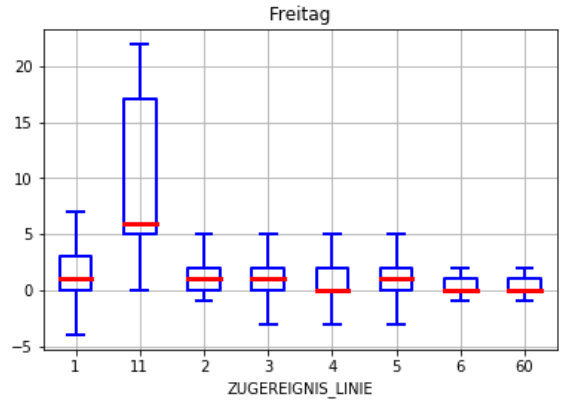
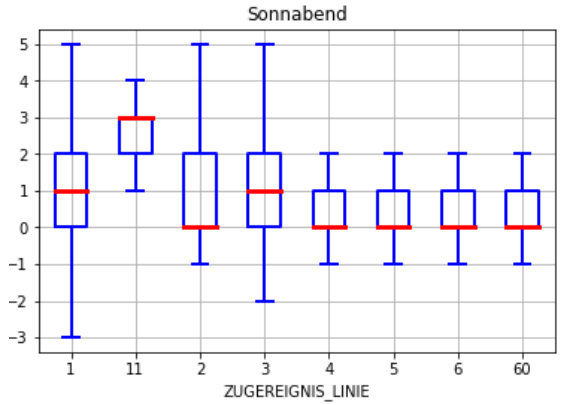
Part I: Introduction

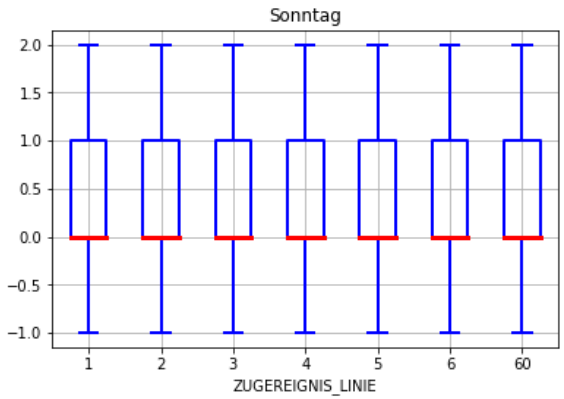
Focus on Pre-processing, Feature Engineering and Machine Learning

Stuttgart S-Bahn Example

(Lorenzo Zanon, Li Zhong, and Dennis Hoppe, HLRS; Oleksandr Shcherbarkov, HLRS)







Acknowledgements

Origin of this example:

Project “Simulierte Welte”, a cooperation between

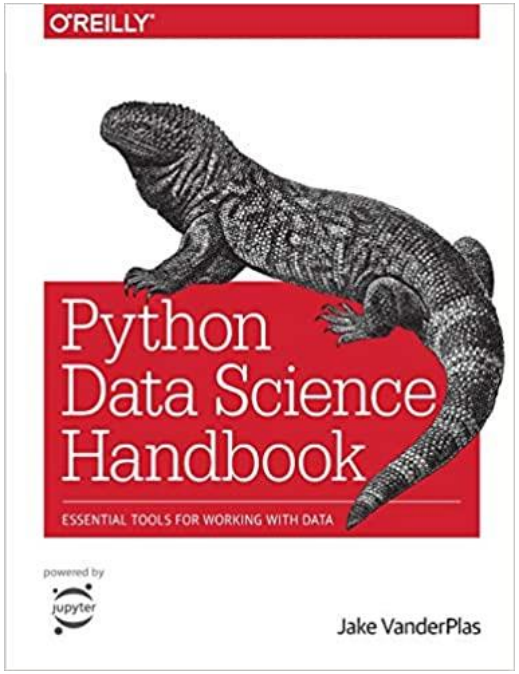
- HLRS
- Steinbuch Centre for Computing (SCC)
- Stuttgart Research Center for Interdisciplinary Risk and Innovation Studies (ZIRIUS)

References

- Link to websites
- Unless specified otherwise, **pages** indications always refer to:

Python Data Science Handbook

(Jake VanderPlas, O'Reilly 2017)



Free online:

https://jakevdp.github.io/PythonDataScienceHandbook/

Purpose: HPC + Data Science

Connect Data Science with High Performance Computing:

- HPC in Germany:

https://www.hlrs.de/about-us/hpc-in-germany/

- Gauss Centre for Supercomputing:

https://www.gauss-centre.eu/

- HLRS Systems:

https://www.hlrs.de/systems/

General objectives of the example

- How to deal with a given set of data?

-> **data manipulation**, preparation

- ML **pipeline**: Train and Validate, Test
- ML performance optimization

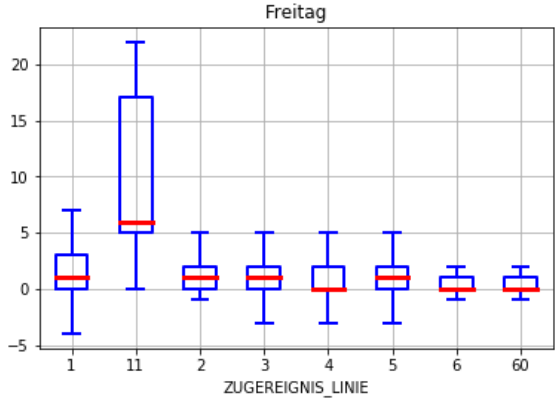
Moreover, with **these** data…

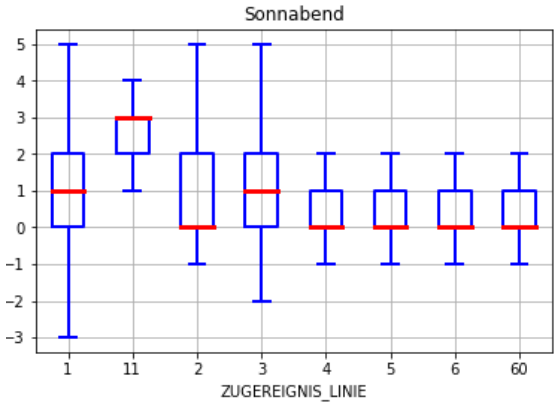
- Can I improve my travel experience in the Stuttgart S-Bahn thanks to ML/DL?
  1. Predict the S-Bahn delays with ML and DL with **minute-precision**: is that feasible?
  2. Is my train going to be **late at all**?

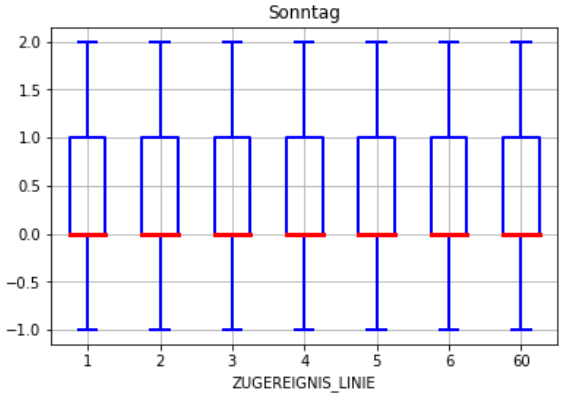
Explorative analysis / Manipulation

- Use Python & tools to perform an explorative analysis:
  - make a first interpretation
  - extract first simple statistical values for the **delay**
- Problem:

Dataset is quite **sparse** and initial feature set is small!





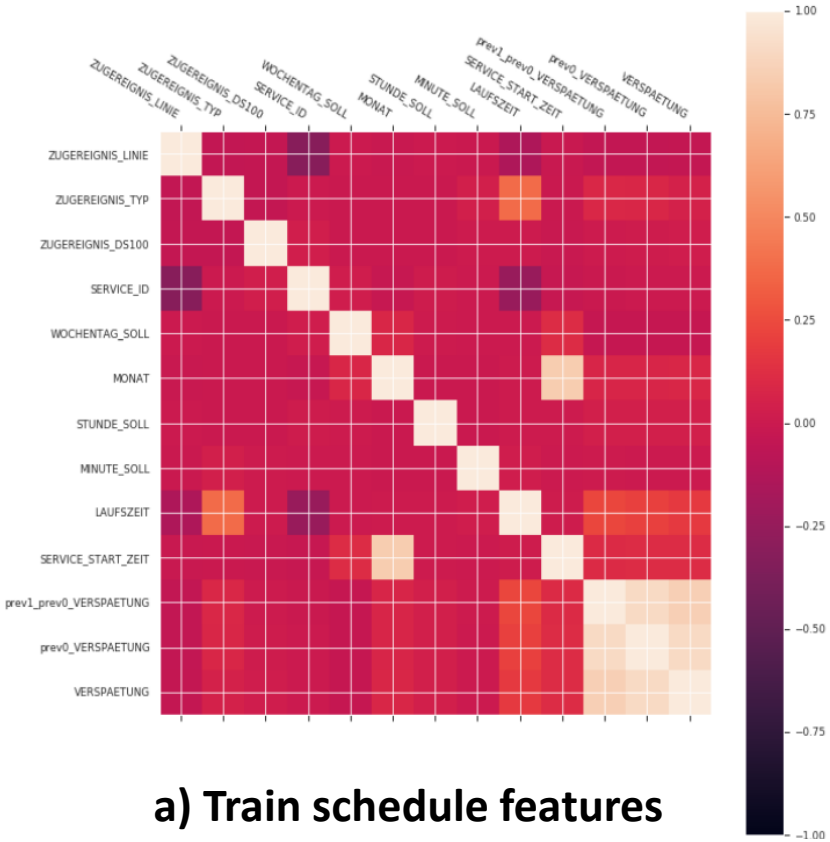


Pre-processing and Feature engineering

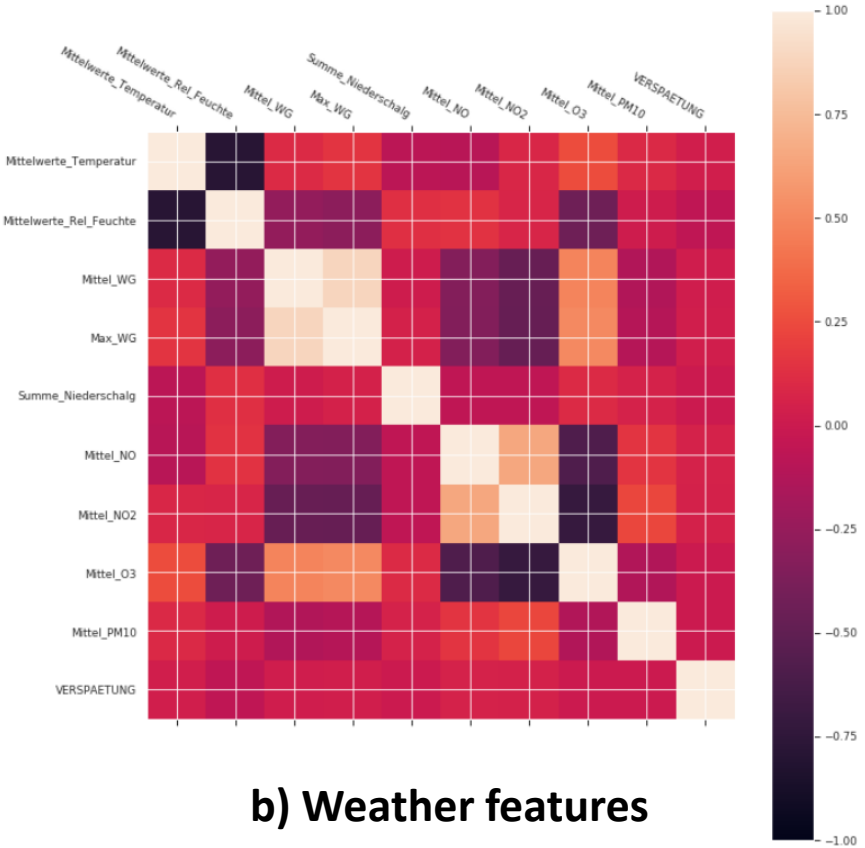
- Clear-up **noisy** data
- **Data augmentation** and **Data fusion**: We will see in data preparation
- Find out **linear** relations with a correlation map (**before** running the model)
- Single out un-necessary features which reduce the model performance (**after** having run the model)

Pre-processing and Feature engineering

Example of a Correlation Map (**before** running the model):



a) Train schedule features



b) Weather features

Predicting Train Delays with Machine Learning

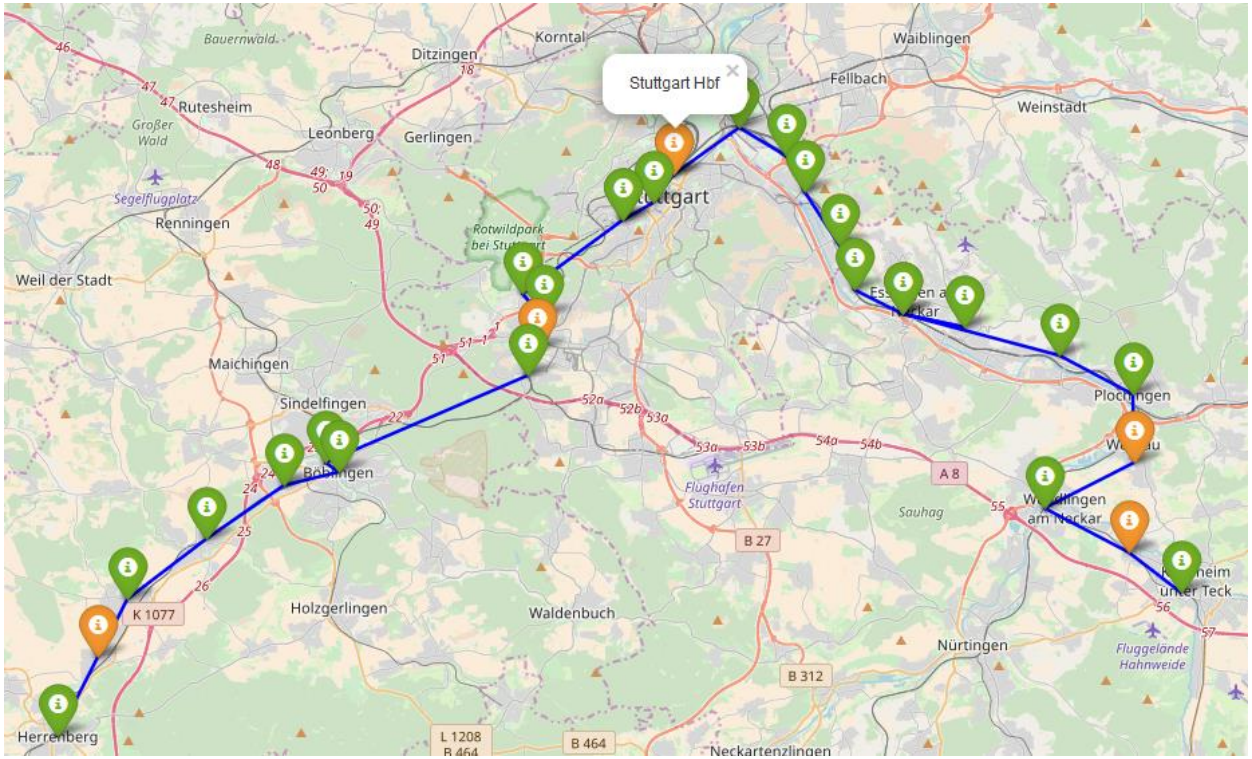
Supervised Learning: Set of features -> Label

- Predict **continuous** delay in minutes with high accuracy using (linear) regression models (p. 390) -> difficult task!
  - Causes: Undetected nonlinear relations, missing features,…
- Classification yields quite good results:
  - From a range of features we predict a **discrete** label (delay yes/no)
  - Tools: Random Forest as an ensemble of Decision Trees (p. 421)
- Outlook: More advanced (and expensive) regression models and classifiers such as Support Vector Machines (p. 405)

Example of classification result & visualization

- Accuracy of delay-classification of S1 at every station:
  - Green: >80%
  - Yellow: >50%

  (= coin flip! ☹)
  - Black: ☹



Predicting Train Delays with Deep Learning – Outlook

Initial DL tests in **this** example with Tensorflow and Keras:

- CNN and RNN for classification (cf. NVIDIA workshop)
- LSTM (Long Short Term Memory) (prediction)

For **this example**, DL results are not significantly better than ML and will not be discussed.

Predicting Train Delays: Post-processing and Optimization

… but how can we **change the results** to improve the model?

- Choose different combinations of **features** (data science)

Outlook:

- Analyse the influence of ratio **training vs. test data** (learning curve)
- **Hyperparameters** (learning design):
  - Tune empirical parameters (e.g. number of training epochs)
  - More complex models could lead to better results or **overfitting**!
  - **No guideline** to tune hyperparameters
- Use a different **method** (modify the architecture)

Requirements for all practicals

Files that you will need **locally** can be found in:

https://fs.hlrs.de/people/zanon/

Username: testuser / Password: sbahn2020

- **slides_sbahn_all.pdf : THESE SLIDES, please open!**
- …_requirements.pdf : Already discussed (emails)
- vi_cheat_sheet.pdf
- S_Bahn_data : Data sources for the S-Bahn example
- … and more (we will see when needed)

Part II: The example step by step on a Jupyter Notebook

Log in and set up

HLRS **CPUs** for DataScience: Urika-GX:
https://kb.hlrs.de/platforms/index.php/Urika_GX
Two systems:

- **Gilgamesch**

Used by several HLRS partners in different projects.
48 Nodes in total (2 Login, 2 IO, 3 Service, **41 Compute Nodes**)

- **Enkidu**

Used for test and training.
16 Nodes in total (2 Login, 2 IO, 3 Service, **9 Compute Nodes**)
⇒36*9 = 324 cores / 60 users = 5.4
⇒ **We use** 5 cores / user (to avoid any crash!)

Log in and set up

PRACTICAL:

Log in in Enkidu as you learned (terminal, cmd window, putty …):

> ssh vdl1XXX@enkidu-login1.hlrs.de **-D 8080**

OR: > ssh enkidu-workshop **-D 8080**

Navigate to the folder:

> cd ./hpda-code/use-cases/sbahn/**scripts**

(**next slide** to recap)

Log in and set up

PRACTICAL:

Prerequisite: From your home directory:

> pwd

… to make sure to be in your home directory:

/home/users/vdl1XXX/

… create these subfolders (for consistency with my path):

> mkdir hpda-code

> cd ./hpda-code

> mkdir use-cases

> cd ./use-cases

> mkdir sbahn

> cd ./sbahn

> mkdir JNotebooks ; mkdir scripts

> cd ./**scripts**

Please **stay** in this folder when you are done.

Log in and set up

PRACTICAL (optional): Load needed modules:

> module load tools/mesos

More about **tools**:

> minfo

Provides **minfo** (view compute resources) and **mreserve** (reserve resources) commands to work with Mesos.

[oleksandr.shcherbakov@hlrs.de]

Log in and set up

PRACTICAL:

Copy Jupyter Notebooks and Scripts to your local folder. In your folder **scripts**:

> scp -r /mnt/lustre/hpclzano/scriptsTN/* ./

> cd ../JNotebooks

> scp -r /mnt/lustre/hpclzano/JNotebooksTN/* ./

/ is necessary to avoid hidden files!*

If any issue: JNotebooks available (display only) in:

https://fs.hlrs.de/people/zanon/JNotebooksTN_html/

Log in and set up

PRACTICAL:

Source data are stored in **Lustre**, a **storage architecture for clusters**:

http://doc.lustre.org/lustre_manual.xhtml#part.intro

Have a look with:

> **ll /mnt/lustre/hpclzano/sbahn_data**

- These are .csv and .xls files that are read-in in the code.
- You can find and download some of these files at:

https://fs.hlrs.de/people/zanon/S_Bahn_data

Log in and set up

What do the **data represent**:

20170901-20171019_Alle_**Soll**ereignisse_S-Bahn_Stuttgart.csv

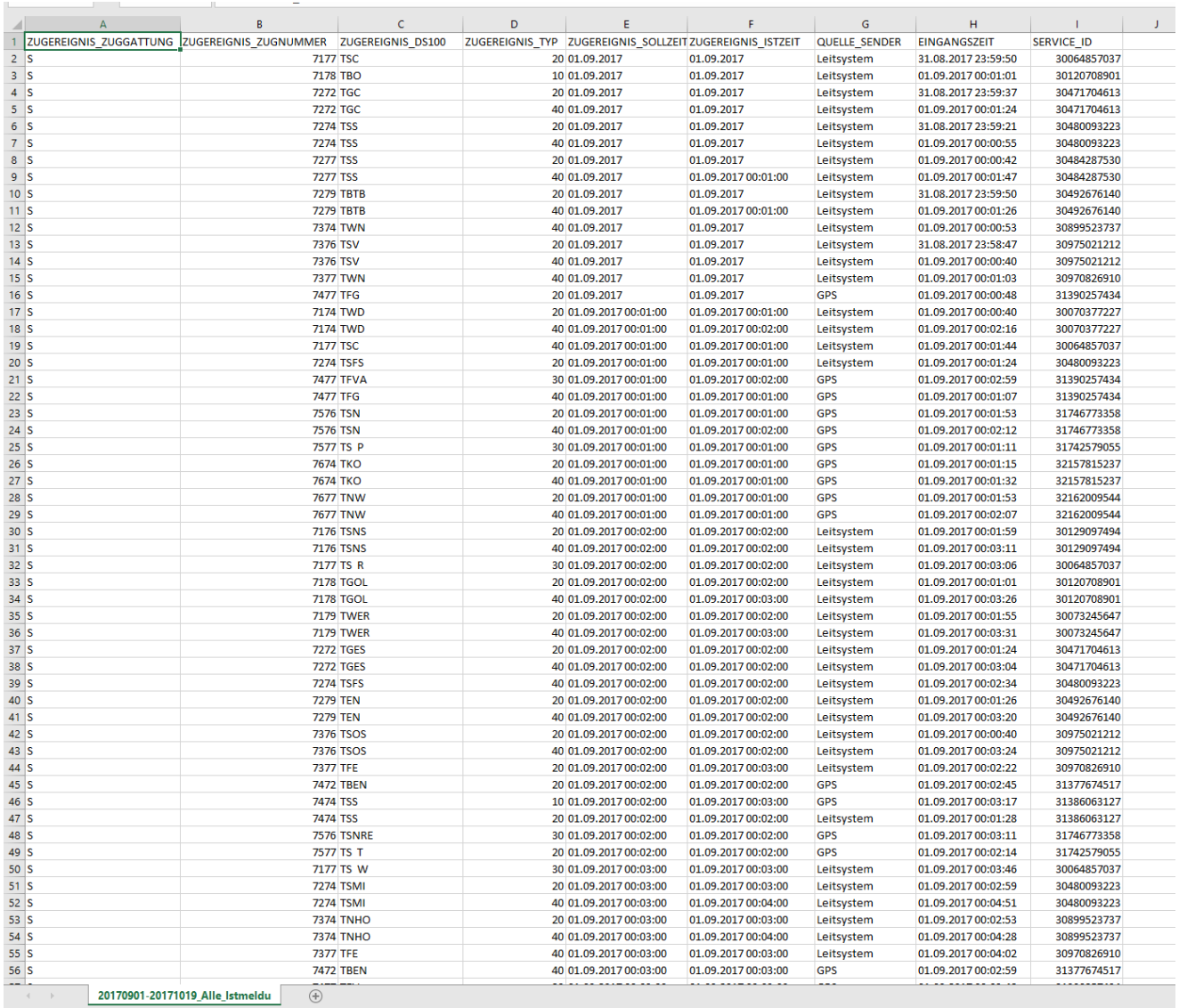
20170901-20171019_Alle_**Ist**meldungen_S-Bahn_Stuttgart.csv

