random forest regressor or a random forest classifier to predict prices?
2. Build a simple random forest. (Use a pipeline to deal with the mix of categorical and numerical features.) How is the depth of the trees determined, and what is the number of features per split?
3. Fit the model on some training data and print some relevant statistics.
4. Plot the predicted vs the observed prices on the test set. You might also want to plot the relative residuals.
5. Make a histogram of the depths of each of the trees in the forest.
6. Make a bar plot of the most important features. (I got this line of ChatGPT: `all_feature_names = list(categorical_feature_names) + numerical_features`. Ensure that this is the same sequence of the pre-processor. For fun, reverse the sequence and see what you get.)
7. How is the importance of a feature determined? Is this easy to understand or, perhaps, to communicate to a non-specialist (i.e., a manager)?
8. It is possible to use a sklearn built-in function to use cross-validation to help assess the generalization ability of the forest. However, now your task is to implement a five-fold CV by yourself, and print the mean absolute error for each fold.
9. I get as a result that the mae is around 800. Is this a large, small, acceptable? Can you provide some background on how to interpret this number, and what it says about the quality of the forest?
10. Figure out how to do a grid search on the number of trees in the forest and the depths to find a good forest. You don't have to run the code, as long as you know how to code it.

Optional: I asked ChatGPT what other interesting things we could analyze from a random forest. One of its suggestions was to make a graph of one of the trees in the forest. Perhaps you can try this too. I tried `graphviz` but the tree was so large that it was barely readable in the `pdf` file to which I saved it. BTW, when plotting many points, it can help to write a plot to a `png` file instead of a `pdf` file. Then I asked ChatGPT whether it was possible to write the tree to an html file. It suggested to use `dtreeviz`.

4.2 Build your own Komoot

We will build a toy version of Komoot; check out the Komoot website to see what the app does. In particular we will compute and plot the shortest hike between two given points. You will learn quite a few things about OpenStreetMap and you will trees and graphs to simplify a number of algorithmic tasks. Finally, visualizing large graphs is an important skill, not only for maps but also in, for instance, social sciences.

4.2.1 Preparation

Here are the instructions for `osmium` to make the correct files. You can skip this, but I include it if you want to make the files yourself.

```
osmium tags-filter /home/nicky/tmp/data/groningen-latest.osm.pbf \
       w/highway  \
       -o /home/nicky/tmp/data/highways-groningen.pbf \
       --overwrite

osmium tags-filter /home/nicky/tmp/data/groningen-latest.osm.pbf \
       r/route=hiking r/route=foot \
       -o /home/nicky/tmp/data/hiking-foot-groningen.pbf \
       --overwrite
```

4.2.2 Plot hiking and foot routes

We want to walk from A to B, where A and B lie about 15 km apart. Let's try to make a map of attractive paths. Later we will select real points for A and B.

You can find the files `highways-groningen.pbf` and `hiking-foot-groningen.pbf` on Brightspace.

1. Use the file `highways-groningen.pbf` to make a map of all highways tagged as `track`. I colored all these tracks blue. Instruct ChatGPT to use `osmium` in the code it gives to you.

2. Once you have the map, you can use google streetview to see what a track is. For example, check out "Laan naar het Klooster".

3. If your map is correct, you must have notices that the highways of type `track` do not form a connected graph. So, such highways don't get us very far.

4. Googling on "highway tags OpenStreetMap" leads to me an overview of useful tags. `footpath` and `path` look promising.

5. Make a map of all highways marked as `track` or `footpath` or `path`. This does look better than our previous map, but is still not OK for our purposes.

6. I now used OpenStreetMap to search on other types of highway or connections. Now I know that right after the Aclo and the Reitdiep there is some small path (not the "jaagpad" which is right at the side of the Reitdiep.) By clicking on it and querying, I discovered that this path is known as Relation 12-59. Look this up on OpenStreetMap (ask ChatGPT how to do this). The tags tell me that this has a `route` as a `type`, and then it is tagged as `hiking`. Then by clicking on `route`, I found that there are `hiking` and `foot` routes.

7. Make a map based on `hiking-foot-groningen.pbf` to show the `hiking` and `foot` routes. That will look much better.

Optional:

1. While playing with this problem, I also included `residential` and `footway` highways, but that made the map really crowded, so I removed these again. However, perhaps you find this interesting.
2. Plot all cycle paths in the province of Groningen.
3. Include motorways and trunks, and color them red, so that is is clear on the map that we want to avoid paths that lie close to noisy roads.
4. Compute the total amount of km of sand path (ways in OpenStreetMap that are marked as `highway=track`, `surface=fine_gravel`, etc) in the province of Groningen.

