Author: Andreas Traut
Date: 08.05.2020 (Updates 24.07.2020)
Download as PDF

Machine Learning with Python

Machine Learning with Python

0. Introduction

a) Aim of this repository: "Small Data" versus "Big Data"

After having learnt visualization techniques in Python (which I showed in my repository "Visualization-of-Data-with-Python"), I started working on different datasets with the aim to learn and apply machine learning algorithms. I was particularly interested in better **understanding the differences and similarities of "Small Data" (Scikit-Learn) approaches versus the "Big Data" (Spark) approaches!**

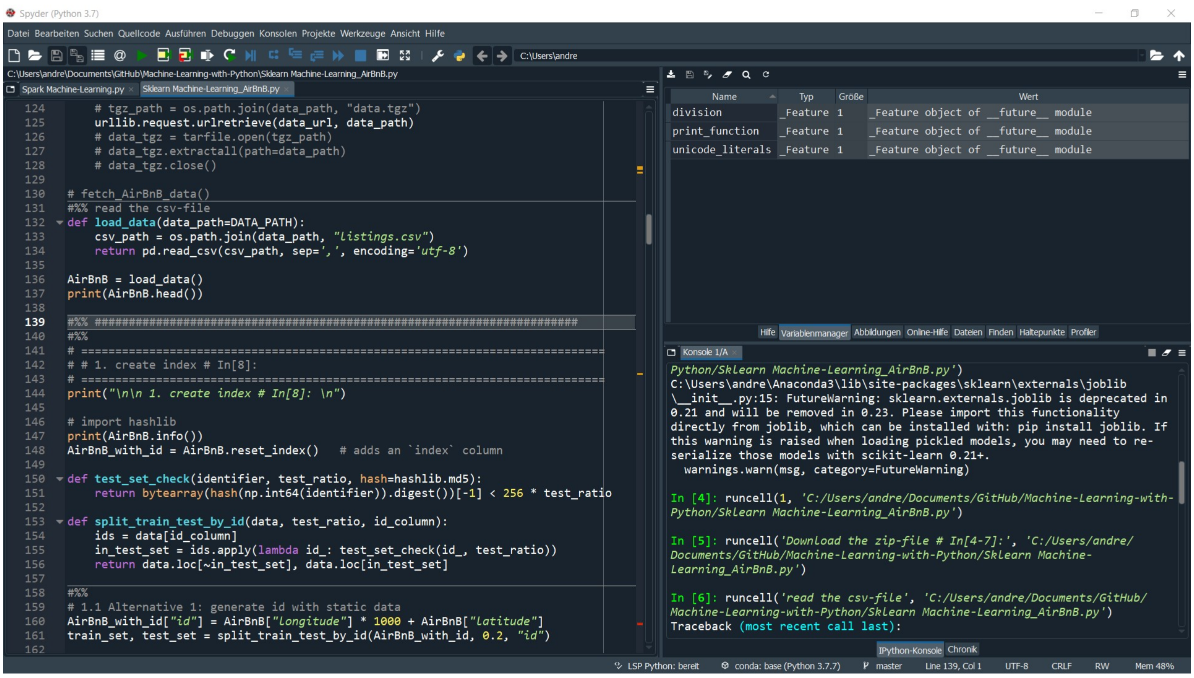
Therefore I tried to focus more on this "comparison" question of "Small Data" coding vs "Big Data" coding instead of digging into too many details of each of these approaches. I haven't seen many comparisons of "Small Data" vs "Big Data" coding and I think understanding this is interesting and important.

b) Motivation for IDEs

I will use Jupyter-Notebooks, which is a widespread standard today, but I will also use Integrated Development Environments (IDEs). The first Jupyter-Notebooks have been developed 5 years ago (in 2015). Since my first programming experience was more than 25 years ago (I started with GW-Basic then Turbo-Pascal and so on and I am also familiar with MS-DOS). I quickly learnt the advantages of using Jupyter-Notebooks. **But** I missed the comfort of an IDE from the very first days!

Why is it important for me to mention the IDEs out so early in a learning process? In my opinion Jupyter-Notebooks are good for the first examinations of data and for documenting procedures and up to a certain degree also for sophisticated data science. But it might be a good idea to learn very early how to work with an IDE. Think about how to use what has been developed so far later in a bigger environment (for example a Lambda-Architecture, but you can take whatever other environment, which requires robustness&stability). I point this out here, because after having read several e-Books and having participated in seminars I see that IDEs are not in the focus.

Therefore: in my examples in this repository here I will also work with Python ".py" files. These ".py" can be executed in an IDE, like e.g. Spyder-IDE, which can be downloaded for free and looks like this:



c) Structure of this repository

(i) First part: "Movies Database" example

Therefore the *first example* uses a Jupyter-Notebook in order to learn the standard procedures (e.g. data-cleaning & preparing, model-training,...). I worked on data converting movies and their revenues.

(ii) Second part: Scikit-Learn Example ("Small Data")

The *second example* is for being used in an IDE (integrated developer environment), like the Spyder-IDE from the Anaconda distribution and apply the "Scikit-Learn Python Machine Learning Library" (you may call this example a "Small Data" example if you want). I will show you a typical structure for a machine-learning example and put it into a mind-map. The same structure will be applied on the third example.