- All Stuttgart S-Bahn journeys over 50 days months
- 1639 journeys are analyzed (complete dataset)
- Train events:
  - **Scheduled timetable** („Soll-events“)
  - **Actual times** (“Ist-events “)

  with many additional information (we will see during the exercise)

Have a look in:

https://fs.hlrs.de/people/zanon/S_Bahn_data/20170901-20171019_Alle_Istmeldungen_S-Bahn_Stuttgart.csv



	A	B	C	D	E	F	G	H	I	J
1	ZUGEREIGNIS_ZUGGATTUNG	ZUGEREIGNIS_ZUGNUMMER	ZUGEREIGNIS_DS100	ZUGEREIGNIS_TYP	ZUGEREIGNIS_SOLLZEIT	ZUGEREIGNIS_ISTZEIT	QUELLE_SENDER	EINGANGSZEIT	SERVICE_ID	
2	S	7177	TSC	20	01.09.2017	01.09.2017	Leitsystem	31.08.2017 23:59:50	30064857037	
3	S	7178	TBO	10	01.09.2017	01.09.2017	Leitsystem	01.09.2017 00:01:01	30120708901	
4	S	7272	TGC	20	01.09.2017	01.09.2017	Leitsystem	31.08.2017 23:59:37	30471704613	
5	S	7272	TGC	40	01.09.2017	01.09.2017	Leitsystem	01.09.2017 00:01:24	30471704613	
6	S	7274	TSS	20	01.09.2017	01.09.2017	Leitsystem	31.08.2017 23:59:21	30480093223	
7	S	7274	TSS	40	01.09.2017	01.09.2017	Leitsystem	01.09.2017 00:00:55	30480093223	
8	S	7277	TSS	20	01.09.2017	01.09.2017	Leitsystem	01.09.2017 00:00:42	30484287530	
9	S	7277	TSS	40	01.09.2017	01.09.2017 00:01:00	Leitsystem	01.09.2017 00:01:47	30484287530	
10	S	7279	TBTB	20	01.09.2017	01.09.2017	Leitsystem	31.08.2017 23:59:50	30492676140	
11	S	7279	TBTB	40	01.09.2017	01.09.2017 00:01:00	Leitsystem	01.09.2017 00:01:26	30492676140	
12	S	7374	TWN	40	01.09.2017	01.09.2017	Leitsystem	01.09.2017 00:00:53	30899523737	
13	S	7376	TSV	20	01.09.2017	01.09.2017	Leitsystem	31.08.2017 23:58:47	30975021212	
14	S	7376	TSV	40	01.09.2017	01.09.2017	Leitsystem	01.09.2017 00:00:40	30975021212	
15	S	7377	TWN	40	01.09.2017	01.09.2017	Leitsystem	01.09.2017 00:01:03	30970826910	
16	S	7477	TFG	20	01.09.2017	01.09.2017	GPS	01.09.2017 00:00:48	31390257434	
17	S	7174	TWD	20	01.09.2017 00:01:00	01.09.2017 00:01:00	Leitsystem	01.09.2017 00:00:40	30070377227	
18	S	7174	TWD	40	01.09.2017 00:01:00	01.09.2017 00:02:00	Leitsystem	01.09.2017 00:02:16	30070377227	
19	S	7177	TSC	40	01.09.2017 00:01:00	01.09.2017 00:01:00	Leitsystem	01.09.2017 00:01:44	30064857037	
20	S	7274	TSFS	20	01.09.2017 00:01:00	01.09.2017 00:01:00	Leitsystem	01.09.2017 00:01:24	30480093223	
21	S	7477	TFVA	30	01.09.2017 00:01:00	01.09.2017 00:02:00	GPS	01.09.2017 00:02:59	31390257434	
22	S	7477	TFG	40	01.09.2017 00:01:00	01.09.2017 00:01:00	GPS	01.09.2017 00:01:07	31390257434	
23	S	7576	TSN	20	01.09.2017 00:01:00	01.09.2017 00:01:00	GPS	01.09.2017 00:01:53	31746773358	
24	S	7576	TSN	40	01.09.2017 00:01:00	01.09.2017 00:02:00	GPS	01.09.2017 00:02:12	31746773358	
25	S	7577	TS P	30	01.09.2017 00:01:00	01.09.2017 00:01:00	GPS	01.09.2017 00:01:11	31742579055	
26	S	7674	TKO	20	01.09.2017 00:01:00	01.09.2017 00:01:00	GPS	01.09.2017 00:01:15	32157815237	
27	S	7674	TKO	40	01.09.2017 00:01:00	01.09.2017 00:01:00	GPS	01.09.2017 00:01:32	32157815237	
28	S	7677	TNW	20	01.09.2017 00:01:00	01.09.2017 00:01:00	GPS	01.09.2017 00:01:53	32162009544	
29	S	7677	TNW	40	01.09.2017 00:01:00	01.09.2017 00:01:00	GPS	01.09.2017 00:02:07	32162009544	
30	S	7176	TSNS	20	01.09.2017 00:02:00	01.09.2017 00:02:00	Leitsystem	01.09.2017 00:01:59	30129097494	
31	S	7176	TSNS	40	01.09.2017 00:02:00	01.09.2017 00:02:00	Leitsystem	01.09.2017 00:03:11	30129097494	
32	S	7177	TS R	30	01.09.2017 00:02:00	01.09.2017 00:02:00	Leitsystem	01.09.2017 00:03:06	30064857037	
33	S	7178	TGOL	20	01.09.2017 00:02:00	01.09.2017 00:02:00	Leitsystem	01.09.2017 00:01:01	30120708901	
34	S	7178	TGOL	40	01.09.2017 00:02:00	01.09.2017 00:03:00	Leitsystem	01.09.2017 00:03:26	30120708901	
35	S	7179	TWER	20	01.09.2017 00:02:00	01.09.2017 00:02:00	Leitsystem	01.09.2017 00:01:55	30073245647	
36	S	7179	TWER	40	01.09.2017 00:02:00	01.09.2017 00:03:00	Leitsystem	01.09.2017 00:03:31	30073245647	
37	S	7272	TGES	20	01.09.2017 00:02:00	01.09.2017 00:02:00	Leitsystem	01.09.2017 00:01:24	30471704613	
38	S	7272	TGES	40	01.09.2017 00:02:00	01.09.2017 00:02:00	Leitsystem	01.09.2017 00:03:04	30471704613	
39	S	7274	TSFS	40	01.09.2017 00:02:00	01.09.2017 00:02:00	Leitsystem	01.09.2017 00:02:34	30480093223	
40	S	7279	TEN	20	01.09.2017 00:02:00	01.09.2017 00:02:00	Leitsystem	01.09.2017 00:01:26	30492676140	
41	S	7279	TEN	40	01.09.2017 00:02:00	01.09.2017 00:02:00	Leitsystem	01.09.2017 00:03:20	30492676140	
42	S	7376	TSOS	20	01.09.2017 00:02:00	01.09.2017 00:02:00	Leitsystem	01.09.2017 00:00:40	30975021212	
43	S	7376	TSOS	40	01.09.2017 00:02:00	01.09.2017 00:02:00	Leitsystem	01.09.2017 00:03:24	30975021212	
44	S	7377	TFE	20	01.09.2017 00:02:00	01.09.2017 00:03:00	Leitsystem	01.09.2017 00:02:22	30970826910	
45	S	7472	TBEN	20	01.09.2017 00:02:00	01.09.2017 00:02:00	GPS	01.09.2017 00:02:45	31377674517	
46	S	7474	TSS	10	01.09.2017 00:02:00	01.09.2017 00:03:00	GPS	01.09.2017 00:03:17	31386063127	
47	S	7474	TSS	20	01.09.2017 00:02:00	01.09.2017 00:02:00	Leitsystem	01.09.2017 00:01:28	31386063127	
48	S	7576	TSNRE	30	01.09.2017 00:02:00	01.09.2017 00:02:00	GPS	01.09.2017 00:03:11	31746773358	
49	S	7577	TS T	20	01.09.2017 00:02:00	01.09.2017 00:02:00	GPS	01.09.2017 00:02:14	31742579055	
50	S	7177	TS W	30	01.09.2017 00:03:00	01.09.2017 00:03:00	Leitsystem	01.09.2017 00:03:46	30064857037	
51	S	7274	TSMI	20	01.09.2017 00:03:00	01.09.2017 00:03:00	Leitsystem	01.09.2017 00:02:59	30480093223	
52	S	7274	TSMI	40	01.09.2017 00:03:00	01.09.2017 00:04:00	Leitsystem	01.09.2017 00:04:51	30480093223	
53	S	7374	TNHO	20	01.09.2017 00:03:00	01.09.2017 00:03:00	Leitsystem	01.09.2017 00:02:53	30899523737	
54	S	7374	TNHO	40	01.09.2017 00:03:00	01.09.2017 00:04:00	Leitsystem	01.09.2017 00:04:28	30899523737	
55	S	7377	TFE	40	01.09.2017 00:03:00	01.09.2017 00:03:00	Leitsystem	01.09.2017 00:04:02	30970826910	
56	S	7472	TBEN	40	01.09.2017 00:03:00	01.09.2017 00:03:00	GPS	01.09.2017 00:02:59	31377674517	

20170901-20171019_Alle_Istmeldu

Log in and set up

Bahnhofsdaten.csv

https://fs.hlrs.de/people/zanon/S_Bahn_data/Bahnhofsdaten.csv

- All Stuttgart S-Bahn stations (numeric ID, DS100 code, full name)

S-Mitte-SZ-Halbstd-Werte_2017.xls

https://fs.hlrs.de/people/zanon/S_Bahn_data/S-Mitte-SZ-Halbstd-Werte_2017.xls

- **Weather** and **fine particle** data at the „Schwabenzentrum“ for the whole year 2017, measured every 30 minutes
- Up-to-date data are made publicly accessible by the Deutsche Bahn:

https://data.deutschebahn.com/dataset/data-s-bahn-stuttgart-ris-archiv-daten

Messstation "Schwabenzentrum" (Amt für Umweltschutz, Abt. Stadtklimatologie)
(Stuttgart-Mitte, Ecke Tor-/ Hauptstätter Straße)

Halbstunden-Mittel-Werte (bzw. Max- und Min-Werte) sämtlicher Komponenten im Dezember 2017

Datum	Uhrzeit	Mittelwerte Temperatur (°C)	Maxwerte Temperatur (°C)	Minwerte Temperatur (°C)	Mittelwerte Rel. Feuchte (%)	Mittel WG (m/s)	Max WG (m/s)	Mittel WR (Grad)	Mittel Druck (hPa)	Summe Niederschlag (l/m²)	Mittel Globalstr. (W/m²)	Mittel Str.-Bilanz (W/m²)	Mittel UVA-Str. (W/m²)	Mittel UVB-Str. (W/m²)	Mittel NO (µg/m³)	Mittel NO2 (µg/m³)	Mittel O3 (µg/m³)	Mittel PM10 (µg/m³)	Mittel PM2,5 (µg/m³)
01/12/2017	00:30	1,2	1,7	0,9	82,7	0,7	1,4	243,1	975,4	0,00	0,0	-46,5	2,00	0,034	3	34	11	12	10
01/12/2017	01:00	1,1	1,7	0,9	83,4	0,4	1,2	202,6	975,2	0,00	0,0	-40,1	2,00	0,034	4	31	11	13	11
01/12/2017	01:30	1,7	2,3	1,1	81,1	0,5	1,6	178,9	975,3	0,00	0,0	-23,0	1,96	0,034	3	30	12	13	11
01/12/2017	02:00	1,3	1,7	1,1	83,0	0,7	1,3	237,7	975,3	0,00	0,0	-23,6	1,95	0,034	2	31	10	14	12
01/12/2017	02:30	1,4	2,0	0,9	83,6	0,4	0,9	169,6	975,4	0,00	0,0	-25,6	1,95	0,033	5	34	7	13	12
01/12/2017	03:00	1,3	1,6	1,1	81,6	0,6	1,8	163,3	975,6	0,00	0,0	-18,6	1,94	0,034	3	27	13	13	12
01/12/2017	03:30	1,5	2,2	1,2	79,3	0,5	1,3	260,8	975,6	0,00	0,0	-16,3	1,93	0,033	1	24	18	14	13
01/12/2017	04:00	2,2	2,9	1,5	78,3	0,7	1,2	74,8	975,8	0,00	0,0	-16,4	1,92	0,034	6	31	12	14	12
01/12/2017	04:30	2,5	3,1	1,6	78,9	0,5	1,3	56,3	976,0	0,00	0,0	-17,2	1,90	0,034	9	36	8	14	12
01/12/2017	05:00	2,2	2,9	1,4	80,5	0,8	1,2	43,9	976,2	0,00	0,0	-26,8	1,88	0,033	10	38	4	19	17
01/12/2017	05:30	2,1	2,5	1,4	83,7	0,9	1,5	27,9	976,5	0,10	0,0	-34,6	1,91	0,033	21	42	3	23	20
01/12/2017	06:00	2,0	2,2	1,7	85,0	0,8	2,1	19,7	976,8	0,20	0,0	-20,7	1,91	0,033	12	37	3	21	19
01/12/2017	06:30	2,0	2,4	1,6	85,9	0,6	0,9	11,7	977,2	0,20	0,0	-13,7	1,91	0,033	27	40	5	19	17
01/12/2017	07:00	1,4	2,0	0,9	86,2	0,6	2,0	316,3	977,7	0,20	0,0	-13,8	1,91	0,033	4	21	21	18	15
01/12/2017	07:30	1,1	1,9	0,7	85,7	0,5	1,9	310,3	978,0	0,00	0,0	-15,1	1,94	0,033	3	19	26	14	12
01/12/2017	08:00	1,2	1,5	0,9	82,9	1,2	1,4	276,3	978,3	0,00	0,0	-14,9	1,99	0,033	3	20	27	13	11
01/12/2017	08:30	1,6	2,1	1,1	82,5	0,5	1,4	335,2	978,7	0,10	3,8	-16,1	2,35	0,035	5	25	24	12	11
01/12/2017	09:00	1,9	2,3	1,4	82,2	0,7	1,0	2,5	979,0	0,00	13,6	-15,6	3,32	0,040	14	33	19	12	11
01/12/2017	09:30	1,6	2,4	1,1	81,4	0,8	0,9	292,4	979,5	0,00	20,5	-19,3	3,90	0,045	16	33	21	16	12
01/12/2017	10:00	1,4	1,7	1,1	81,8	0,8	1,3	263,4	979,9	0,00	32,9	-26,7	4,19	0,049	14	33	20	17	13
01/12/2017	10:30	1,7	2,2	1,2	80,7	1,1	1,4	268,0	980,3	0,00	50,1	-101,3	5,16	0,059	9	29	24	18	16
01/12/2017	11:00	2,2	2,6	1,7	78,3	0,7	1,8	317,0	980,7	0,00	78,2	13,3	6,22	0,070	7	25	27	18	16
01/12/2017	11:30	2,5	2,9	2,2	77,2	0,5	1,5	358,9	980,9	0,00	72,6	41,5	5,29	0,065	13	33	20	17	13
01/12/2017	12:00	2,7	3,4	2,3	77,4	0,4	1,4	59,8	980,9	0,00	202,9	136,3	6,68	0,073	31	45	13	20	15
01/12/2017	12:30	2,4	3,3	1,8	77,8	1,4	2,3	88,1	981,0	0,00	193,5	120,7	6,94	0,077	29	45	11	20	15
01/12/2017	13:00	2,1	2,5	1,8	78,9	1,5	2,8	83,5	981,1	0,00	73,8	33,7	4,55	0,059	23	46	11	21	16
01/12/2017	13:30	2,5	3,1	1,8	77,9	1,1	3,0	67,7	981,2	0,00	72,9	34,9	4,73	0,060	45	56	8	24	20
01/12/2017	14:00	2,3	3,1	1,8	78,0	1,5	2,6	46,1	981,5	0,00	25,1	3,7	3,26	0,047	29	51	9	23	20
01/12/2017	14:30	2,3	3,0	1,8	77,4	1,4	3,5	41,3	981,7	0,00	33,5	7,2	3,55	0,048	24	45	13	22	19
01/12/2017	15:00	1,9	2,3	1,6	79,6	1,4	3,2	65,9	981,9	0,00	31,8	-19,1	4,60	0,054	36	53	9	22	18
01/12/2017	15:30	1,7	2,1	1,4	78,4	1,3	2,9	71,8	982,2	0,00	26,9	-40,5	4,29	0,050	29	50	11	21	16
01/12/2017	16:00	1,7	2,2	1,3	77,7	1,0	1,9	100,6	982,7	0,00	20,6	-62,7	3,57	0,044	24	50	11	21	15
01/12/2017	16:30	2,0	2,9	1,4	76,2	0,9	1,2	95,8	983,2	0,00	9,6	-70,8	2,61	0,039	20	52	10	20	15
01/12/2017	17:00	2,1	2,6	1,4	76,6	0,7	1,1	93,9	983,4	0,00	0,6	-74,3	2,12	0,037	31	60	5	20	15
01/12/2017	17:30	2,1	2,6	1,6	76,6	0,6	0,7	93,4	983,7	0,00	0,0	-72,8	2,07	0,036	41	64	3	20	15
01/12/2017	18:00	1,8	2,6	1,2	78,1	0,6	1,5	82,9	983,9	0,00	0,0	-72,8	2,07	0,036	41	65	3	21	16
01/12/2017	18:30	1,7	2,3	1,1	79,5	0,4	0,9	118,2	984,2	0,00	0,0	-64,5	2,07	0,036	66	70	3	23	17
01/12/2017	19:00	1,7	2,3	0,9	81,4	0,4	1,2	41,5	984,5	0,00	0,0	-50,5	2,07	0,036	70	68	3	25	19
01/12/2017	19:30	1,8	2,3	1,3	82,1	0,7	1,3	63,7	984,9	0,00	0,0	-34,0	2,05	0,036	87	72	3	28	21
01/12/2017	20:00	2,0	2,3	1,6	82,0	0,5	1,4	22,7	985,2	0,00	0,0	-15,1	2,02	0,037	46	59	3	30	23
01/12/2017	20:30	2,2	2,5	1,7	81,4	0,7	1,5	22,9	985,4	0,00	0,0	-19,4	2,05	0,038	67	62	3	31	24
01/12/2017	21:00	2,1	2,5	1,6	81,2	0,7	1,5	17,7	985,6	0,00	0,0	-16,3	2,11	0,040	53	57	3	32	26
01/12/2017	21:30	2,2	2,6	1,7	81,1	1,2	2,4	18,6	985,9	0,00	0,0	-14,2	2,11	0,040	39	51	3	31	27
01/12/2017	22:00	2,1	2,6	1,8	81,3	0,8	1,5	13,2	986,1	0,00	0,0	-17,2	2,12	0,040	31	45	3	31	27
01/12/2017	22:30	2,1	2,5	1,4	81,3	0,8	1,3	26,9	986,3	0,00	0,0	-36,1	2,12	0,040	22	42	3	31	27
01/12/2017	23:00	2,0	2,5	1,7	81,3	1,2	2,1	21,3	986,6	0,00	0,0	-27,4	2,13	0,040	20	40	3	30	26
01/12/2017	23:30	2,0	2,3	1,6	81,4	1,2	2,5	25,8	986,8	0,00	0,0	-12,0	2,12	0,040	30	42	3	29	25
02/12/2017	00:00	2,1	2,5	1,7	81,8	1,2	2,0	19,8	987,1	0,00	0,0	-12,0	2,12	0,040	45	48	3	30	27
02/12/2017	00:30	2,2	2,6	1,9	81,3	1,4	2,0	20,3	987,2	0,00	0,0	-13,3	2,12	0,040	33	43	3	32	28
02/12/2017	01:00	1,8	2,2	1,6	82,2	1,4	2,7	21,5	987,4	0,00	0,0	-37,7	2,12	0,040	14	35	3	32	27
02/12/2017	01:30	1,7	2,0	1,4	81,3	0,9	3,0	17,9	987,6	0,00	0,0	-69,8	2,16	0,040	14	34	3	30	26
02/12/2017	02:00	1,0	1,6	0,5	83,1	1,1	2,2	59,0	987,8	0,00	0,0	-71,0	2,19	0,040	19	35	4	31	27
02/12/2017	02:30	0,9	1,3	0,5	84,0	0,9	1,4	73,6	988,1	0,00	0,0	-74,1	2,21	0,040	18	34	3	32	29
02/12/2017	03:00	1,0	1,6	0,5	83,9	0,7	1,1	45,1	988,3	0,00	0,0	-74,5	2,24	0,040	23	35	3	32	28
02/12/2017	03:30	0,3	0,9	-0,1	85,8	1,2	1,7	66,2	988,5	0,00	0,0	-74,3	2,26	0,040	29	37	3	32	28
02/12/2017	04:00	0,7	1,3	0,1	83,8	1,1	1,7	25,4	988,8	0,00	0,0	-69,1	2,26	0,040	29	37	3	32	28
02/12/2017	04:30	0,3	1,2	-0,3	83,8	0,8	2,6	53,0	989,0	0,00	0,0	-71,6	2,27	0,040	30	38	3	33	29
02/12/2017	05:00	0,3	0,8	-0,2	83,5	0,4	0,9	347,9	989,1	0,00	0,0	-70,8	2,27	0,040	22	38	3	32	29

März 2017 | April 2017 | Mai 2017 | Juni 2017 | Juli 2017 | Aug. 2017 | Sept. 2017 | Okt. 2017 | Nov. 2017 | **Dez. 2017** | Jahr 2017

Log in and set up – HDFS

- I/O files are (usually large) dataframes which are read from and written to through the Hadoop Distributed File System (**HDFS**).
- HDFS allows for distributed storage in an HDFS cluster (provided in the Urika-GX systems).