4.2.3 Plot shortest path from A to B

1. I selected two points in OpenStreetMap as start and end and looked up their node coordinates. These are my points: We start at `leens = np.array([[53.3615233, 6.3932052]])` and and end at `gn_station = np.array([[53.2113213, 6.5656461]]).` Ask ChatGPT how to plot a node by means of its coordinates on OpenStreetMap.
2. Now these start and end point do not necessarily lie on one of the ways that make up our graph with hiking and foot routes. There is a dumb way to find the nearest node of our graph, and a smart way that uses `BallTree` of `sklearn`. Use this data structure to help find the node in the routes nearest to the start. Then do the same for the end node.
3. Make a graph with `networkx` of all the hiking and foot routes. Add a `weight` attribute to each edge, corresponding to the length of each edge.
4. Compute the shortest path from start to end.
5. Plot the shortest path on an html page.
6. Compute the length of this shortest path.
7. Optional: You can try to export your path to a `gpx` file. Then mail it to yourself (or some other trick to get the file on your phone). I use `https://organicmaps.app/` to open the track.

Optional:

1. Compute the statistics of the surfaces of the shortest path, so total km over track, etc.
2. What surface of the Netherlands lies within 2 km of a motorway, trunk or primary road?
3. Very hard (nearly impossible) challenge: What path from Pieterburen to the Pietersberg maximizes the fraction of km walked on tracks?

5 Tutorial 5

5.1 Toyota, ridge and lasso

For this part you need the notes I made available on Brightspace on descent methods for Ridge, Lasso and Neural networks.

1. Implement coordinate descent for Lasso to predict the price as a function of the *numerical features* of the Toyotas.
2. Implement Ridge regression by solving $(X'X + \lambda)\beta = X'y$ by using `np.linalg.solve`. Include only some *numerical features* of X.
3. Use the implementation of `sklearn` of ridge regression to compute β^* for the same features as in the previous step. Compare the results.
4. Implement gradient descent for ridge regression to compute β^*. Hint, the lecture notes contain the basic algorithm in python. Compare the β with the previous step. (I had quite some trouble with getting this to work. If η is not quite OK, gradient descent does not work well. It is also important to apply scaling, i.e., subtract the mean of each feature and divide by the std. Hopefully you find out that building even this very simple example from scratch is not entirely straightforward. And then, hopefully you'll memorize that getting neural networks to learn efficiently is extremely hard.)
5. Use ChatGPT to provide the code to use the neural networks of `sklearn` to predict the prices. ChatGPT gave me a 2 layer network. Then I played a bit with the number of nodes per layer; I even tried layers with just one node, just to see what would happen. Then I tried a one-layer network with 10 nodes. That gave bad results. However, increasing to 100 gave real good results again. Please play a bit too. (Again, realize it is not our goal to become good car sales people; our goal is to understand a bit how to train neural networks and see what the effects are of some hyper parameters.) Are the some settings that train faster, or slower?
6. Compare the root mean square errors of the neural network and random forests. (Of course, if you choose ridiculous hyper parameters you can always make one type of method look bad. So, you should look for hyper parameters that are reasonable, and then compare.)
7. What do you learn from this comparison? Do the relatively simmple tools perform well?

Optional challenges.

1. Use boosting of sklearn to predict prices. (I have not done this myself, so at present I don't know whether this is hard or not. I suspect that with ChatGPT it's easy.)
2. Adapt the code from the book of Michael Nielsen, `https://github.com/minseoksong23/neural-networks-and-deep-learning-python3`, to make a neural network to predict the price of a Toyota as a function of the *numerical features*. You can see how you fare with neural networks by comparing the results to the predictions based on Lasso, Ridge, random forests, and the neural network of `sklearn`. Do not hesitate to consult ChatGPT whenever you can.

5.2 Geography

We are going to compute a shortest path along all bridges in the city of Groningen with a heuristic. You know of course the traveling salesman problem. Now we let the person find its way along the bridges.

This exercise has some aims.

1. Data extraction and (pre)processing, and database management: Show how easy it is to save intermediate computational steps to a sql database.
2. Use simple heuristics to find approximate solutions for a very common OR problem.
3. Data visualization/presentation and analysis: Plot the solution to that it becomes possible to interpret it and communicate about its properties. In other words, make a nice graph.
4. Understand some cool algorithms.
5. Use tricks to circumvent lots of unnecessary computations. (See the distance functions below.)