(iii) Third part: Spark Example ("Big Data")

The *third example* is a "Big Data" example and will use a Docker environment and apply the "Apache Machine Learning Library", a scalable machine learning library. The mind-map from the second part will be extended and aligned to the second example.

In this example I also show some *Big Data Visualizations techniques*, show how the *K-Means Clustering Algorithm in Apache Spark ML* works and explain the *Map-Reduce* programming model on a Word-Count example.

(iv) Summary Mind-Map

I provide a summary mind-map, which possibly helps you to structure your code. There are lots of similarities between "Small Data" and "Big Data".

(v) Digression (Excurs) to Big Data Visualization and K-Means Clustering Algorithm and Map-Reduce

In this Digression (Excurs) I will provide some examples for Big Data Visualization, K-Means Clustering and Map-Reduce.

d) Future learnings and coding & data sources

For all of these topics various tutorials, documentation, coding examples and guidelines can be found in the internet **for free**! The Open Source Community is an incredible treasure trove and enrichment that positively drives many digital developments: Scikit-Learn, Apache Spark, Spyder, GitHub, Tensorflow and also Firefox, Signal, Threema, Corona-Warnapp... to be mentionned. There are many positive examples of sharing code and data "for free".

Coding:

If you Google for example *"how to prepare and clean the data with spark"*, you will find tons of documents around *"removing null values"* or *"encoders"* (like the OneHotEncoder for treating categorical inputs) or *"pipelines"* (for putting all the steps in an efficient, customizable order) so on. You will be overwhelmed of all this. Some resources to mention are the official documentation and a few more Github repositories like e.g. tirthajyoti/Spark-with-Python (MIT licence), Apress/learn-pyspark (Freeware License), mahmoudparsian/pyspark-tutorial (Apache License v2.0). What I will do here in my repository is nothing more than putting it together so that it works for my problem (which can be challenging as well sometimes). Adapting it for your needs should be easier from this point on.

Data:

If you would like to do further analysis or produce alternate visualizations of the Airbnb-data, you can download them from here. It is available below under a Creative Commons 1.0 Universal "Public Domain Dedication" license. The data for the Vermont-Vendor-Payments can be downloaded from here and are available under the Open Data Commons Open Database License. The movies database doesn't even mention a license and is from Kaggle. There you find a lot of more datasets and also coding examples for your studies.

I. "Movies Database" Example

A good starting point for finding useful datasets is "Kaggle" (www.kaggle.com). I downloaded the movies dataset from here. The dataset from Kaggle contains the following columns:

Rank | Title | Year | Score | Metascore | Genre | Vote | Director | Runtime | **Revenue** | Description | RevCat

In this example I want to predict the **"Revenue"** based on the other information, which I have for each movie (e.g. every movie has a year, a scoring, a title ...). There are some "NaN"-values in the column "Revenue" and instead of filling them with an assumption (e.g. median-value) as I did in another Jupiter-Notebook (see here), I wanted to predict these values. You might guess the conclusion already: predicting the revenue based on the available information as shown above (the columns) might not work. But essential to me is more to follow a well established standard-process of data-cleaning, data-preparing, model-training and error-calculation in this example in order to learn how to apply this process to better datasets, than the movies-dataset, later.

Therefore, here is how I approached the problem step-by-step:

1. Separate "NaN"-values

I separated the rows with "NaN"-values in column "Revenue"

These are the datarows, where column "Revenue" is null:

```
In [10]: movies_RevenueNaN = movies[movies["Revenue"].isnull()]
         movies_RevenueNaN.head()
```

Out[10]:

	Rank	Title	Year	Score	Metascore	Genre	Vote	Director	Runtime	Revenue	Description
82	83	A Clockwork Orange	1971	8.3	80.0	Crime, Drama, Sci-Fi	662768	Stanley Kubrick	136	NaN	In the future, a sadistic gang leader is impri...
513	514	To Kill a Mockingbird	1962	8.3	87.0	Crime, Drama	262064	Robert Mulligan	129	NaN	Atticus Finch, a lawyer in the Depression-era ...
581	582	Death Proof	2007	7.0	NaN	Action, Thriller	236539	Quentin Tarantino	113	NaN	Two separate sets of voluptuous women are stal...
620	621	My Neighbour Totoro	1988	8.2	86.0	Animation, Family, Fantasy	226126	Hayao Miyazaki	86	NaN	When two girls move to the country to be near ...
685	686	Hachi: A Dog's Tale	2009	8.1	NaN	Drama, Family	212349	Lasse Hallström	93	NaN	A college professor's bond with the abandoned ...

```
In [11]: len_movies_RevenueNaN = len(movies_RevenueNaN)
         len_movies_RevenueNaN
Out[11]: 2527
```