\+ : Speed of **parallel** execution

– : HDFS files cannot be handled as „normal“ files.

See:

https://hadoop.apache.org/docs/r2.8.5/hadoop-project-dist/hadoop-hdfs/HdfsUserGuide.html

Log in and set up – HDFS

Basic commands to perform operations on these files:

https://hadoop.apache.org/docs/r2.8.5/hadoop-project-dist/hadoop-hdfs/HDFSCommands.html

and

https://hadoop.apache.org/docs/r2.8.5/hadoop-project-dist/hadoop-common/FileSystemShell.html

PRACTICAL

In particular:

> **hadoop fs -ls**

displays your local hdfs files (none or Trash folder).

Log in and set up – HDFS

PRACTICAL - optional

Copy an HDFS file to your directory:

> hadoop fs **-cp** hdfs://192.168.0.1:8020/user/hpclzano/df_ts.csv df_ts2.csv

You can now see the file with (in chronological order)

> hadoop fs -ls **-t**

Check out more -ls options in https://hadoop.apache.org/docs/r2.8.5/hadoop-project-dist/hadoop-common/FileSystemShell.html

Log in and set up – HDFS

PRACTICAL - optional

Remove the df_ts2.csv file with: hadoop fs -rm **[name_of_file]**

> hadoop fs -rm df_ts2.csv

ERROR! since hdfs files are treated like **directories**.

In fact:

> hadoop fs -ls df_ts2.csv

… shows?

Log in and set up – HDFS

PRACTICAL - optional

… it shows the many smaller parts in which the file is now distributed, now as files:

…

-rw-r--r-- 3 hpclzano s29931 470189 2020-03-18 14:28 df_proper.csv/part-… .csv

…

To properly remove the file:

> hadoop fs -rm **-R** df_ts2.csv

Log in and set up – HDFS

Later in **Spark**, I/O is done in such a way:

df_test5_5.write.mode('overwrite')**.csv('df_test5_5.csv', header = True)**

df_train_classification = spark.read.option('header', True).option('inferSchema', True)\
.csv('df_train_classification.csv').cache()

Writing/reading are done locally by default to/from HDFS files!

Log in and set up – HDFS

Whenever you should read a file from my directory, the „solution prefix“ is added to the read-in command:

hdfs_solution = 'hdfs://192.168.0.1:8020/user/hpclzano/ …'

```
df_train_classification = spark.read.option('header', True).option('inferSchema', True)\
.csv(hdfs_solution+'df_train_classification.csv').cache()
```

(these lines are already correct)

Log in and set up – Data

To sum up, where do we **store** the data?

- Code and some output (plots): locally
- Source data: Lustre filesystem

  Read-only from: /mnt/lustre/hpclzano/sbahn_data
- Data I/O: HDFS, distributed file system

WORKFLOW S-Bahn example

- (Handlers.ipynb -> called inline)
- **DataManipulation.ipynb**
- DataVisualization.ipynb
- MachineLearning.ipynb
- PredictionVisualization.ipynb

Data preparation: Objectives

- Feature engineering: see next slides.
- Explorative Data Analysis
  - Data interpretation
  - Find out existing correlation through basic statistical tools (visualization part)

Data preparation: Objectives – skip for now

Feature engineering (pp. 375-382):

Vectorization of real-life data (-> ML **pipeline**)

Transform into numerical columns:

- Categorical data: several (sparse) columns of 0/1
- Text data: (weighted) word count
- (Image features)

Data preparation: Objectives

Feature engineering (pp. 375-382):

Derived features:

Manipulate the data to obtain different / more significant features that could have an impact on the model:

- Time information is also **weekday/-end**, (off-)peak hour, weekend, holiday, in which season,…
- Time information is even more useful if combined with **weather/fine-particles** data

⇒We will **merge** different Dataframes.

Data preparation: Objectives

Feature engineering (pp. 375-382):

- Combine time information: „Delay at station“ can be enriched with „Delay at previous station(s)“:

\+ : Much higher probability of identifying possible delay

– : This information is available only at very short notice!

Imputation of missing data / outliers:

- NaNs can be replaced e.g. with the mean of the remaining values
- … or we can also simply delete corresponding rows in the Dataframe

Data preparation tools

Programming language:

- Python:

https://docs.python.org/3/tutorial/

Main **tools** for data **manipulation**:

- Numpy

Efficient interface to store and operate on **dense data buffers**

https://numpy.org/doc/stable/reference/index.html

Data preparation tools

Main **tools** for data **manipulation**:

- Pandas

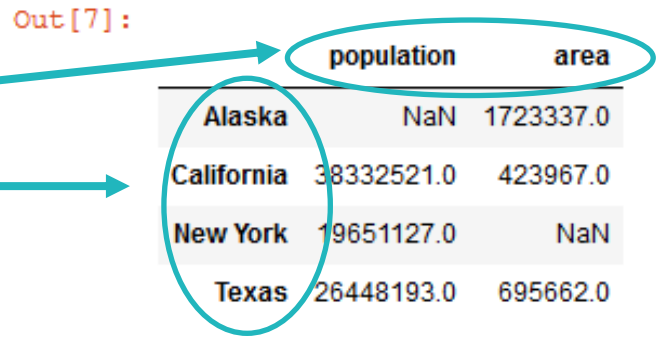
https://pandas.pydata.org

Data Manipulation with Pandas (**pp. 97-215**)

- Package built on Numpy, provides useful structures for ML such as Series and DataFrames
- **DataFrames** (DFs): a DF is a 2D Numpy object with flexible **row indices** and **column names**:

states.columns

states.index

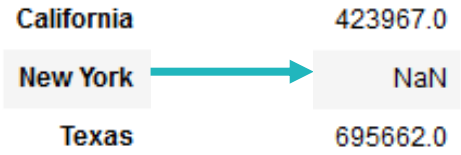


	population	area
Alaska	NaN	1723337.0
California	38332521.0	423967.0
New York	19651127.0	NaN
Texas	26448193.0	695662.0

Data preparation – Basic Structures

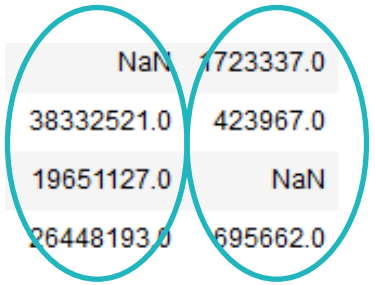
- Python **dictionary**:

Maps keys to values of **arbitrary** type



- Numpy **array (pp. 33-96):**

Multi-dimensional typed Python arrays.

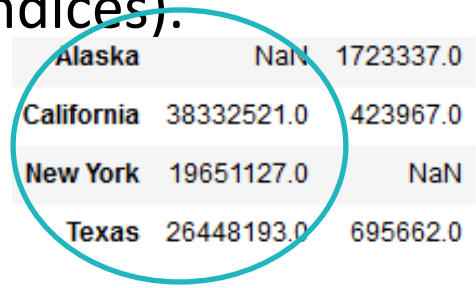


- Pandas **Series (pp. 97-110)**:

1D-arrays of indexed, **typed** data (with flexible indices).

More efficient than a dictionary!

DF: Aligned Series sharing the **same index**.



Data preparation tools

Main **tools** for data **manipulation**:

- Apache Spark

*"Spark is a **fast and general cluster computing system for Big Data**. It provides **high-level APIs** in Scala, Java, **Python**, and R, and an optimized engine that supports general computation graphs for data analysis. It also supports a rich set of higher-level tools including **Spark SQL for** SQL and **DataFrames, MLlib for machine learning**, GraphX for graph processing, and Spark Streaming for stream processing."*

https://pypi.org/project/pyspark/

Spark does **not** have official DL support.

For this reason, we work with **both** Pandas and Spark.

Data preparation with Enkidu

Disclaimer:

- You might find some bugs/inconsistences in the code
- Due to the circumstances, you might find the exercises very easy
- Feel free to experiment more after the course

Data preparation with Enkidu

PRACTICAL

> module load workshop (at every new log-in!)

> jupyter notebook

Copy link in the Firefox profile that you created.

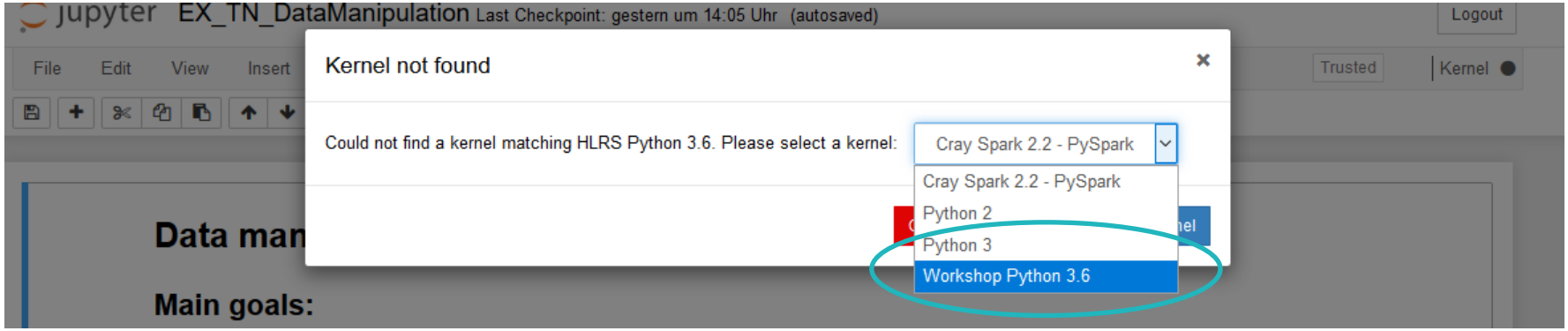
In JN, browse and open the Manipulation Notebook:

hpda-code/use-cases/sbahn/…

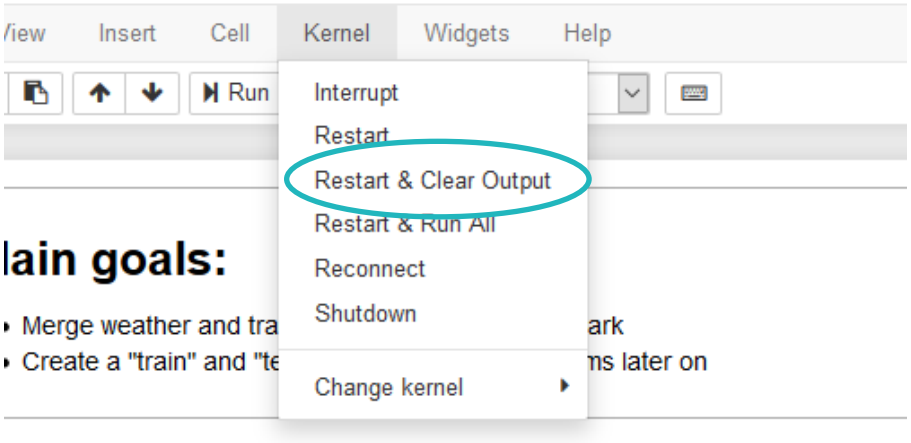
… /JNotebooks/EX_TN_DataManipulation.ipynb

Data preparation with Enkidu **EX_TN_DataManipulation**

Choose **Workshop Python 3.6** (supports „magics“ and more…)



Choose: **Kernel -> „Restart and clear output“** to start with a clean workspace.



Data preparation with Enkidu

Remark:

This Kernel „HLRS 3.6“: Spark 2.2.0

Only on Urika GX – **Gilgamesch**:

Option: To choose the „experimental“ kernel with Spark 2.4 (stable, but not tested on this example!).

Data preparation with Enkidu

Important

- From now on, your cmd window cannot be used any more!
- It is useful to have another **new cmd window**
- Execute a new Enkidu login (see requirements slides):

> ssh vdl1XXX@enkidu-login1.hlrs.de -D 8080

OR: > ssh enkidu-workshop -D 8080

> module load workshop

Data preparation with Enkidu

Important

Replace in the Notebook EX_TN_DataManipulation.ipynb:

username_local = "_____"

with your local username vdl1XXX (needed for I/O paths).

(run „id“ in an Enkidu cmd window)

____ = exercise, replace! (you will get an error otherwise)

Solution: EX_TN_**SOL**_DataManipulation.ipynb

Data preparation with Enkidu

Start executing – stay in slides

Important

- Run **one Notebook** at a time only!
- Go through the Notebook **at your pace** or follow me.

Data preparation with Enkidu

Also useful:

Open in your **browser** (another **window**) the reference pages to look up any function that we will need

Spark:

https://spark.apache.org/docs/latest/api/python/

Use „Quick search“ on the left.

Needed solution usually follows: „pyspark.sql.“

Data preparation with Enkidu

Also useful :

Pandas:

https://pandas.pydata.org/pandas-docs/stable/index.html

Use „Search the docs“ up on the left.

Needed solution usually follows: „pandas.DataFrame“

Look in the Parameters list for the parameters we need.

These instructions are repeated in the notebook!

Data preparation with Enkidu - hints

- **Enkidu**

16 Nodes in total (2 Login, 2 IO, 3 Service, **9 Compute Nodes à 512GB**)

⇒36*9 = 324 cores / 60 users = 5.4

⇒ **max.** 5 cores / user (+ 24 free cores)

- We start **safe** with

> **%spark 5 40g Sbahn_manipulation**

Data preparation with Enkidu - hints

-> Start sharing notebook.

- **To speed up the exercise:**
- Only 10% of the train DataFrames are kept after reading in the data
- Weather DataFrames: No need to reduce since these data are filtered out according to the train ones

Data preparation

Data manipulation procedure (interactively in the notebook):

1. Weather data:

- csv file, read-in and manipulated with Pandas, e.g.:
  drop columns, delete or replace rows with damaged entries
- Generate both a Pandas and a Spark DF
  (df_pd_weather and df_weather)

Data preparation

2. Train data:

- „ist“ (real) and „soll“ (scheduled) data are joined into **one** Spark DF (df_all)
- Time: Use unix time https://www.unixtimestamp.com to:
  - Easily perform operations on the time entries, such as compute the **delay** based on „ist“ and „soll“ times
  - Obtain **useful features**: month, weekday, journey duration, etc.

Data preparation

3. Merge train and weather data:

- Create a new merged DataFrame (df_proper)
- Filter out „outlier“ delays
- Add delay at stations -1 and -2 (df_ts)
- df_ts_classification includes a 0-1 column (delay yes/no, with threshold set to 0 minutes)

Data preparation

4. Split the DataFrames into „training“ and „test“ (for ML later on):

- Split df_ts and df_ts_classification for the ML pipeline:

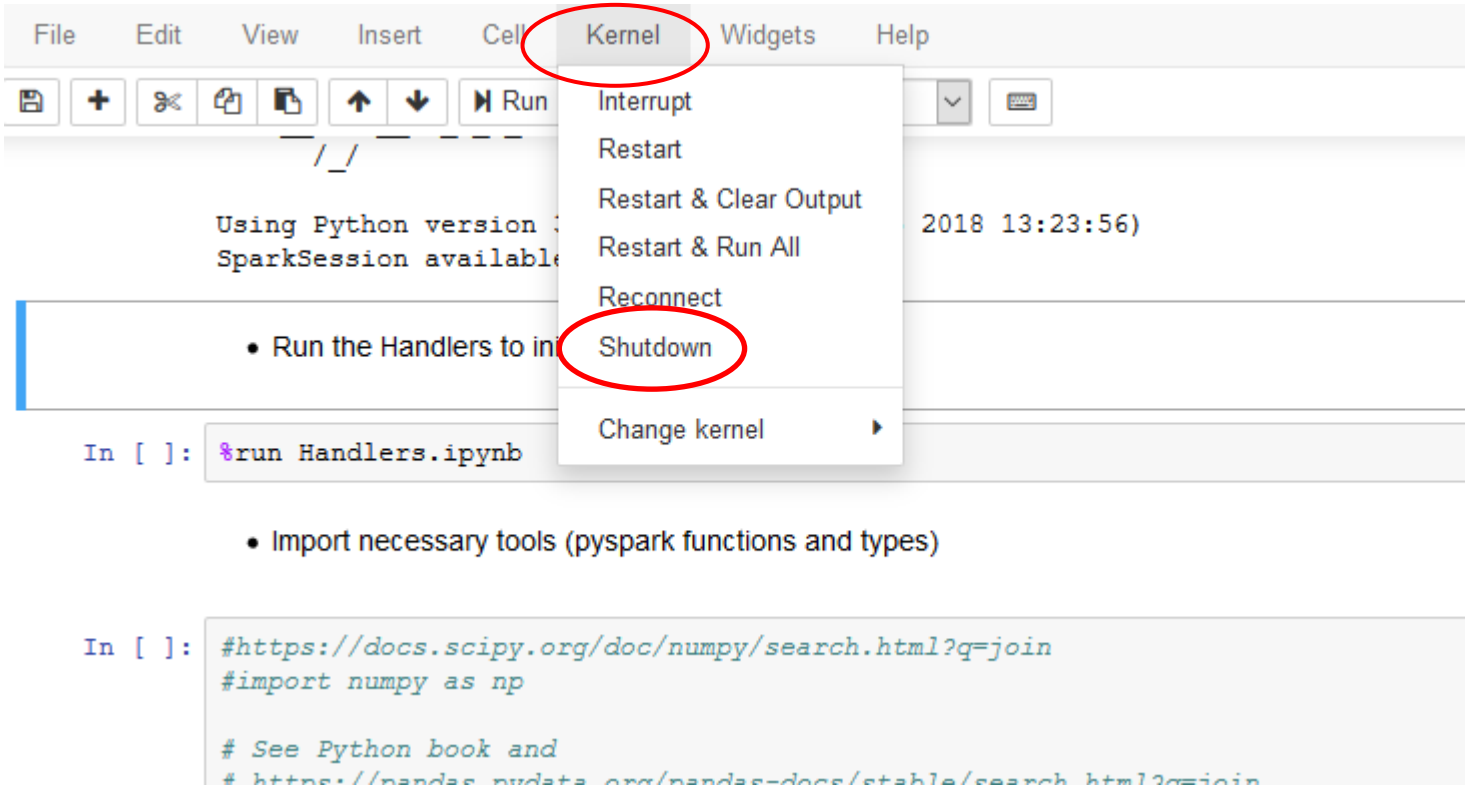
70% training and 30% test (using **random split**):

df_ts_training, df_ts_test

- Write down all DataFrames with HDFS -> **not executed**

Data preparation

When all is done, **shut down the kernel**:



Data preparation – feature columns (1)

|-- MONAT: **month**
|-- TAG: **day**
|-- STUNDE_SOLL: **sched. departure hour**
|-- MINUTE_RANGE: **0 or 30**
|-- ZUGEREIGNIS_LINIE: **S-Bahn line number**
|-- SERVICE_START_ZEIT: **unix time for scheduled service start**
|-- ZUGEREIGNIS_ISTZEIT: **unix time for actual departure**
|-- ZUGEREIGNIS_TYP: **40=departure**
|-- ZUGEREIGNIS_SOLLZEIT: **unix time for scheduled departure**
|-- SERVICE_ID: **service unique ID**
|-- ZUGEREIGNIS_ZUGNUMMER: **train number**
|-- ZUGEREIGNIS_DS100: **station code**

Data preparation – feature columns (2)

|-- MINUTE_SOLL: **scheduled minute**

|-- STUNDE_SER: **scheduled service start hour**

|-- MINUTE_SER: **scheduled service start minute**

|-- WOCHENTAG_SER: **scheduled service start weekday**

|-- WOCHENTAG_SOLL: **scheduled weekday**

|-- VERSPAETUNG: **delay (minutes)**

|-- LAUFSZEIT**: diff. sched. departure - sched. service start (min.)**

|-- Datum: **date**

|-- prev0_VERSPAETUNG: **delay (minutes) at the station -1**

|-- prev1_prev0_VERSPAETUNG: **delay (minutes) at the station -2**

Data preparation – feature columns (3)

|-- Mittelwerte_Temperatur: **temperature**

|-- Mittelwerte_Rel_Feuchte: **humidity**

|-- Mittel_WG: **avg. wind speed**

|-- Max_WG: **max wind speed**

|-- Summe_Niederschalg: **total precipitation**

|-- Mittel_NO: **avg. pollution data…**

|-- Mittel_NO2: **…**

|-- Mittel_O3: **…**

|-- Mittel_PM10:

|-- TimeUnix: **unix time of the weather forecast**

Data Visualization

Main **tools** for **visualization** of manipulated data:

- **Matplotlib** (plt):

A comprehensive library for creating visualizations in Python:

https://matplotlib.org/

- **Seaborn** (sns):

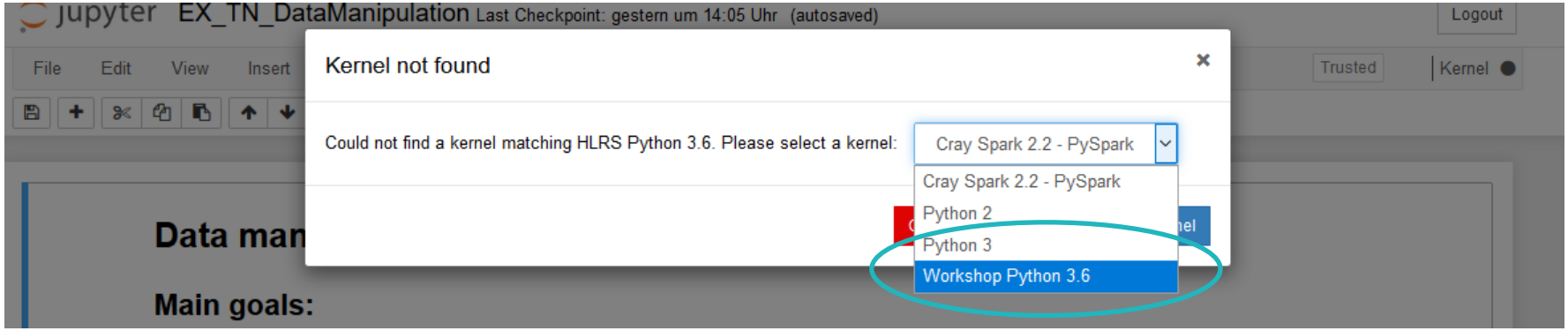
A Python data visualization library based on matplotlib. It provides a high-level interface for drawing **statistical** graphics:

https://seaborn.pydata.org/

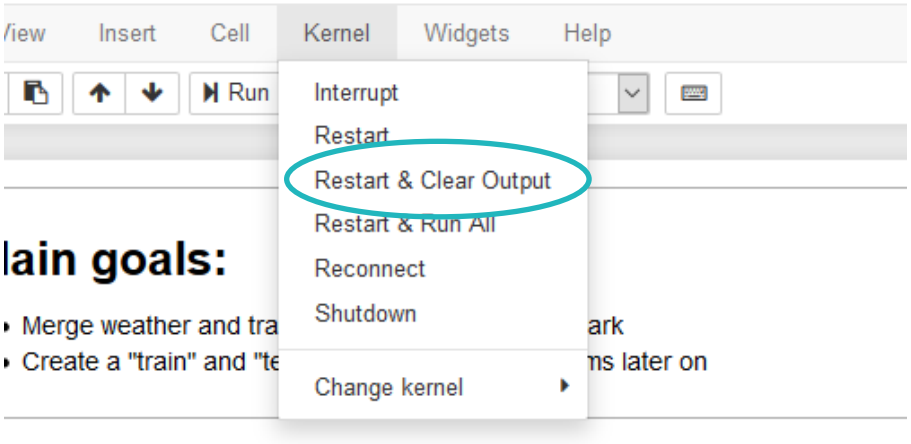
Disclaimer: *We will not go into the details of each plotting algorithm/tool!*

Data visualization with Enkidu **EX_TN_DataVisualization**

Choose **Workshop Python 3.6** (supports „magics“ and more…)



Choose: **Kernel -> „Restart and clear output“** to start with a clean workspace.