Here are the steps:

1. Use `osmium` (and ChatGPT) to read all bridges from the file `groningen_city_area.pbf`, see Brightspace. I obtain 507 bridges.
2. For the next steps it is practical to write the node ids, lat's and lon's, to a sqlite database. So, after reading the data with the `osmium` handler, write the node info to a `sqlite` database with name `bridges.sql` and with the table name `node`.
3. Use the database and `folium` to plot the bridge points, for instance as circle markers. (I call this file `plot_bridges.py`.)
4. Use the nearest neighbor code from `https://github.com/norvig/pytudes/blob/main/ipynb/TSP.ipynb` to make a nearest neighbor optimal path. (If you find this hard to adapt, use my code below.) BTW, the code of Norvig is extremely good, hence extremely useful to study well.
5. Make a function to plot the path with `folium`.
6. Adapt the 2 opt heuristic as explained in `https://github.com/norvig/pytudes/blob/main/ipynb/TSP.ipynb` to make the path shorter and remove intersections. If you find this hard, check my code below.
7. Plot the adapted path.

Before presenting the code, here are some optional challenges:

1. Use distance that do not assume that the earth is flat.
2. Find the path that is shortest up to 2opt that follows the streets instead of through air.

The code about the tsp is adapted from Norvig. I included two functions to compute the distance between two points. The haversine function returns the distance, through air, between two points that are close enough that the earth appears flat. This function is a bit slow, thought, because of the conversion to radians. However, for the algorithm, we don't need the distance in meters. The `distance_proxy` function works just as well for our purposes, but is much faster.

For the tour length computation I do, of course, use the correct distance function because for humans (us), we need a result in a unit we understand. Here is how I call the distance functions.

```python
distance = lambda m, n: distance_proxy(m, n, node_dict)
h_distance = lambda m, n: haversine_distance(m, n, node_dict)

nn_tour = nearest_tsp(node_dict.keys(), distance)
```

code/tsp_bridges/nearest_neighbor_functions.py

```python
import numpy as np

earth_radius = 6371000  # meters
cos_phi2 = np.cos(np.radians(53)) ** 2


def distance_proxy(node1, node2, node_dict):
    lat1, lon1 = node_dict[node1]
    lat2, lon2 = node_dict[node2]
    return (cos_phi2 * (lat1 - lat2) ** 2 + (lon1 - lon2) ** 2) ** 0.5


def haversine_distance(node1, node2, node_dict):
    lat1, lon1 = node_dict[node1]
    lat2, lon2 = node_dict[node2]
    lat1_rad, lon1_rad = np.radians(lat1), np.radians(lon1)
    lat2_rad, lon2_rad = np.radians(lat2), np.radians(lon2)

    dlat = lat2_rad - lat1_rad
    dlon = lon2_rad - lon1_rad
    return earth_radius * (dlat**2 * cos_phi2 + dlon**2) ** 0.5


def tour_length(tour, distance):
    "The total distances of each link in the tour, including the link from last back to first."
    return sum(distance(tour[i], tour[i - 1]) for i in range(len(tour)))


def nearest_neighbor(A, unvisited, distance):
    """Find the nearest unvisited node to node A by calculating distances on the fly using the node_dict."""
    return min(unvisited, key=lambda C: distance(A, C))


def first(collection):
    """The first element of a collection."""
    return next(iter(collection))


def nearest_tsp(nodes, distance, start=None):
    """Create a partial tour that initially is just the start node.
    At each step, extend the partial tour to the nearest unvisited neighbor
    of the last node in the partial tour, while there are unvisited nodes remaining.
    """

    start = start or first(nodes)
    tour = [start]
    unvisited = set(nodes) - {start}

    def extend_to(C):
        tour.append(C)
        unvisited.remove(C)

    while unvisited:
        next_node = nearest_neighbor(tour[-1], unvisited, distance)
        extend_to(next_node)

    return tour
```

I replaced Norvig's function to compute subsegments by `itertools.combinations`. The latter works somewhat slower, but is easier to understand.

code/tsp_bridges/opt2_functions.py

```
from itertools import combinations


def opt2(tour, distance):
    "Perform 2-opt segment reversals to optimize tour."
    changed = False
    for i, j in combinations(range(len(tour)), 2):
        if is_reversal_improvement(tour, i, j, distance):
            tour[i:j] = reversed(tour[i:j])
            changed = True
    return tour if not changed else opt2(tour, distance)


def is_reversal_improvement(tour, i, j, distance):
    "Would reversing the segment `tour[i:j]` make the tour shorter?"
    # Given tour [...A,B--C,D...], would reversing B--C make the tour shorter?
    # A, B, C, D = tour[i - 1], tour[i], tour[j - 1], tour[j % len(tour)]
    A, B, C, D = tour[i - 1], tour[i], tour[j - 1], tour[j]
    return distance(A, B) + distance(C, D) > distance(A, C) + distance(B, D)
```