Data Visualization

Important before starting:

Replace in the Notebook EX_TN_DataVisualization:

username_local = "_____"

with your local username vdl1XXX (needed for I/O paths).

(run „id“ in an Enkidu cmd window)

____ = exercise, replace! (you will get an error otherwise)

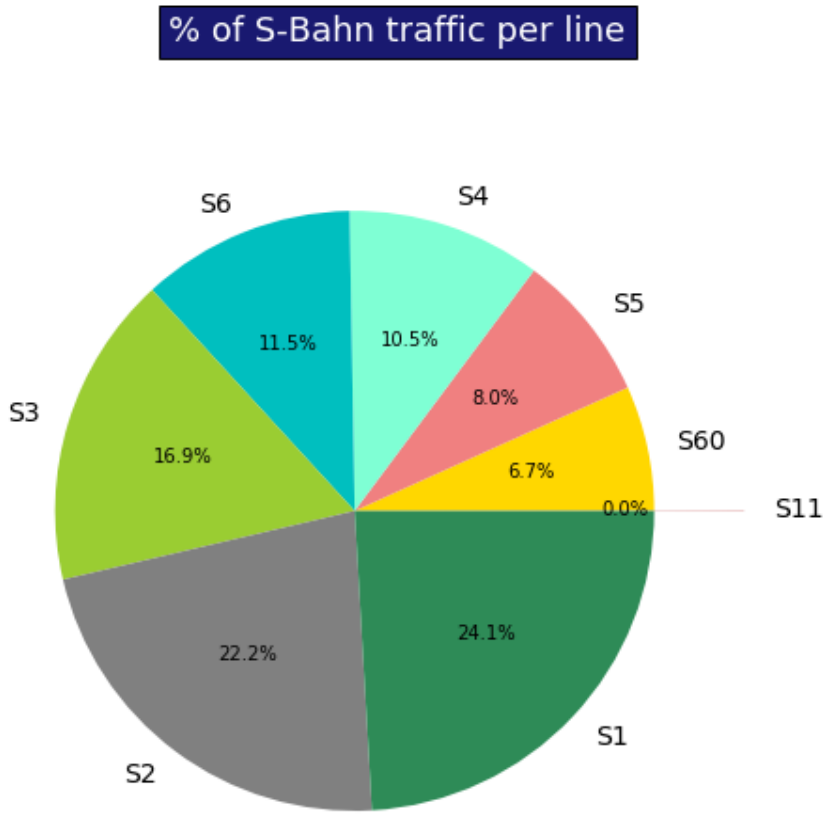
Solution: EX_TN_**SOL**_DataVisualization.ipynb

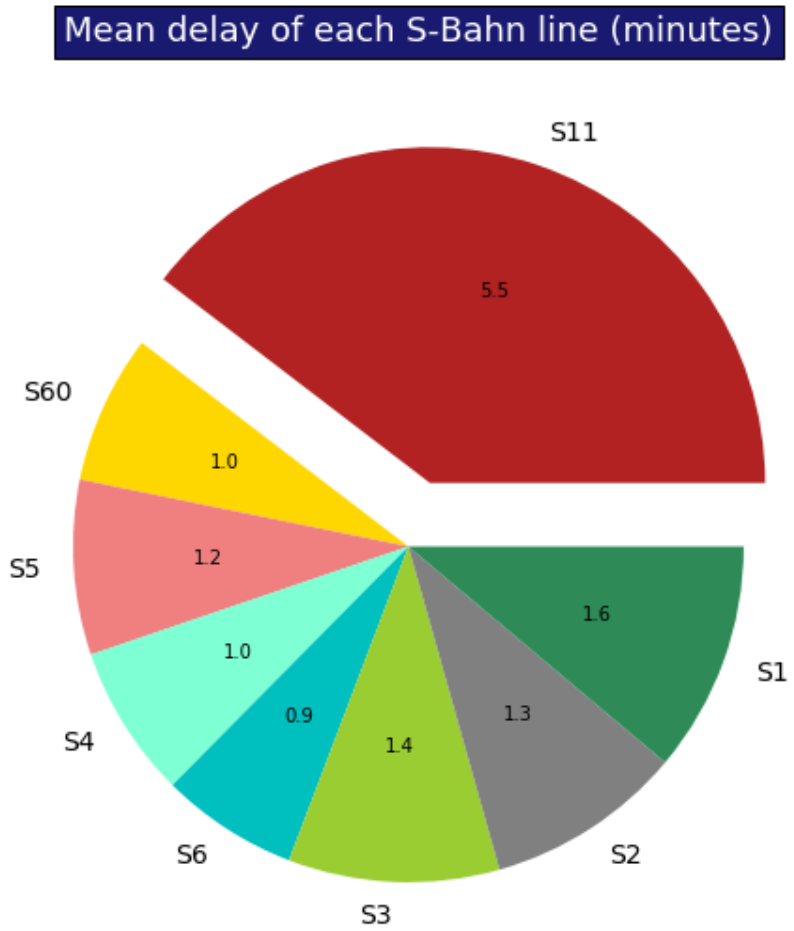
Data Visualization

1. Compute first statistical data and obtain a **pieplot**

- Read-in the complete Spark DataFrame **df_ts**, with manipulated:
  - Train/Time features
  - Weather features
  - Delay 0, -1, -2
- Obtain Pandas DataFrame **df_ts_pd**
- Obtain Pandas DataFrame **df_stats** based on df_ts_pd with:
  - statistical information (count, max, mean, min)
  - … grouped by S-Bahn line number
- Plot the **count** and **mean** information into two pie plots

Data Visualization





Data Visualization

2. Barplot

- **Goal**: Highlight frequency of three types of delays (in minutes):

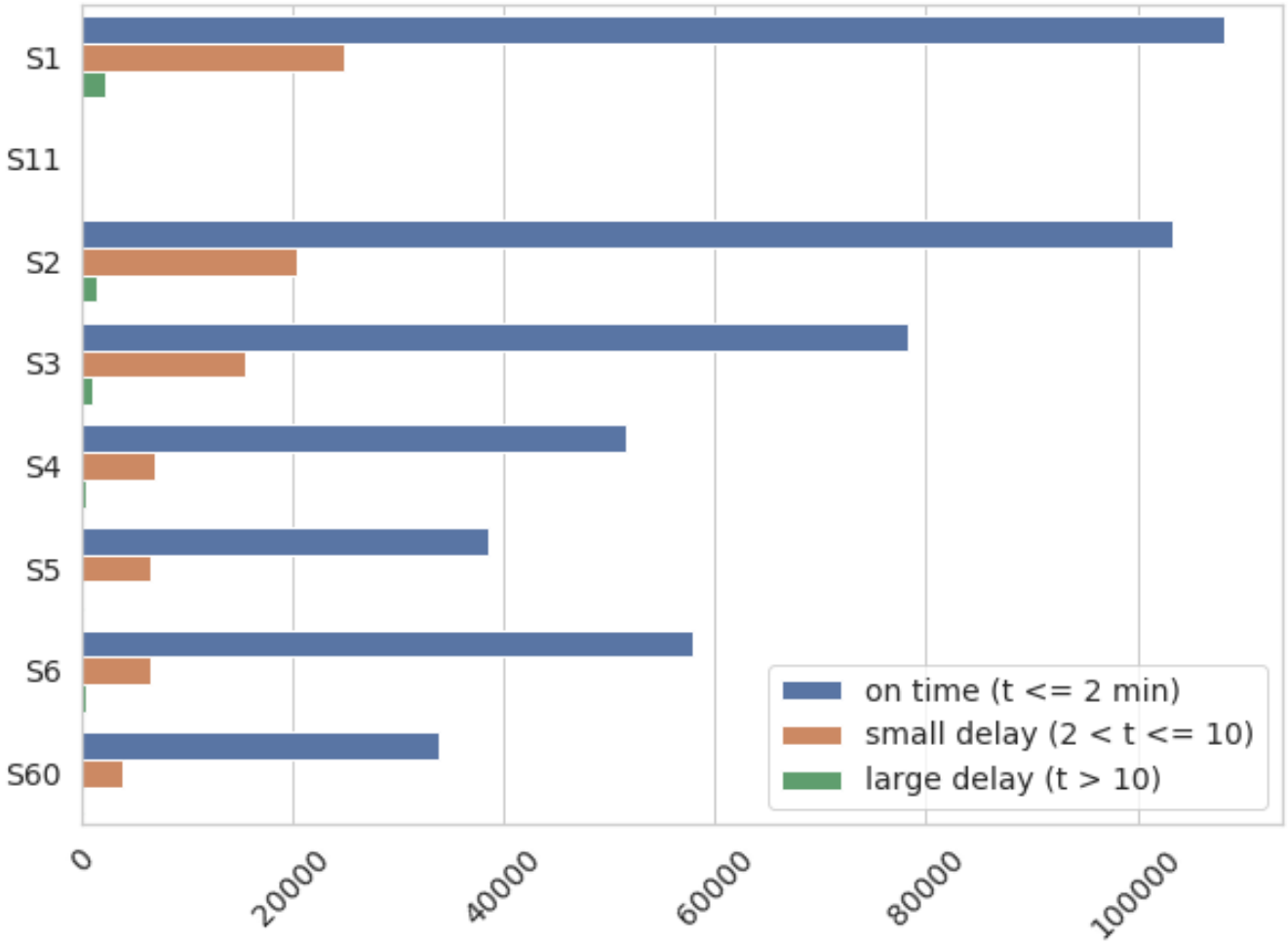
  delay<2; **2<=delay<=10**; **delay>10**

- Define a **Lambda function** to assign a numbered code to each category of delay:

  delay -> **0**; **1** ; **2**

- **Apply** the Lambda function to the **delay column** of the Pandas DataFrame
- Plot the barplot using **Seaborn**

Data Visualization



Data Visualization

3. Correlation map

- Objective:
  - Find out **linear relations** among the features and with the **delay**…
  - …to identify a **subset of features** on which the ML algorithms will (potentially) successfully run
- **Nonlinear** relations are **not** detected!
- Additional tools:
  - Scikit-learn: open source machine learning library that also provides various tools for data pre-processing:

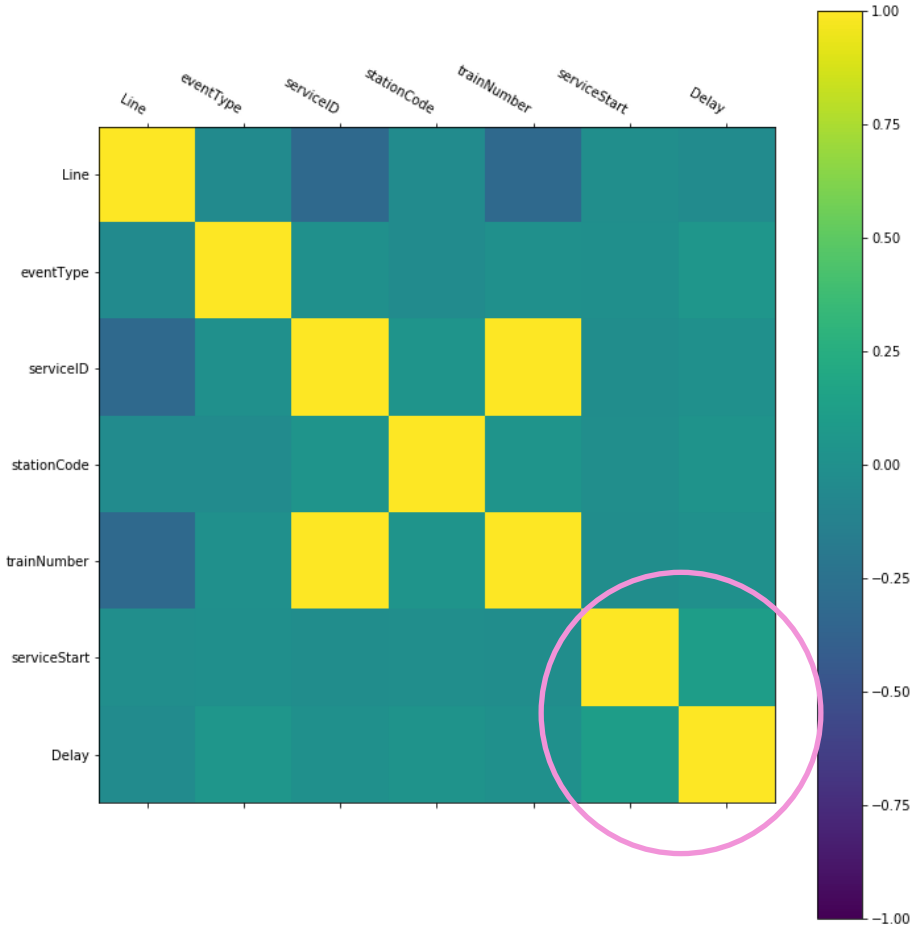
    https://scikit-learn.org
  - … in particular, the LabelEncoder for vectorization

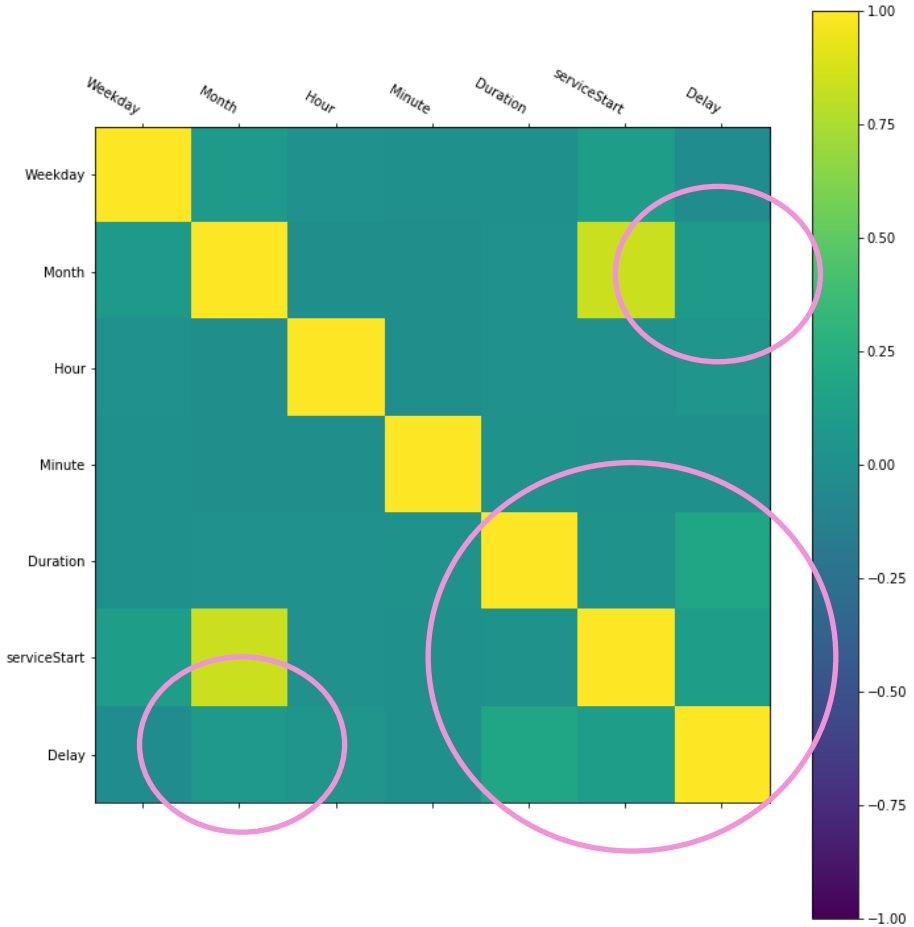
Return here to NB (Ex. to get ONE correlation plot).

Data Visualization

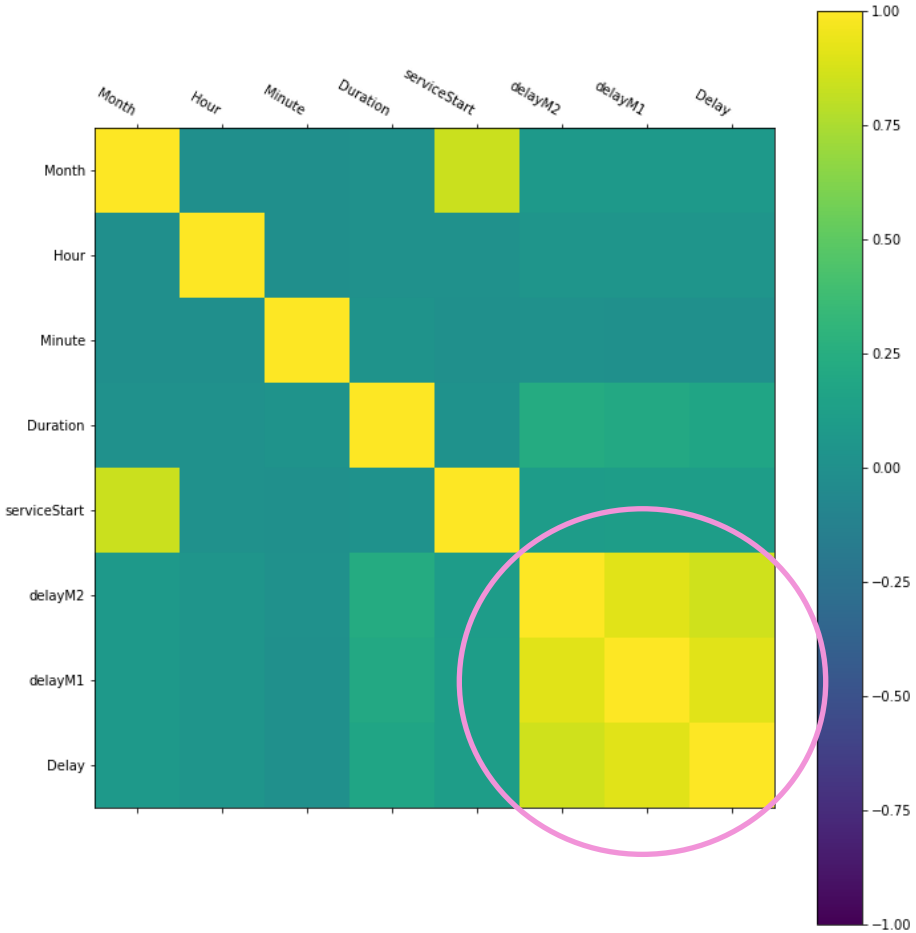
- We consider **4 sets of feature columns** to observe possible correlation to the delay:
  - Original train schedule features of the original dataset
  - Additional features from time manipulation
  - Include delay at the previous stations (-1, -2)
  - Include weather features
- Accordingly, **4 Pandas DataFrames** (as subsets of df_ts_pd) are created.
- The **Pearson correlation coefficient** is computed with Pandas and then plotted with matplotlib.

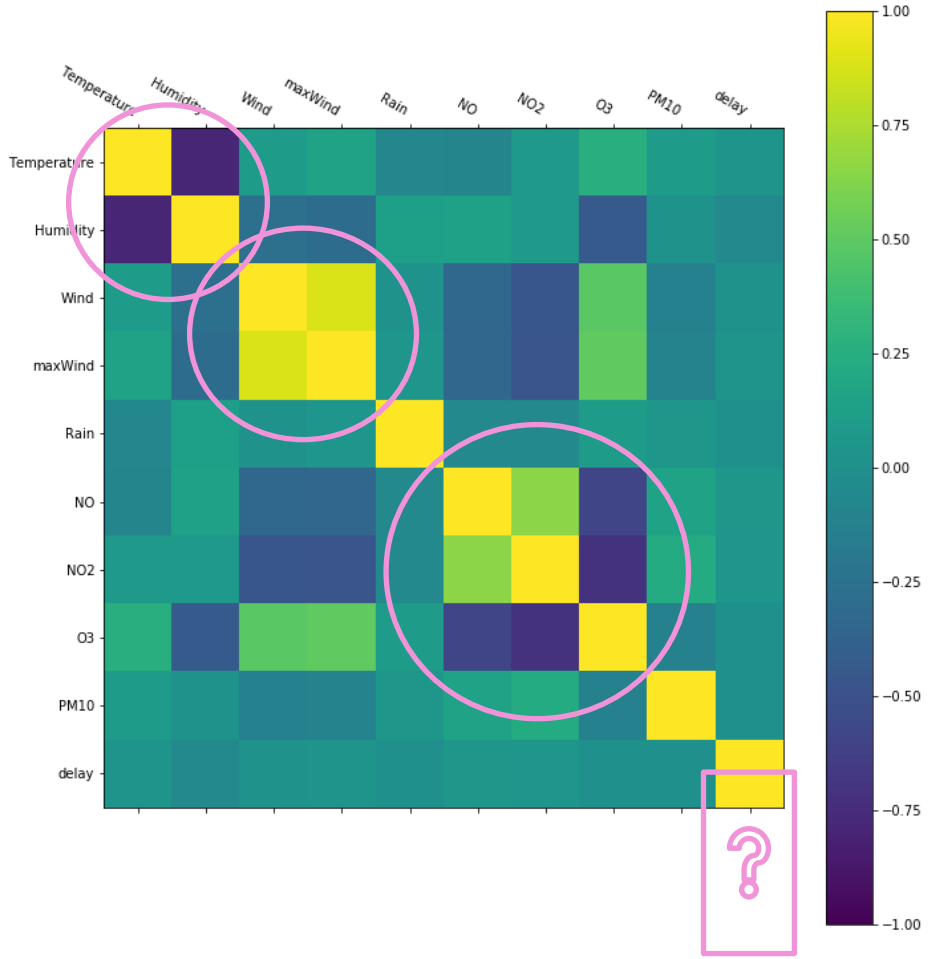
Data Visualization





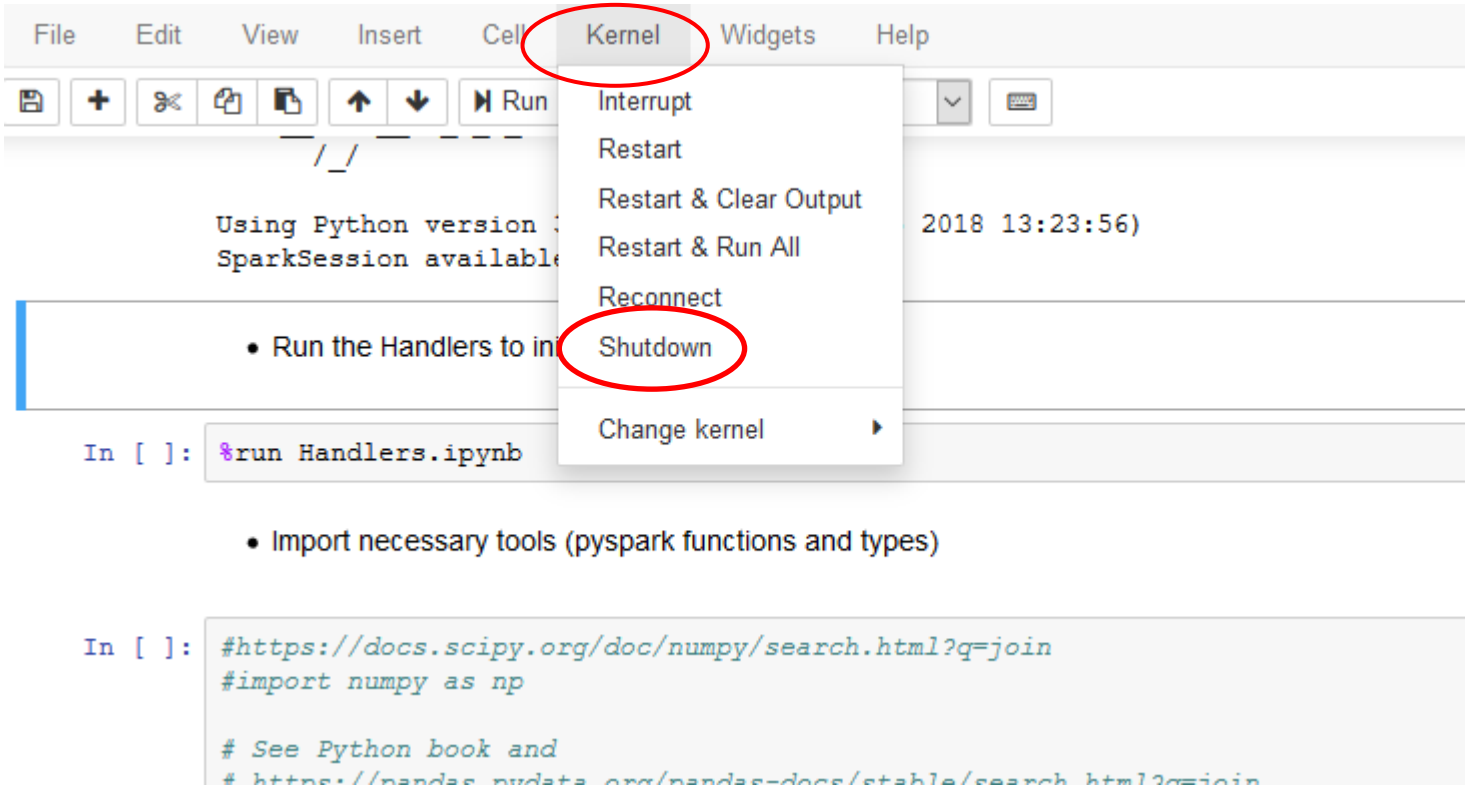
Data Visualization





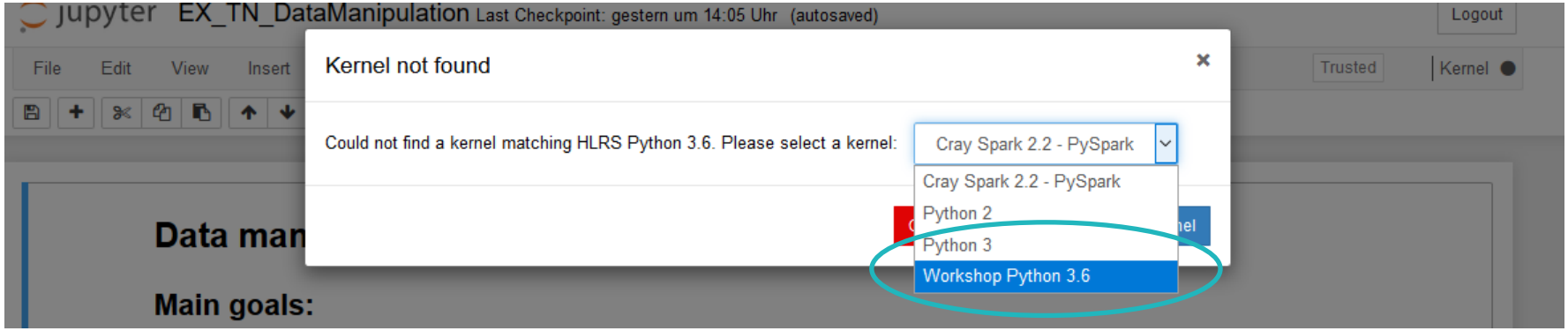
Data Visualization

When all is done, **shut down the kernel**:

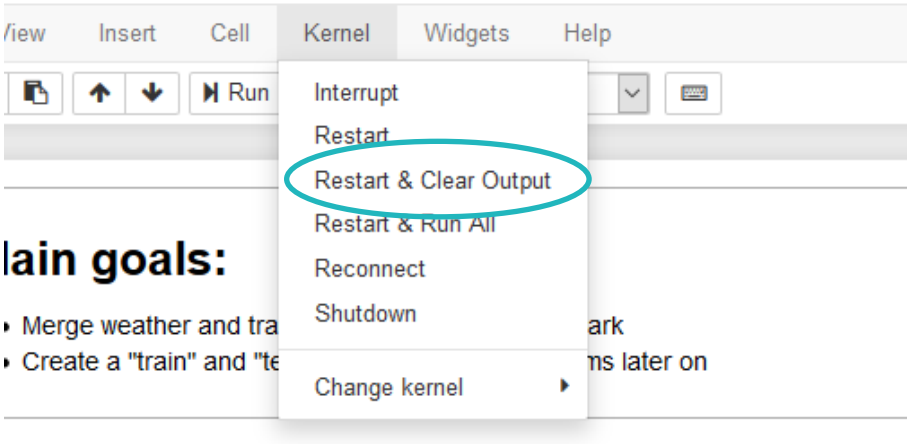


Machine Learning with Enkidu **EX_TN_MachineLearning**

Choose **Workshop Python 3.6** (supports „magics“ and more…)



Choose: **Kernel -> „Restart and clear output“** to start with a clean workspace.



Machine Learning

Important before starting:

Replace in the Notebook EX_TN_MachineLearning:

username_local = "_____"

with your local username vdl1XXX (needed for I/O paths):

(run „id“ in an Enkidu cmd window)

____ = exercise, replace! (you will get an error otherwise)

Solution: EX_TN_**SOL**_MachineLearning.ipynb

Machine Learning

1. Initialize Apache Spark

%spark 5 40g Sbahn_machineLearning

2. Read in data from manipulation

Read in the DataFrames after DataManipulation as HDFS data:

- **Training**: *df_train* (70% of all data)
- **Test**: *df_test* (remaining 30%)
- Additionally, **classification** (delays y/n) at stations 0, -1, -2: *df_train_classification* and *df_test_classification*

Machine Learning

3. Run the **regression** algorithm

- Define the **feature columns** to run the model in several combinations
  - (details later)
  - Notice that the feature columns are listed twice:
    - « cols_to_inx = feature columns sent to the ML pipeline
    - « cols_inx = output of **StringIndexer** in the ML pipeline (next slide)
- Execute the **ML pipeline** (next slides)

-> Open **Handlers.ipynb** (in the notebook) and go to the function **linReg**

Machine Learning

3. Run the **regression** algorithm

The **ML pipeline** is defined in the Handlers (*linReg*):

Arguments: DataFrame for training, lists feature columns

Feature Engineering steps:

https://spark.apache.org/docs/latest/ml-features

- **StringIndexer**:

Encodes a **string column** of labels to a column of (**numerical**) label indices (cf. one-hot encoding seen in NVIDIA).

- **VectorAssembler**:

Combines the given list of columns (feature matrix) into **a single vector column** in order to train ML models

Machine Learning

3. Run the **regression** algorithm

- **Normalizer**:
  - Normalizes each vector to have unit norm
  - It takes a parameter p∈ $[1, \infty]$: which specifies the **L^p-norm** used for normalization (p=2 by default, changed to p=1):

$$v_{norm} = \frac{v}{\|v\|_p}$$

$$\|v\|_p = \left(\sum |v_i|^p \right)^{1/p}$$

⇒ Standardize the input data and improve the behaviour of the learning algorithm

Q: What if p = infinity?...

Machine Learning

3. Run the **regression** algorithm

ML pipeline in the Handlers (*linReg*):

https://spark.apache.org/docs/latest/ml-classification-regression

- Define the **model class**: **LinearRegression** with the respective parameters (e.g. maxIter, regParam) or other algorithms (GeneralisedLinearRegression, …)

https://spark.apache.org/docs/1.5.2/mllib-linear-methods.html

=> All the steps above define the **estimator** (i.e., the architecture) pipeline

- Define a suitable metric for the **evaluator**: Root Mean Squared Error (RMSE, alternatively: MSE, MAE, …)

https://spark.apache.org/docs/latest/mllib-evaluation-metrics.html#regression-model-evaluation

Machine Learning

3. Run the **regression** algorithm

ML pipeline in the Handlers (*linReg*):

Validation steps:

- Define the **parameter grid** to (overwrite already defined parameters and) fit the model at each grid point:
  - Here: 2x2 regularization parameters

https://spark.apache.org/docs/1.5.2/ml-linear-methods.html

(for an overview of the parameters)

- Define the **cross validation**, to cross-validate:
  - Here: 10 folds over the training dataset

https://spark.apache.org/docs/latest/ml-tuning.html#cross-validation

=> The **(2x2)x10=40 models** are being trained

Machine Learning

3. Run the **regression** algorithm

Already done: Call the **ML pipeline** defined in the Handlers (*linReg*) to

- **fit** the model to the training data and
- … return the obtained **model** (i.e., the weights) and the **evaluator** (i.e., RMSE)

-> Go back to MachineLearning.ipynb (share NB)

Now:

- Define the evaluator outside the pipeline (**exercise**)
- Load the pretrained ML model (**exercise**)

… then call the **ML pipeline** defined in the Handlers (*linReg**Test***) to

- **predict the delays** on the test dataset and
- **… evaluate** the quality of the inference by computing the RMSE

PRACTICAL = EX 1-2-3 + plot

Then slides.

Machine Learning

4. Evaluate the **regression** algorithm (LR)

Optimize the algorithm:

- Which features give the best/worse RMSE?
- **Significant correlation** of the features to the label (or target: delay)

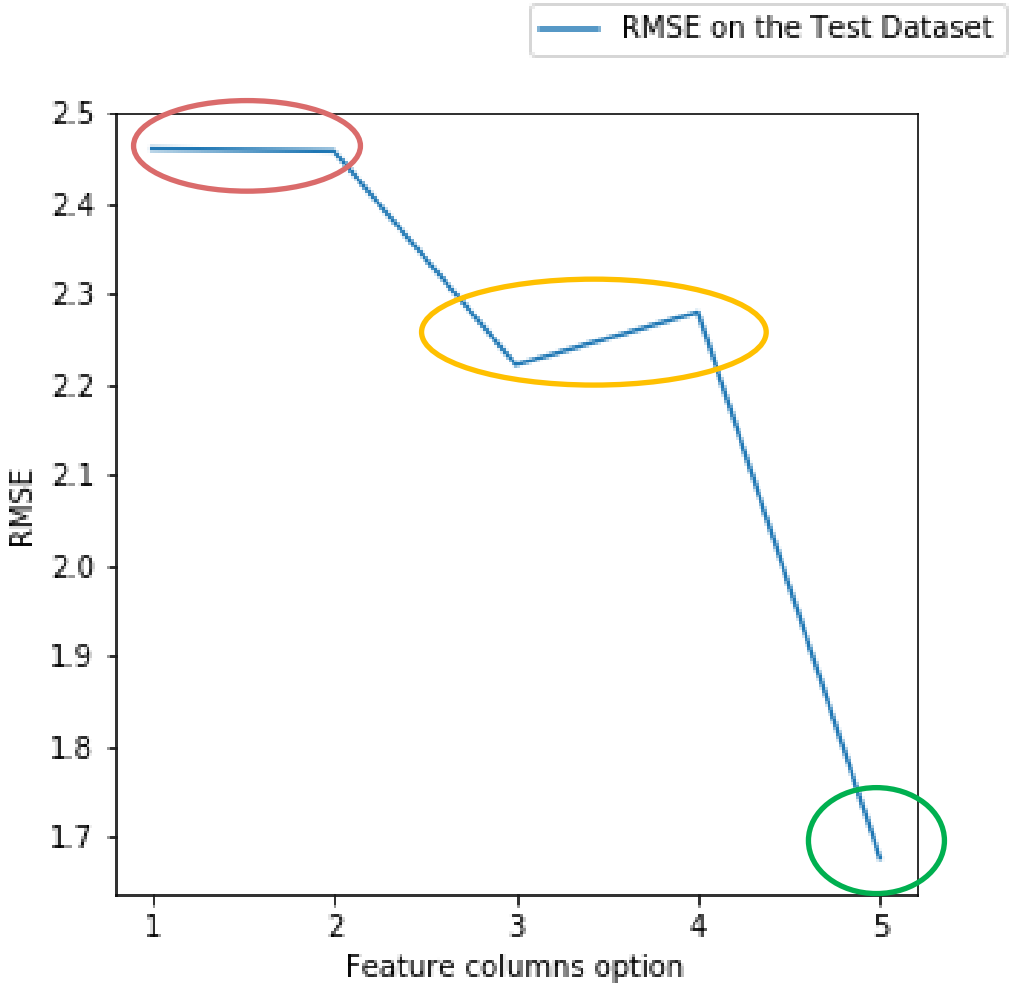
$\Rightarrow$ these columns are relevant **features** for the model (in theory).

We selected sets of **feature columns** to run the model in several combinations:

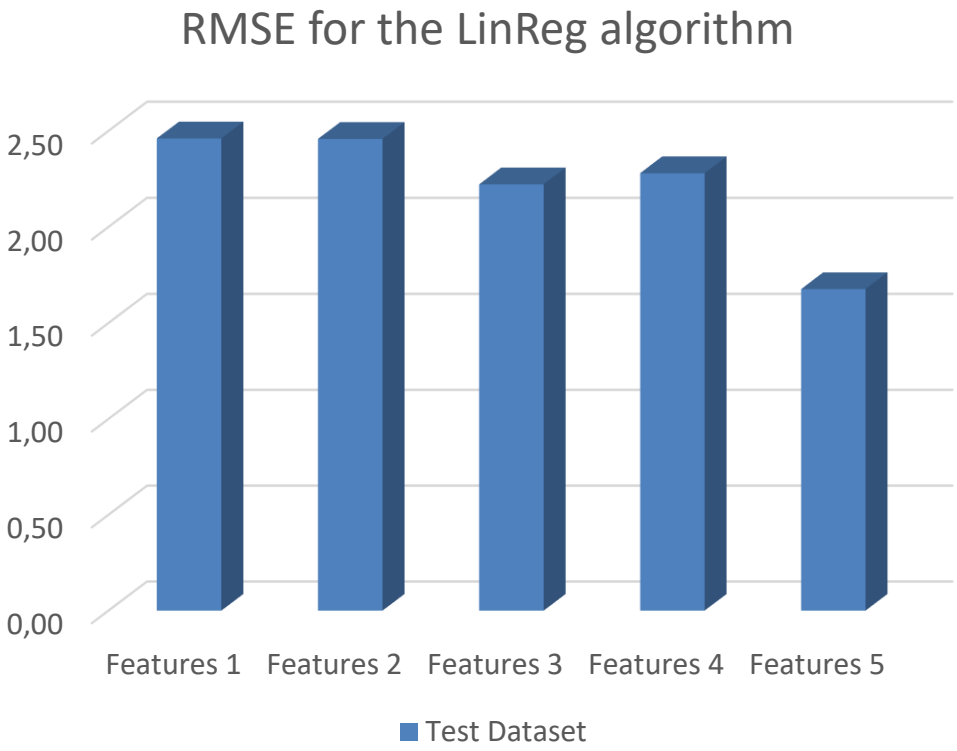
1) Basic information of the original train dataset (5)
2) Basic information, weather data (11)
3) Delay at station -1 (additional features) (1)
4) Delay at stations -1,-2 (additional features) (2)
5) Basic information, delays and duration (additional features), weather data (8)

Machine Learning

1) Basic information of the original train dataset (5)
2) Basic information, weather data (11)
3) Delay at station -1 (additional features) (1)
4) Delay at stations -1,-2 (add. features) (2)
5) Basic information, delays and duration (add. features), weather data (8)

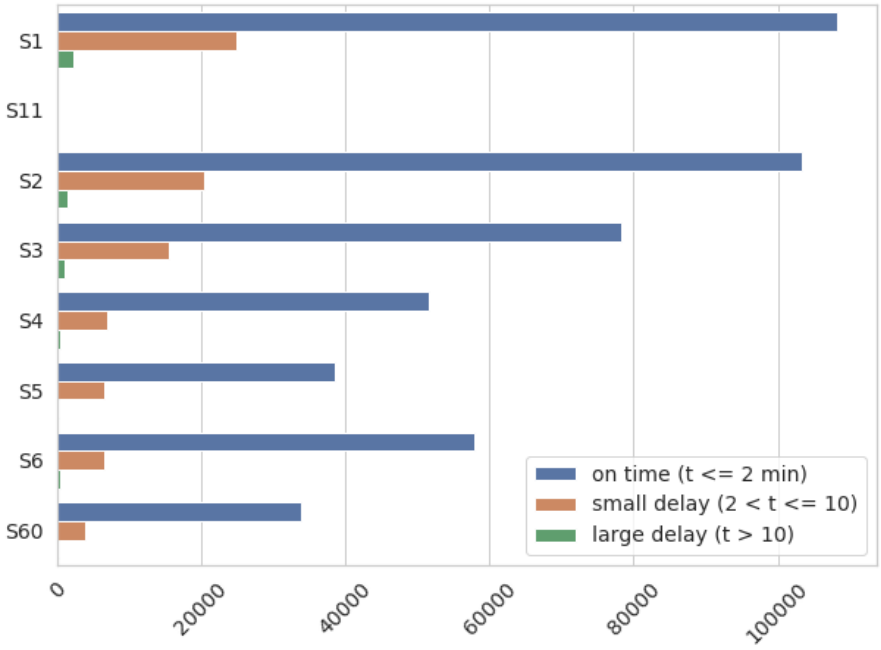


Machine Learning



- Most delays **below 2 minutes**
- Best average error of 1.5 minutes

=> ☹

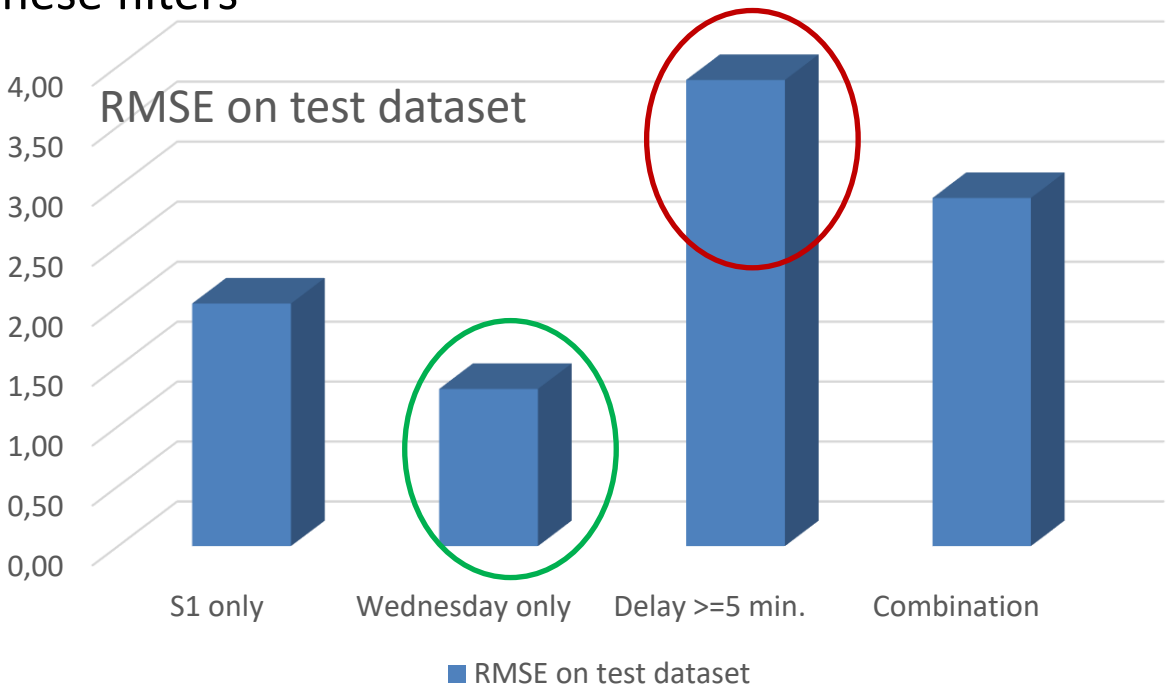


Machine Learning

Back to the Notebook

Strategies to optimize the outcome:

- Filter both the training and test datasets (lines, weekdays, delay), and
- Combine these filters



Machine Learning

Other strategies to optimize the algorithm:

- Observe the learning curve by changing the **ratio training/test dataset**
- Different **LinearRegression** algorithms (see options in Handlers)
- Different **evaluators** (MSE, MAE, …)
- **Hyperparameters**: Tuning with higher / lower values could lead to better results but also to **overfitting**! No guidelines. (see options Handlers)

Machine Learning

5. Run the **classification** algorithm

- We predicted the delay
  - in minutes
  - on the dataset as a whole
- We aim now to predict
  - **delay at all** (delay yes/no)
- **Test** will be done on a subset (small exercise):
  - the data of the line **S1** (first split block)…
  - clustered by **station** (second split block)

⇒ **Goal: Answer to:**

Do we have a delay at a particular station of the line S1?

PRACTICAL EX 4 in the Notebook. *Share.*

Machine Learning

5. Run the **classification** algorithm

- **Feature columns**: same as LinearRegression
- Execute the **ML pipeline:** similar to LinearRegression (next slides)

-> Open **Handlers.ipynb** (in the notebook) and go to the function **clfcTrain**

Machine Learning

5. Run the **classification** algorithm

Call the **ML pipeline** in the Handlers (*clfcTrain*):

Feature Engineering steps:

https://spark.apache.org/docs/latest/ml-features

- **StringIndexer**
- **VectorAssembler**
- **Normalizer**

} same as LinearRegression

Machine Learning

5. Run the **classification** algorithm

Call the **ML pipeline** in the Handlers (*clfcTrain*):

https://spark.apache.org/docs/latest/ml-classification-regression.html#random-forest-classifier

https://spark.apache.org/docs/latest/mllib-ensembles.html#random-forests

- Define the **algorithm**: **Random Forest Classifier** with the respective parameters (numTrees: see next slides)

=> All the steps above define the **estimator** pipeline (i.e., the architecture)

- Define the **evaluator**: Accuracy (see next slides)

Machine Learning

5. Run the **classification** algorithm

- **Random Forest** (pp. 421-432): Combines the information of overfitting randomized decision trees (**ensemble method**), then averages the results.
- Each decision tree fits a **randomized** subset of data chosen automatically by the algorithm.

In the RandomForestClassifier, one can choose:

- **numTrees**: Number of trees increase =>

  Increase of compute time (linear), accuracy at test-time

  – Here: 100
- (maxDepth: Max depth of each tree can be tuned to produce overfitting)

Machine Learning

5. Run the **classification** algorithm

- **MulticlassClassificationEvaluator** is the evaluator:

https://spark.apache.org/docs/2.2.0/mllib-evaluation-metrics.html

  - generalizes Binary Evaluator (0/1 = prediction correct or not)…
  - … in case of multiple choices (e.g. digits 1-10)

- Evaluation Metric is the **accuracy** $^1/_N \sum_{i=1}^{N} \delta_0(y_i - \hat{y}_i\ )$ where:
  - y and $\hat{y}$ are one-dimensional vectors (binary classification) of resp. true output and predictions
  - δ_0 returns 1 if the prediction is correct, 0 otherwise
  - N is the **number of samples** for each station of the S1 line

Machine Learning

5. Run the **classification** algorithm

Validation steps:

- Define the **parameter grid**
- Define the **cross validation**, to cross-validate e.g. 10 folds over the training dataset: 10 models are being trained

Machine Learning

5. Run the **classification** algorithm

Already done: Call the **ML pipeline** in the Handlers (*clfcTrain*) to

- **fit** the model to the training data and
- … return the obtained **evaluator** (i.e., accuracy) and the **model** (i.e., the weights)

-> Go back to MachineLearning.ipynb

Now:

- Define the evaluator outside the pipeline (**exercise**)
- Load the pretrained ML model

… then call the **ML pipeline** defined in the Handlers (***clfcLoopTest***) to

- **predict the delay label** on the test dataset (one DataFrame for each station) and
- **… evaluate** the quality of the inference by computing the accuracy

-> See results in the notebook (and in the visualization part next)

PRACTICAL = EX 5 + display "df_accu_pd"

Then slides.

Machine Learning

6. **Performance** of the classification algorithm

- Training **5 different sets of features**, each one **running 3 times** (identical function calls).
- We **could** obtain different performances by changing the line:

(restart the kernel each time!)

%spark **108 450g** Sbahn

i.e. start a Spark session with values:

--total-executor-cores 108 --executor-memory 450g

(done in Urika-GX Gilgamesh: 41 compute nodes x 36 cores)

Machine Learning

- Using 3,2,1 full nodes respectively:
  - %spark **108** 450g Sbahn
  - %spark **72** 450g Sbahn
  - %spark **36** 450g Sbahn

- ((Optionally, in another terminal))

> module load tools/mesos

> minfo

displays occupied nodes and occupied resources per node.

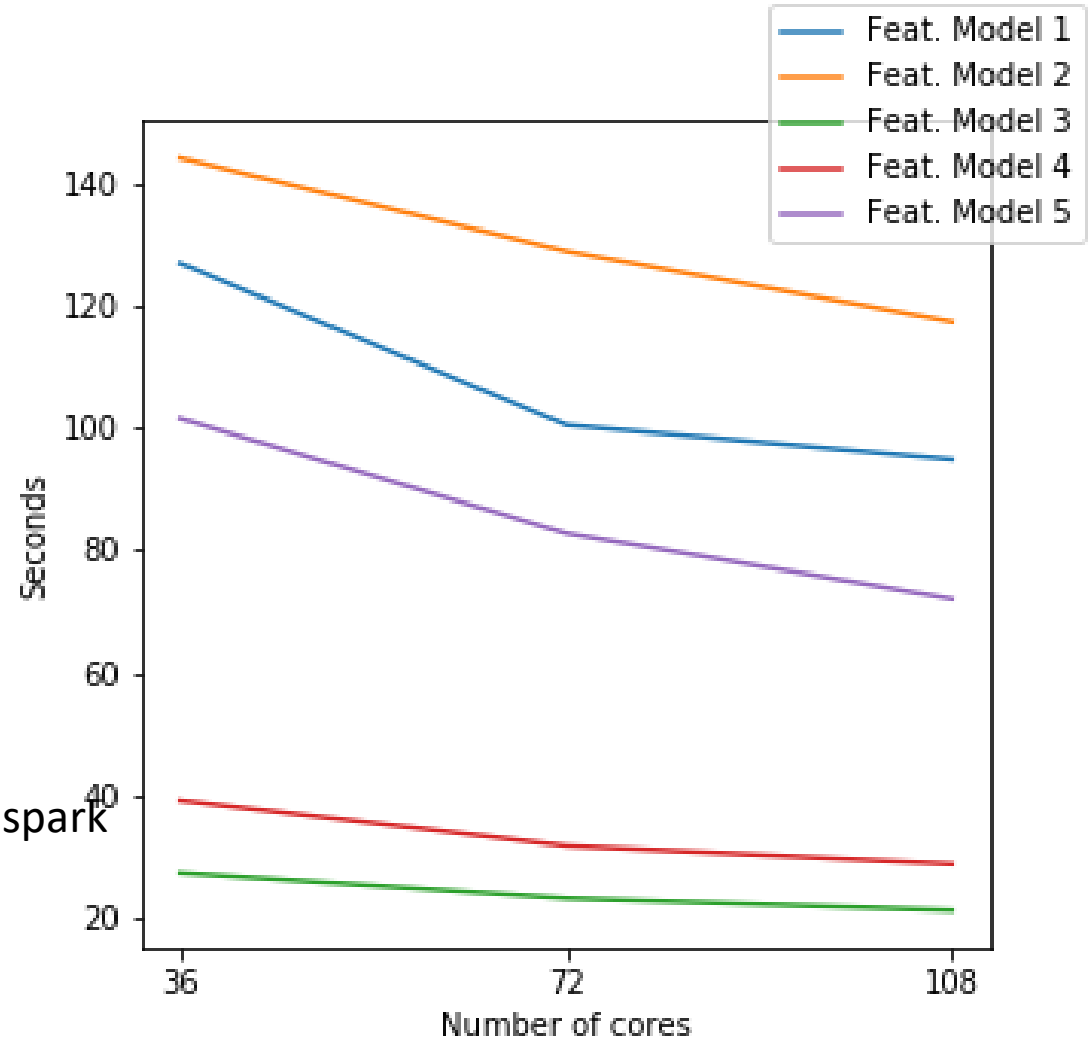
You can observe the current state of the cluster and what resources are occupied.

Machine Learning

Performance of the classification algorithm (strong scaling of fitting the **training** dataset):

– Average over 3 consecutive runs of the same function on Gilgamesch for:

- 5 feature combinations
- 1,2,3 nodes in parallel with spark



Machine Learning

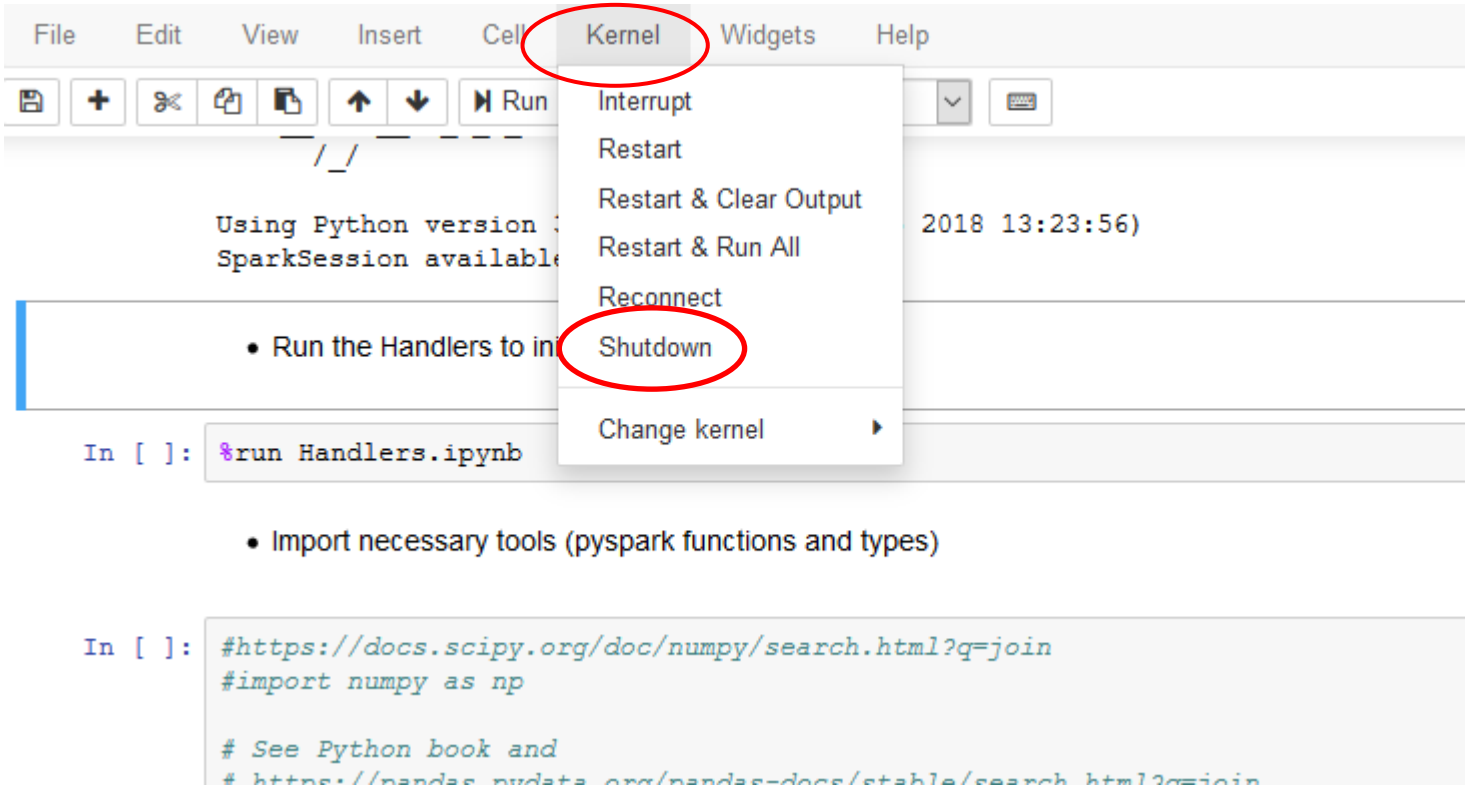
Disclaimer on this benchmark:

- Performance depends on the number of cores in a Spark session, but also…
- on the **memory / executor** used (here 450GB / 500 GB available)
- on the **communication overhead** between the nodes: max. efficiency by using all cores inside the node
- on the **balance** dimension of the dataset and number of threads

In the Notebook: observe the accuracy (we will visualize) and plot the precomputed performance results.

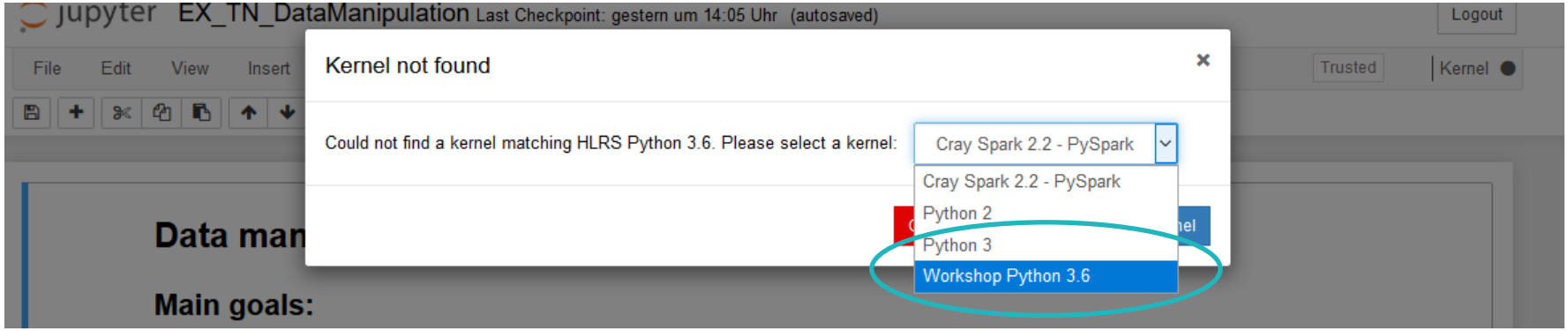
Machine Learning

When all is done, **shut down the kernel**:

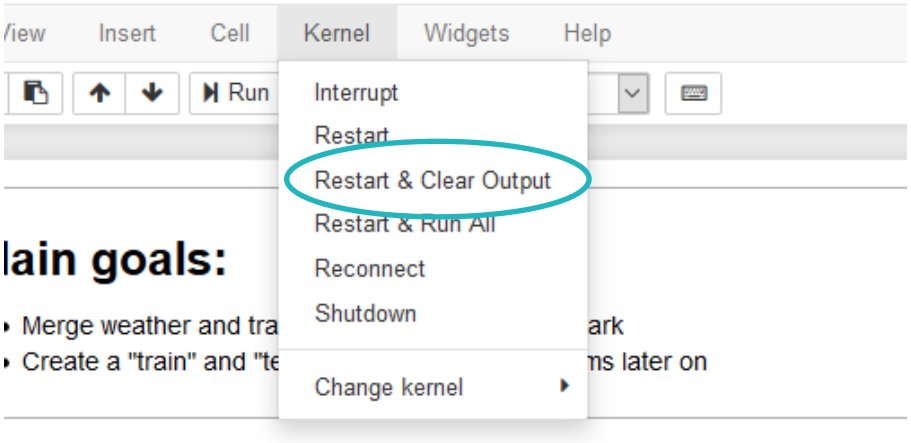


SKIPPED Visualization of Classification with Enkidu

Choose **Workshop Python 3.6** (supports „magics“ and more…)



Choose: **Kernel -> „Restart and clear output“** to start with a clean workspace.



SKIPPED Visualization of Classification Prediction

Important before starting:

Replace in the Notebook EX_TN_PredictionVisualization:

username_local = "_____"

with your local username vdl1XXX (needed for I/O paths).

(run „id“ in an Enkidu cmd window)

____ = exercise, replace! (you will get an error otherwise)

Solution: EX_TN_**SOL**_ PredictionVisualization.ipynb

SKIPPED Visualization of Classification Prediction

Goal: Visualize the classification results on a **map**.

- Plot packages to pip-install:
  - **geopy**: Python client to locate coordinates using geocoders
    https://geopy.readthedocs.io/en/stable/
  - **folium**: library to visualize data on a map
    https://python-visualization.github.io/folium/
- Read-in the results of ML-Classification of a Gilgamesch run on the complete dataset…
- that is, the **accuracy** of the predicted delay, averaged on all test samples **for each station**
- Convert this spark DataFrame into a pandas DataFrame

SKIPPED Visualization of Classification Prediction

- Generate two lists of stations:
  - All **ordered** stations on the S1 line
  - S1 stations according to the accuracy DataFrame (**not ordered**)
- Produce a python **dictionary** *DS100 : station name*. E.g. *TB : Backnang*
- Define a **threshold** for the accuracy colour-code on the map:

acceptable (>=0.8), borderline (0.5<= t < 0.8), trash (<0.5)

=> A colour is assigned to each station.

SKIPPED Visualization of Classification Prediction

1. Accuracy Plot

- One **accuracy** value for each station on the S1 line
- … averaged value based on predictions on the **test dataset**
- Predictions were calculated according to the **five combinations of features** => **five different plots**
- Accuracy displayed in colour according to the **threshold**
- The plot functions are defined in Handlers (class “PredictionVisualize”):
  - **getCoords**: Associate a colour to each station based on the accuracy,
  - … and find the location of every station with Geopy.
  - **drawMapDic**: Draw the map with folium.

SKIPPED Visualization of Classification Prediction

Feature columns recap:

1) Basic information of the original train dataset (5)
2) Basic information, weather data (11)
3) Delay at station -1 (additional features) (1)
4) Delay at stations -1,-2 (add. features) (2)
5) Basic information, delays and duration (add. features), weather data (8)

SKIPPED Visualization of Classification Prediction

To open the plots: **Browse** in the Notebook to:

/hpda-code/use-cases/sbahn/Plot/ClassificationPlot/Plotavg*

In case you cannot see the map, visualize the resulting map **locally on your laptop**:

Open an empty terminal, create a directory on the Desktop:

> cd ~/Desktop

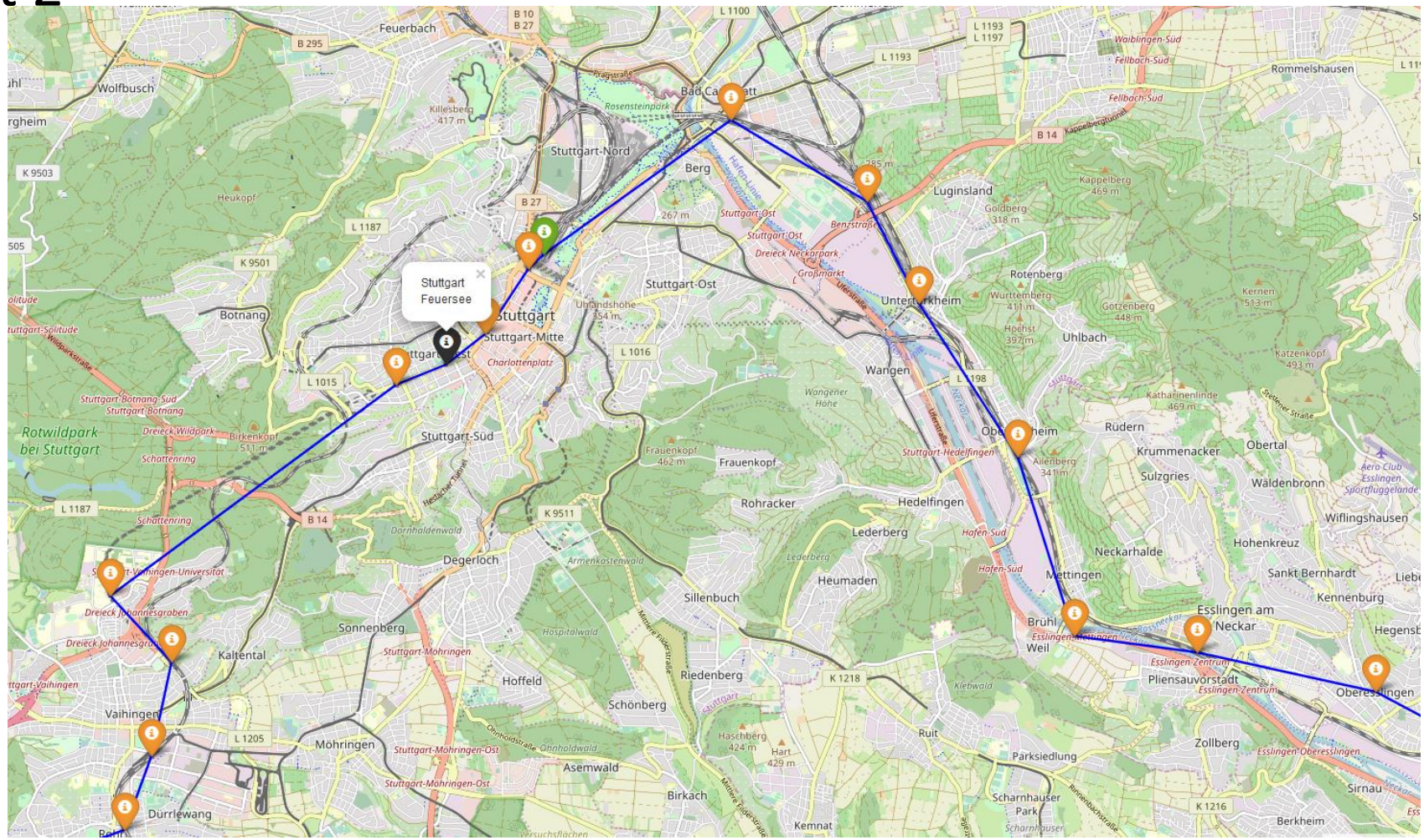
> mkdir classPlot

> cd ./classPlot

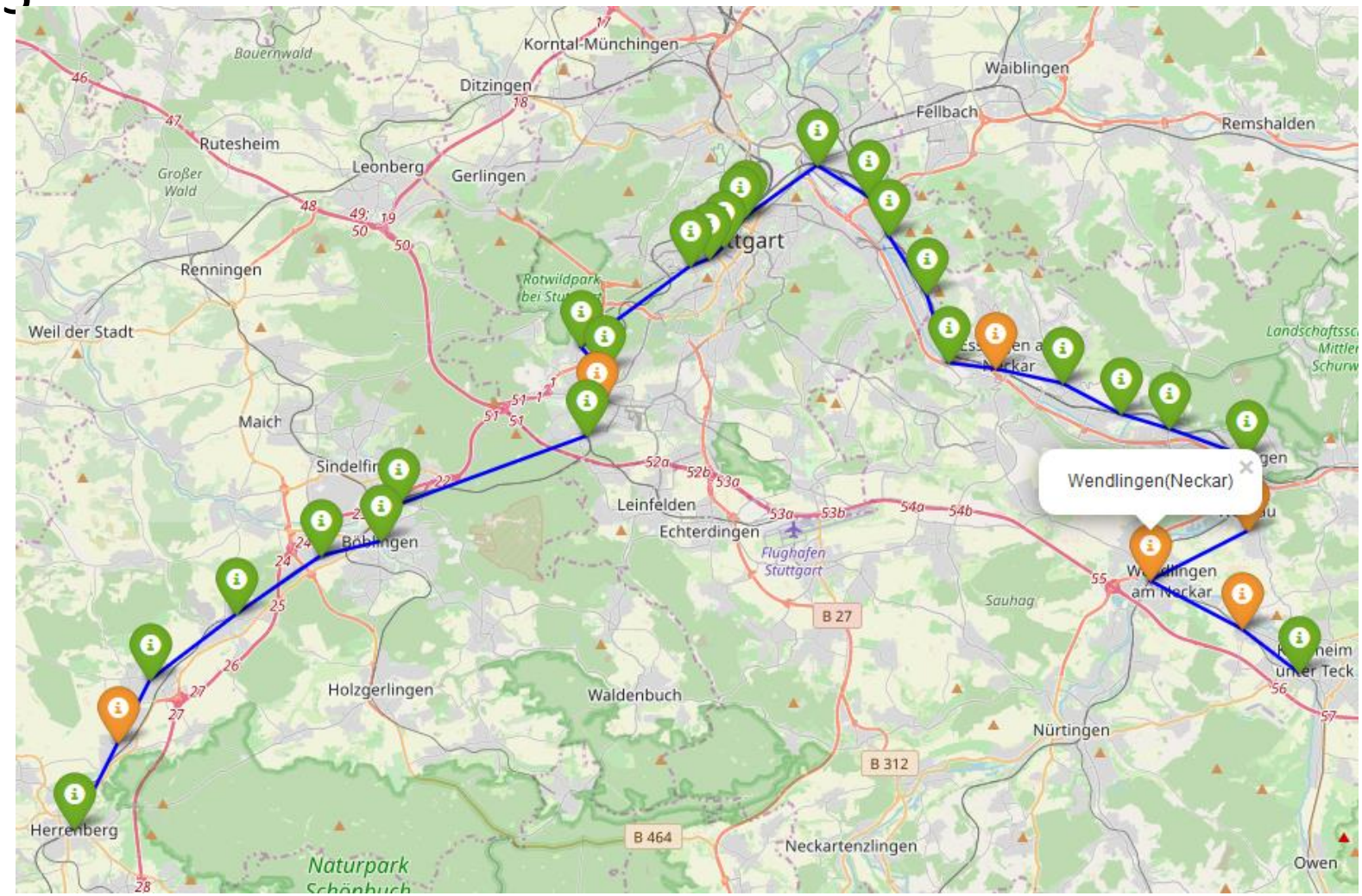
> scp **vdl1XXX**@enkidu-login1.hlrs.de:/home/users/**vdl1XXX**/hpda-code/use-cases/sbahn/Plot/ClassificationPlot/Plotavg* ./

Then, open each plot with a browser, e.g. Chrome.

SKIPPED Visualization of Classification Prediction: Feature set 2

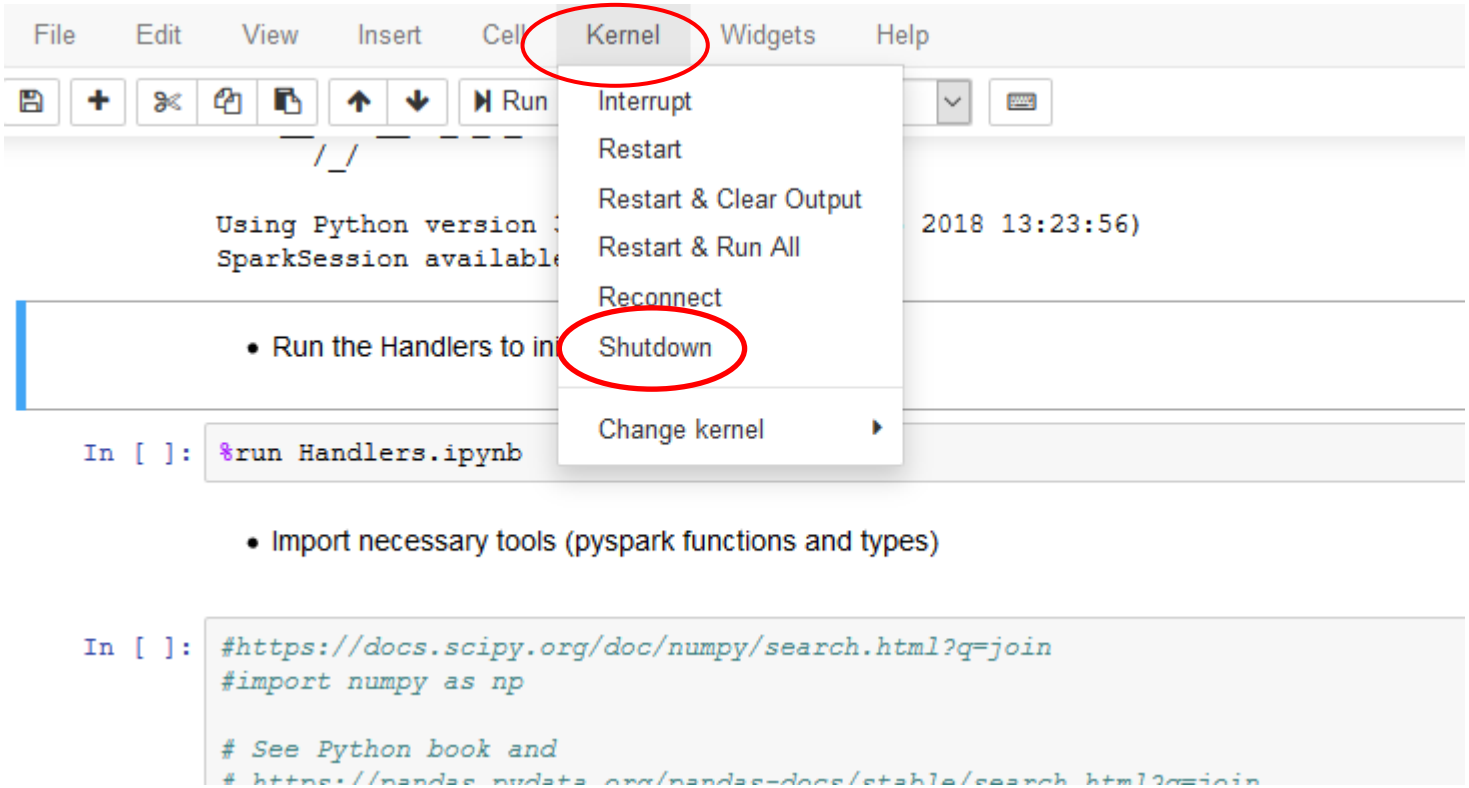


SKIPPED Visualization of Classification Prediction: Feature set 5



Visualization of Classification Prediction

When all is done, **shut down the kernel**:



Part III: Code as a Python SCRIPT in Urika and Vulcan

Code as a Python SCRIPT

Goal: connect Data Analysis and HPC

- Urika-GX is an HLRS “special-purpose” system for Data Analysis
- **Production** would not happen on a Jupyter Notebook

- … but as a **batch job** on an HWW system at HLRS (e.g., Hawk and Vulcan)
- HWW is the “Höchstleistungsrechner für Wissenschaft und Wirtschaft GmbH“ that manages production on these systems

Issue:

- Differently than on Urika-GX, ML tools such as Apache Spark are not provided by default in HWW! => **work in progress**

Code as a Python SCRIPT

HLRS HWW Systems:

- **Hawk**: AMD CPUs not optimized for ML/DL (yet)

See benchmark:
https://www.anandtech.com/show/14466/intel-xeon-cascade-lake-vs-nvidia-turing

- **Vulcan**: NEC Cluster for the execution of parallel programs (CPUs and GPUs)

https://kb.hlrs.de/platforms/index.php/NEC_Cluster_Hardware_and_Architecture_(vulcan)

Code as a Python SCRIPT

Plan:

- From the interactive JN to running the example as a **script** on Urika-GX
- We are still using the JN but only as **text editor** (option to use e.g. vi)
- Run the script in Urika-GX
- Run the script as a batch job on Vulcan (completely new environment!)

Code as a Python SCRIPT – Urika

Goal: Run a script containing:

- Data Manipulation (ML_manipulation.py)
- ML Classification (ML_classification.py)

- **Main program** and Classification Visualization (**ML_start.py**)
- needed algorithms (ML_handlers.py)

Similarities to JN:

- The **procedure** stays the same as in the Jupyter Notebook!
- Lustre source files stays the same.

… and differences to JN:

- We keep the **I/O data** separated (new directories and HDFS file names)
- Some nice features / shortcuts of the JN are **lost**: **next slides**.

Code as a Python SCRIPT – Urika

The JN “magics” **%** (or **!** = execute in command line) must be replaced:

- %spark => import and initialize a spark SESSION:

```
from pyspark.sql import SparkSession
...
spark = SparkSession.builder.getOrCreate()
```

- %run Handlers.ipynb => explicitly open and read the handlers notebook:

```
exec(open("EX_TN_ML_handlers.py").read())
```

- The same for EX_TN_ML_manipulation and EX_TN_ML_classification!

Code as a Python SCRIPT – Urika

- %project PROJECTNAME

Creates a path for pip packages: environment variable and sys.path are updated to install custom packages.

Every step must be done now manually in an **extra script**: **EX_TN_exportfile.sh**

(See also https://janakiev.com/blog/python-shell-commands/ for instructions to include shell commands into python.)

Code as a Python SCRIPT

- The script can be run with **3 commands in one single line**:

```
. EX_TN_exportfile.sh && spark-submit  --name SBahn_script --executor-memory
40G --total-executor-cores 5 EX_TN_ML_start.py > EX_TN_output2.txt ; sed '/^....-
..-.. ..:..:.. INFO/d' EX_TN_output2.txt > EX_TN_output.txt
```

&& : Execute the next command only if the previous one succeeded.

; : Execute the next command regardless of the success of the previous one.

Details: next slides.

Code as a Python SCRIPT

- Inline options for spark-submit:

  Note: We want to replicate the JN “magic”

  %spark 5 40g

  => spark-submit --name SBahn_script --executor-memory **40**G --total-executor-cores **5** EX_TN_ML_start.py > EX_TN_output2.txt

  – Which memory size for each executor (=node): 500 GB / node (**--executor-memory**)

  – How many cores we want to use (**--total-executor-cores**)

  (if exceeded, other jobs will be blocked on this node!)

  – Forward the output to the file **EX_TN_output2.txt**

Code as a Python SCRIPT – advanced

Other options for spark-submit:

- **--master** : specify on which Spark cluster (already preconfigured correctly => no need). E.g.

  --master local[*]

  would run **on the login node** with the specified number of worker threads (cores). If not specified => 1 core.

- **--num-executors** : on how many executors (i.e. nodes) we want to distribute the job **(communication overhead!)**

See also (some references on tuning Spark):

https://spark.apache.org/docs/latest/submitting-applications.html

https://docs.cloudera.com/runtime/7.1.0/running-spark-applications/topics/spark-submit-options.html

https://www.slideshare.net/jcmia1/apache-spark-20-tuning-guide

Code as a Python SCRIPT

- The output contains a lot of lines “INFO ----------” that we can delete for readability:

  sed '/^....-..-.. ..:..:.. INFO/d' EX_TN_output2.txt > EX_TN_output.txt

  – gives “sed” a line pattern to delete
  – EX_TN_output2.txt can be opened any time
  – EX_TN_output.txt contains the cleaned output after run is finished

- (not implemented) In order not to print any log messages at all:

https://community.cloudera.com/t5/Support-Questions/Spark-job-submit-log-messages-on-console/td-p/163043

Code as a Python SCRIPT – Visualization **SKIP**

Let us have a quick look at how we can visualize our classification results:

Browse to the folder:

> cd /home/users/**vdl1XXX**/hpda-code/use-cases/sbahn/scripts

And open in your Notebook (or text editor):

EX_TN_ML_start.py

Code as a Python SCRIPT – Visualization **SKIP**

Goal: Visualize the classification results on a **map**.

- Plot packages to pip-install (look for the pip install up in the script):
  - **geopy**: Python client to locate coordinates using geocoders
    https://geopy.readthedocs.io/en/stable/
  - **folium**: library to visualize data on a map.
    https://python-visualization.github.io/folium/

Code as a Python SCRIPT – Visualization **SKIP**

Further down (at ML CLASSIFICATION)

- Read-in the results of ML Classification run on the complete dataset (with Gilgamesch)…
- .. And convert this spark DataFrame into a pandas DataFrame: df_accu_pd
- It contains the **accuracy** of the predicted delay:
  - Computed on all test samples of the **S1 test dataset**
  - Averaged **for each station**
  - For the **five sets** of feature columns => **five different plots**

Code as a Python SCRIPT – Visualization **SKIP**

(do threshold coding)

- Generate two lists of stations:
  - All **ordered** stations on the S1 line
  - S1 stations according to the accuracy DataFrame (**not ordered**)
- Produce a python **dictionary of stations** *DS100 : station name*, e.g. *TB : Backnang*
- Define a **threshold** for the accuracy colour-code on the map:

acceptable (>=0.8), borderline (0.5<= t < 0.8), trash (<0.5)

=> A colour is assigned to each station.

Code as a Python SCRIPT – Visualization **SKIP**

- The plot functions are defined in Handlers (class “PredictionVisualize”):
  - **getCoords**: Associate a colour to each station based on the accuracy,
  - … and find the location of every station with Geopy.
  - **drawMapDic**: Draw the map with folium.

We will see at the end of the Exercise.

Code as a Python SCRIPT – EXERCISE

Start sharing Notebook…

Run the ML code as a script executing the following steps:

- **In the Enkidu terminal**, be in the correct directory:

> cd /home/users/**vdl1XXX**/hpda-code/use-cases/sbahn/scripts

- Activate Python and Internet for the pip-installs:

~~> module load python/3.6~~

> ~~module load tools/proxy~~

> module load workshop

For the next steps: **Jupyter Notebook OR a text editor (e.g. vi) are needed!!**

Code as a Python SCRIPT – EXERCISE cont.

- **OPTION:** To use **EX_TN_SOL_exportfile.sh** and **EX_TN_SOL_ML_start.py**

(the first line with the **username** must still be changed in **both files**!).

Otherwise:

- Replace _____ in **EX_TN_exportfile.sh**:
  - Define **MYUSERNAME**
- Replace **all** _____ in **EX_TN_ML_start.py**
  - Define also here username_local
  - Use the commands in slide ref for the other gaps
- **Run** the script **in the command window** as in slide ref

Code as a Python SCRIPT – EXERCISE cont.

Share Terminal…

- Check successful output:

> cat EX_TN_output.txt

… *Success!!!*

- Or **correct the errors** and **re-run**!
- **geocode** might be slow at the first run (requires a second one)
- **pip install** might not work with “crowd”

((Optional))

- Type in the command line **minfo** to monitor which resources you are actually using
- Monitor the output interactively: In the JN, open **EX_TN_output.txt** and refresh while still running

Code as a Python SCRIPT – EXERCISE cont.

Remark:

In EX_TN_ ML_start.py:

run_manipulation = False

run_ml_classification = False

In the interest of time, we are **reading-in** manipulation and ML results from **my hdfs directory**! (instead of running every step)

If no path is specified, **I/O** in manipulation and classification would happen in the **local hdfs directory** (your directory).

Code as a Python SCRIPT – EXERCISE cont.

- You should have obtained the Random Forest plots in (JN or terminal):

… sbahn/ScriptPlot/ClassificationPlot

- **These plots cannot be directly opened in the JN browser profile (NO INTERNET)!**

- They are available in

https://fs.hlrs.de/people/zanon/S_Bahn_class_plot

Code as a Python SCRIPT – EXERCISE cont.

- **Option**: Download them on your local system.

Works **only** with direct login to Enkidu (=Uni-Stuttgart VPN, sorry for the inconvenience)! **In a local window**:

> cd ~/Desktop

> mkdir scriptPlot

> cd ./scriptPlot

> scp **vdl1XXX**@enkidu-login1.hlrs.de:/home/users/vdl1XXX/hpda-code/use-cases/sbahn/ScriptPlot/ClassificationPlot/Plotavg* ./

Then, open each plot **with a browser**, e.g. Chrome.

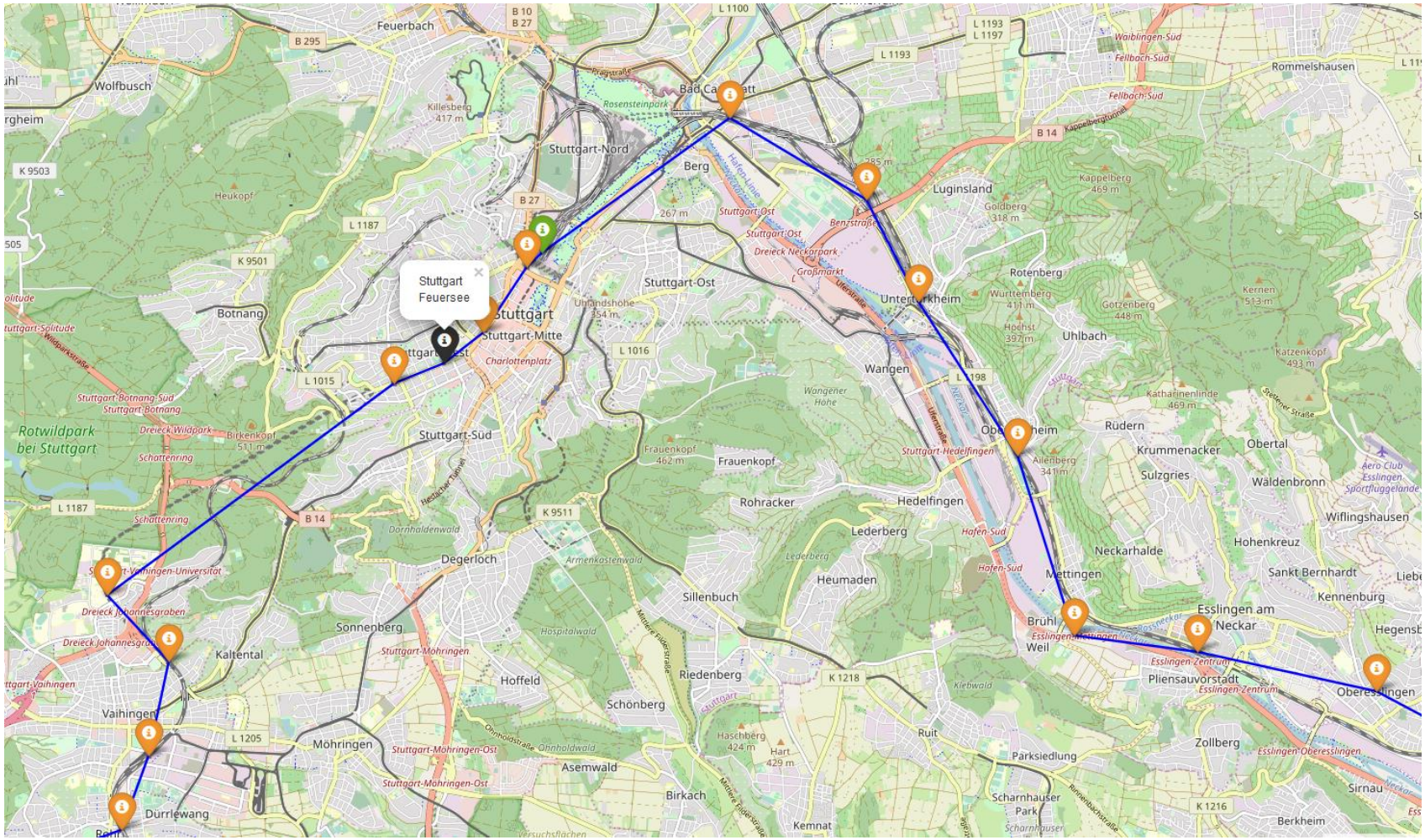
Code as a Python SCRIPT – EXERCISE cont.

Share slides…

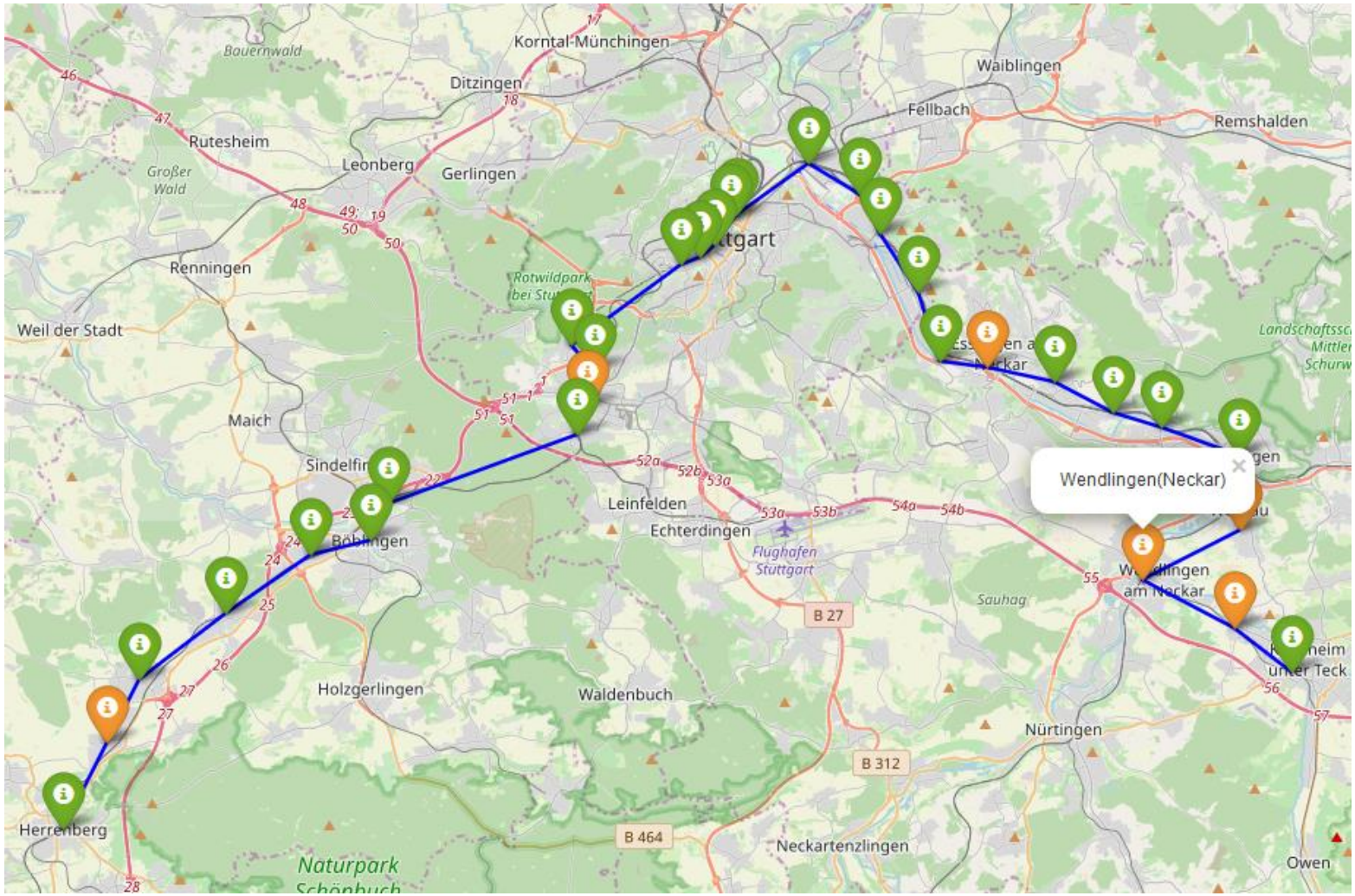
Feature columns recap:

1) Basic information of the original train dataset (5)
2) Basic information, weather data (11)
3) Delay at station -1 (additional features) (1)
4) Delay at stations -1,-2 (add. features) (2)
5) Basic information, delays and duration (add. features), weather data (8)

Code as a Python SCRIPT – EXERCISE cont. Feature set 2



Code as a Python SCRIPT – EXERCISE cont. Feature set 5



Code as a Python SCRIPT

- **Close** the JN browser
- **Close** the connection to Enkidu: Close all terminal windows, Putty, MobaXterm, …

Log in and set up – Vulcan

Disclaimer:

ML on Vulcan is a work in progress!

Goal:

- General introduction on using HWW resources
- … and ML tools on them (e.g. Pyspark)
- Many information not restrictive to Vulcan (also for using on Hawk).

Log in and set up – Vulcan – **requirement slide**

Set up your access to **Vulcan**:

- **Close any other active VPN**
- Open Fortinet or the Open Source equivalent (with your **username (vdl1XXX)** and **password**):

https://kb.hlrs.de/platforms/index.php/VPN

- Open a new command window and login with your **username (vdl1XXX)** and **password**:

> ssh -X vdl1XXX@vulcan.hww.hlrs.de

- Windows: You can use putty (next 2 slides)

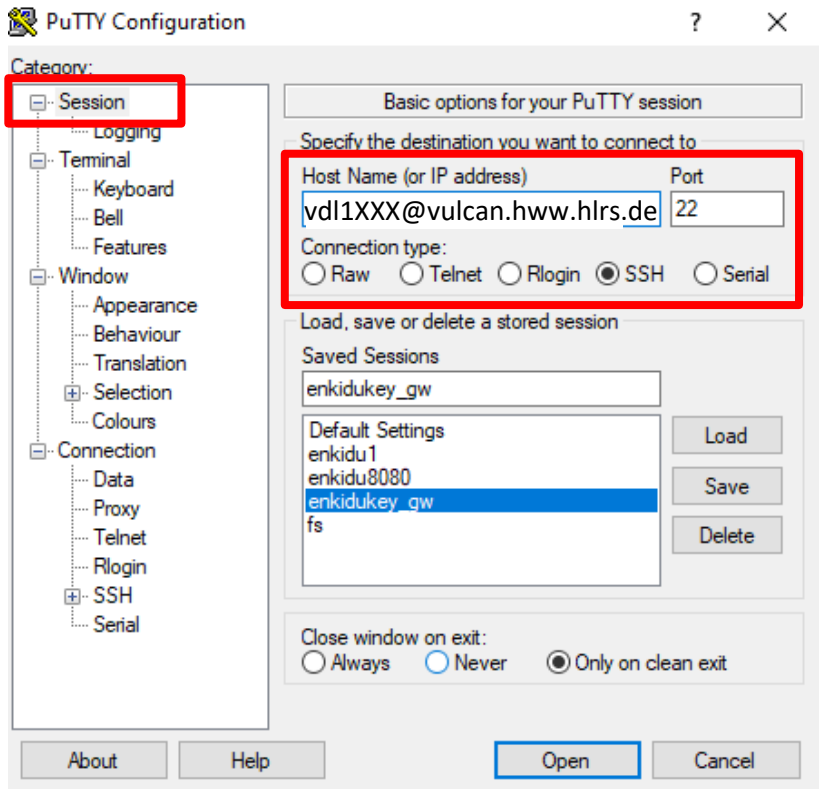
https://www.chiark.greenend.org.uk/~sgtatham/putty/latest.html

Log in and set up – Vulcan – **requirement slide**

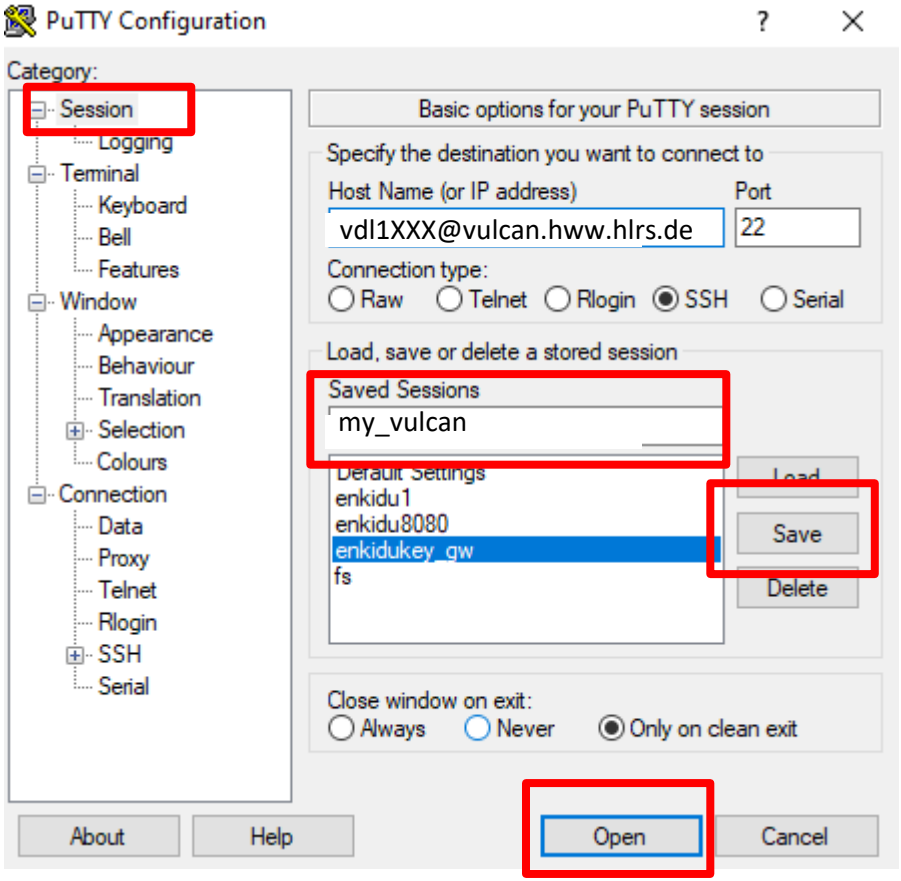
Option: Open putty:

- Set the Host Name (vdl1XXX = your username):

vdl1XXX@vulcan.hww.hlrs.de



Log in and set up – Vulcan – **requirement slide**



- Enter a name in "**Saved Sessions**"
- **Save** for the next logins
- (Click on **Load** to use this session again)
- **Open**

Log in and set up – Vulcan

- Text editing:

vi, vim, emacs, gedit

- **vi** cheat-sheet:

https://fs.hlrs.de/people/zanon/vi_cheat_sheet.pdf

- **gedit** requires X11 enabled. In that case, use a **separate login**:

> ssh -X vdl1XXX@vulcan.hww.hlrs.de

> gedit

Log in and set up – Vulcan

What is needed in Vulcan:

1. S-Bahn script and source data
2. Required software (spark)
3. To submit a batch job:
   - ***Frontend nodes:*** *are intended as single point of access to the entire cluster. Here you can set your environment, move your data, edit and compile your programs and* ***create batch scripts****. Interactive usage* ***like run your program*** *which leads to a high load is* ***NOT allowed*** *on the frontend/login nodes.*
   - ***Compute nodes*** *for running parallel jobs are only available through the* ***batch system.***

https://kb.hlrs.de/platforms/index.php/NEC_Cluster_access_(vulcan)

Vulcan **DATA**

- Source data are again stored in a **Lustre** system in Urika. Vulcan provides the Lustre filesystem NEC_lustre: https://kb.hlrs.de/platforms/index.php/NEC_Cluster_Disk_Storage_(vulcan)
- **Lustre** system is accessible only with via **workspaces**, as in the next slides.

  See also: https://kb.hlrs.de/platforms/index.php/Workspace_mechanism

Vulcan **DATA**

- Workspaces…
  - allocate disk space for your jobs
  - have an identifier (a name)
- A workspace can be generated with **ws_allocate** and its path stored to an environmental variable:

MYSCR=`ws_allocate *workspaceFavouriteName #days*`

echo $MYSCR

(the default path is a **lustre** path!)

- Workspaces **expire**! Can be extended, retrieved from trash, reminders can be sent automatically.

Vulcan **DATA** – advanced / optional

- The tool **ws_exchange** allows for flexible exchange of data among users **within their workspaces**: https://kb.hlrs.de/platforms/index.php/CAE_utilities#ws_exchange_procedure It creates:
  - a new **temporary** workspace with protected content
  - a **subdirectory** with random name (but public r/w/x permission).
- This is what we are going to use to exchange data.

- **ws_cp2exchange** is a special command which enables copying your data directly in the exchange subdirectory.

Vulcan **DATA**

PRACTICAL

Create your workspace and copy there data and scripts necessary for the S-Bahn example.

Next slides.

Vulcan **DATA – requirement slide**

One-time operation!

Once you have logged in, create a workspace (with **name and duration** = e.g. 20 days) and navigate to that workspace:

> MYSCR=`ws_allocate *workspaceFavouriteName 20*`

> cd $MYSCR

At every new log-in or during the course:

Recover the workspace id with ws_list (line **id: …**):

> ws_list

Define the **path** variable MYSCR:

> MYSCR=`ws_find *corresponding_id*`

Navigate to your workspace:

> cd $MYSCR

Careful with the quotes!!! `…`

Vulcan **DATA – requirement slide** – data already there!

```
> cd $MYSCR
```

In the workspace, copy the following folder:

```
> scp -r /lustre/nec/ws2/ws/hpclzano-exchange20200610T184027/gahbNFVollg/sbahn_data .
```

(directory with the **source files** of the S-Bahn ML example)

Unzip the content of sbahn_data:

```
> cd sbahn_data
> unzip S-BahnStuttgartDaten+Wetter.zip
> ls
```

You should see 9 .csv + 1 .xls files, and the .zip file.

Vulcan **DATA**

PRACTICAL (download script)

Navigate to your workspace:

> **cd $MYSCR**

Copy the following folders from my exchange directory:

> scp -r /lustre/nec/ws2/ws/hpclzano-exchange20200610T184027/gahbNFVollg/scripting_TN .

(directory with the **S-Bahn scripts**)

Vulcan **SOFTWARE**

- HWW systems have a **module** system:

https://kb.hlrs.de/platforms/index.php/NEC_Cluster_Software_Environment_(vulcan)

- Modules can be loaded / unloaded
- The environmental setting (= loaded packages) will **not** be saved and will be **lost** for a new session:
- … A new session (login, new job, **compute node**) will have the default HWW environment.
- Modules support **multiple versions** of software
- Check with *module avail*, *module list*

Vulcan **SOFTWARE**

PRACTICAL

- Multiple versions of python are available:

> **module load python/3.6**

Vulcan **SOFTWARE**

- Python alone is of course not sufficient
- We need a full Spark installation: *work in progress…*
- A simpler step (though not sufficient to run the full example!): To get **Pyspark** (Python API for Spark)

https://pypi.org/project/pyspark/

- As with many other Python packages, it can be obtained via the Python package installer:

pip install

Vulcan **SOFTWARE** – advanced / optional

- General **Internet** is not available in the HWW systems!
- Use an “ssh tunnel” to a local machine as a replacement:

https://kb.hlrs.de/platforms/index.php/Secure_Shell_ssh

- … for **pip install** see in particular:

https://kb.hlrs.de/platforms/index.php/Secure_Shell_ssh#pip_.28Python_package_installer.29

- The additional packages will be **locally** available on the python **module** used for the pip install (e.g. python/3.6).

You can try it yourself before your login expires (Monday, July 20th) .

Vulcan **SOFTWARE**

In our case:

- The extra-packages are already in the shared directory.
- In the main python script **ML_start.py**, an additional line points to **the repository**:

sys.path.append("/lustre/nec/ws2/ws/hpclzano-exchange20200610T184027/gahbNFVollg/site-packages")

Mini-Practical

> ll /lustre/nec/ws2/ws/hpclzano-exchange20200610T184027/gahbNFVollg/site-packages

… shows what has been pip-installed (pyspark, but also other packages!).

Vulcan **SOFTWARE**

- **Spark** is available in Vulcan on the Urika-CS Container:

https://kb.hlrs.de/platforms/index.php/Urika_CS

- It can run on the AI/Big Data nodes:
  - CS-500 [clx-21] = 8 nodes 2 x 20 core-CPU per node (**Spark**)
  - CS-Storm [clx-ai] = 1 node 2 x 18 core-CPU + 8 GPU's per node

More on the architecture of Vulcan:

https://kb.hlrs.de/platforms/index.php/NEC_Cluster_Hardware_and_Architecture_(vulcan)

[oleksandr.shcherbakov@hlrs.de]

Code as a Python SCRIPT – Vulcan – **BATCH JOBS**

PRACTICAL (1 slide)

Navigate to the script directory:

> cd $MYSCR/scripting_TN

> ll

You should find:

- The ***.py** files (main file is **ML_start.py**)
- submit_job_TN.pbs : We will see later.

Code as a Python SCRIPT – Vulcan – **BATCH JOBS**

- We would be tempted to already start the job with (as in Urika):

> spark-submit --name S[illegible]ahn_script[illegible]-executor-memory 80G --total-executor-cores 20 [illegible]L_start[illegible]y > output.txt

Not on the login node!

Code as a Python SCRIPT – Vulcan – **BATCH JOBS**

Without a full spark installation, **spark-submit is not available!**

Equivalent spark options can be nevertheless defined inside the main python script **ML_start.py**:

spark = SparkSession.builder.appName("SBahn").config**("spark.total.executor.cores", "20")**.config**("spark.executor.memory", "80g").master("local")**.getOrCreate()

Code as a Python SCRIPT – Vulcan – **BATCH JOBS**

Nevertheless…

- Spark is not fully installed, many commands including HDFS I/O would not work

Advanced:

- **“master local”:** There is **no Spark standalone cluster** on Vulcan, then the job will run locally on the **compute node**

https://spark.apache.org/docs/latest/cluster-overview.html

Code as a Python SCRIPT – Vulcan – **BATCH JOBS**

- **Compute nodes** have 4 characteristics:
  - node_type: node ID
  - node_type_cpu: CPU name
  - node_type_mem: memory on this node
  - node_type_core: number of cores on this node
- All information about running jobs on the compute nodes:

https://kb.hlrs.de/platforms/index.php/Batch_System_PBSPro_(vulcan)

Code as a Python SCRIPT – Vulcan – **BATCH JOBS**

- E.g. selecting **4 nodes** of type:

							128GB memory			
hsw	Haswell@2.60GHz	256gb	20c			QDR	Intel Xeon E5-2660v3 @ 2.60GHz, Haswell, 256GB memory	2 x 10 core-CPU per node	4	0
hsw	Haswell@2.50GHz	128gb	24c			QDR	Intel Xeon E5-2680v3 @ 2.50GHz, Haswell, 128GB memory	2 x 12 core-CPU per node	152	192 (1 is in shared mode)
hsw	Haswell@2.50GHz	128gb	24c			FDR	Intel Xeon E5-2680v3 @ 2.50GHz, Haswell, 128GB memory	2 x 12 core-CPU per node	0	0
hsw	Haswell@2.50GHz	256gb	24c			QDR	Intel Xeon E5-2680v3 @ 2.50GHz, Haswell, 256GB memory	2 x 12 core-CPU per node	16	0
skl	Skylake@2.0GHz	192gb	40c			EDR	Intel Xeon Gold 6138 @ 2.00GHz, Skylake, 92GB memory	2 x 20 core-CPU per node	100	0
smp	Skylake@2.40GHz	1536gb	40c			FDR	Intel Gold 6148 @ 2.40GHz, Skylake, 1.5TByte memory	2 x 20 core-CPU per node will be shared with other jobs! Please use "qsub -q smp -l	1 (shared)	0

type **type_mem** **type_CPU** **type_core**

… would imply:

= 4 X 24 cores = 96 total core-CPUs; executor memory up to 128 GB per node

Code as a Python SCRIPT – Vulcan – **BATCH JOBS**

- Job submission takes place through a **job script**
- … where at least three job characteristics must be specified:
  - **Number** of nodes
  - **At least one** node variable (of the four above)

  -l select=#nodes:node_...=…
  - **Walltime**:

  -l walltime=hh:mm:ss

  (allocated nodes will not be shared with other jobs!!)

Code as a Python SCRIPT – Vulcan – **BATCH JOBS**

- The job characteristics should **at least** match the resources required by the spark session in the python script.
- E.g. **4 hsw nodes** would match:

```
spark = SparkSession.builder.appName("SBahn").config("spark.total.executor.cores", "96").config("spark.executor.memory", "128g").master("local").getOrCreate()
```

Code as a Python SCRIPT – Vulcan – **BATCH JOBS**

PRACTICAL (3 slides: share terminal)

> cd $MYSCR/scripting_TN

- Open the submission script with (or with another text editor: emacs, gedit, …)

> **vi submit_job_TN.pbs**

- Edit (type *a*) **all the gaps** "_____" to:
  - … request **1 node** of type **hsw** for **20 minutes** (see previous slides)
  - … correctly start the job (other gaps)
- To save and quit, type: *ESC* + *:wq*
- If you have issues with **vi**, just have a look at my solution (cannot be used to run the job!):

> **cat submit_job_TN_SOL.pbs**

Code as a Python SCRIPT – Vulcan – **BATCH JOBS**

PRACTICAL

- Submit the job with (-q … : only vdl1XXX accounts!):

> **qsub** -q dlagpu submit_job_TN.pbs

- Type:

> **qstat**

to see if the job is running, queued, **finished (= no output)** etc.

For many other monitoring options, visit:

https://kb.hlrs.de/platforms/index.php/Batch_System_PBSPro_(vulcan)

Code as a Python SCRIPT – Vulcan – **BATCH JOBS**

PRACTICAL

- Type:

> **cat output_batch.txt**

to see the successful output once the job is finished:

Vulcan exit: …

In case of failure, you can read the **numbered** error file:

> cat my_sbahn.eXXXXX

Code as a Python SCRIPT – Vulcan – **BATCH JOBS**

If we are short on time: go to this slide

- In case of small jobs (as this one): debugging or optimizing code but not for production,
- (or if you had troubles using vi…)

jobs can be run **interactively on the compute node**, without the need of a script.

Please do the **next practical** to submit the same job interactively.

Code as a Python SCRIPT – Vulcan – **BATCH JOBS**

PRACTICAL

- We first request the compute node resources with qsub -I (-q … : for vdl1XXX accounts!), **one line**:

```
> qsub -q dlagpu -I -l
select=1:node_type=hsw -l
walltime=00:20:00
```

… and wait for availability.

- We then see on which compute node we are directed.

Code as a Python SCRIPT – Vulcan – **BATCH JOBS**

PRACTICAL

We need again:

- python (and installed packages) available on the compute node
- **to redefine the environment** on the compute node (every new submitted job / access to the compute node will have the default HWW environment!)

Execute:

> **module load python/3.6**

> **ws_list**

> **MYSCR=`ws_find** *workspaceFavouriteName*`

> **cd $MYSCR/scripting_TN**

> **export MYWS=$MYSCR**

Code as a Python SCRIPT – Vulcan – **BATCH JOBS**

PRACTICAL

- Finally launch interactively the python script:

> **python ML_start.py > output_script_int.txt**

- The output text file at completion:

> **cat output_script_int.txt**

- To exit batch system and go **back to the login** node:

> **logout**

Code as a Python SCRIPT – Vulcan – **BATCH JOBS**

Again, with only **pyspark installed**, we can only run up to one part of data manipulation…

On Vulcan: Current **problem** at ML_handlers.py (dfSplit) when it comes to the Spark RDD (resilient distributed dataset) and therefore HDFS files (to operate on data in parallel):

https://spark.apache.org/docs/latest/rdd-programming-guide.html

“use a build of PySpark linking to your version of HDFS”

… which needs a complete spark installation!

(work in progress…)

… or using the Urika-CS Container.

Code as a Python SCRIPT – Vulcan – **BATCH JOBS**

> exit

And close the **VPN**

Actually stay in Vulcan for the next part. Go back to the workspace:

> cd $MYSCR

Scripts and solutions

All scripts and solutions are available in fs:

https://fs.hlrs.de/people/zanon/scripts_solutions

In case you want to copy your own work (accounts expire on **Monday, July 20**):

Vulcan:

- In the Vulcan workspace, **pwd** tells you the exact path
- In a local terminal:
  - switch on the HWW VPN
  - type **one line** as below:

```
> scp -r
vdl1XXX@vulcan.hww.hlrs.de:/path/to/workspace/and/folder
/path/to/local/destination
```

Scripts and solutions

Enkidu

Only with Uni-Stuttgart VPN (“direct login”)! Otherwise send us an email. In a local terminal, **one** line as below:

```
> scp -r vdl1XXX@enkidu-login1.hlrs.de:/path/to/desired/folder
/path/to/local/destination
```