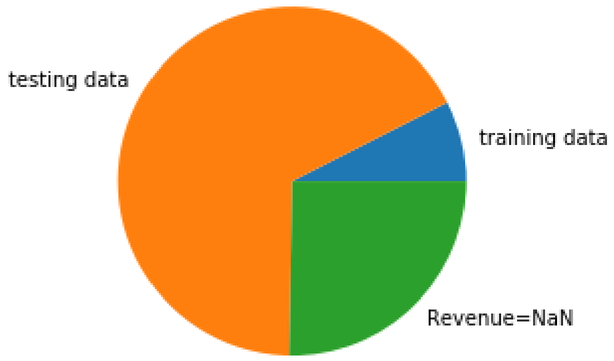
2. Draw a stratified sample

I drew a stratified sample (based on "Revenue") on this remaining dataset and I received a training dataset and testing dataset:

```
In [24]: split = StratifiedShuffleSplit(n_splits=1, test_size=0.9, random_state=42)
         for train_index, test_index in split.split(movies_NotNullC, movies_NotNullC["RevCat"]):
             strat_train_set = movies_NotNullC.iloc[train_index]
             strat_test_set = movies_NotNullC.iloc[test_index]
```

```
In [25]: strat_test_set["RevCat"].value_counts() / len(strat_test_set)
Out[25]: 1    0.903657
         2    0.069878
         3    0.016057
         4    0.010407
         Name: RevCat, dtype: float64
```

```
In [26]: movies_NotNullC["RevCat"].value_counts() / len(movies_NotNullC)
Out[26]: 1    0.903653
         2    0.069851
         3    0.016058
         4    0.010438
         Name: RevCat, dtype: float64
```



3. Create a pipeline

I created a pipeline to fill the "NaN"-value in other columns (e.g. "Metascore", "Score").

```
In [38]: num_pipeline = Pipeline([
             ('imputer', SimpleImputer(strategy='median')),
         ])

         num_attribs = ['Rank', 'Year', 'Score', 'Metascore', 'Vote', 'Runtime']

         full_pipeline = ColumnTransformer([
             ('num', num_pipeline, num_attribs)
         ])
```

Now apply the Pipeline:

```
In [40]: movies_train_prepared = full_pipeline.fit_transform(movies_train)
```

Now let's count the "nan" values in the new prepared datast "movies_train_prepared". I have to transform it back to a Pandas-Dataframe format first:

```
In [41]: tmp_num = movies_train.select_dtypes(include=[np.number])
         tmp_prep = pd.DataFrame(movies_train_prepared, columns=tmp_num.columns,index=movies_train.index)
         tmp = tmp_prep[tmp_prep["Metascore"].isnull()]
         tmp
```

Out[41]:

Rank	Year	Score	Metascore	Vote	Runtime

Zero, as we wanted! All "nan"-values in "movies_train_prepared" have been removed by the "median" value (this was how the pipeline was built). Great,

4. Fit the model

I used the training dataset and fitted it with the "DecisionTreeRegressor" model

```
In [42]: tree_reg = DecisionTreeRegressor(random_state=42)
         tree_reg.fit(movies_train_prepared, movies_train_labels)

         movies_predictions = tree_reg.predict(movies_train_prepared)
```

5. Cross-validation

I verified with a cross-validation, how good this model/parameters are

```
In [46]: scores = cross_val_score(tree_reg,
                                  movies_train_prepared,
                                  movies_train_labels,
                                  scoring="neg_mean_squared_error",
                                  cv=10)

         rmse_scores = np.sqrt(-scores)

         def display_scores(scores):
             print("Scores:", scores)
             print("Mean:", scores.mean())
             print("Standard deviation:", scores.std())

         display_scores(rmse_scores)

         Scores: [ 69.66166984  62.59977724  62.6450061  112.98391756  47.99491086
           52.16787811  42.65104005  83.21059338  41.09199207  63.84158945]
         Mean: 63.88483746577791
         Standard deviation: 20.447326452253154
```

6. Prediction

I did a prediction on a subset of the testing dataset and did a side-by-side comparison of prediction and true value

```
In [53]: side_by_side = [(true, pred) for true, pred in zip(list(some_data_label), list(some_data_predictions))]
         side_by_side

Out[53]: [(5.78, 10.91),
          (0.05, 0.44),
          (0.3, 2.68),
          (159.6, 11.99),
          (33.63, 2.19),
          (44.9, 26.83),
          (38.52, 22.52),
          (33.04, 2.19),
          (3.2, 11.99),
          (37.49, 16.38),
          (17.88, 64.19),
          (41.19, 191.45),
          (16.19, 12.19),
          (0.05, 3.02),
          (16.68, 64.19),
          (35.11, 11.54),
          (36.0, 40.22),
          (3.61, 19.64),
          (0.59, 0.05),
          (0.99, 5.48)]
```

I performed a prediction on the testing dataset and calculated the mean-squared error

```
In [55]: movies_test_prepared =full_pipeline.fit_transform(movies_test)
         movies_test_predictions = tree_reg.predict(movies_test_prepared)
         lin_mse = mean_squared_error(movies_test_labels, movies_test_predictions)
         lin_rmse = np.sqrt(lin_mse)
         lin_rmse

Out[55]: 54.68522284077671
```

7. Conclusion

The conclusion of this machine learning example is obvious: it is rather not possible to predict the "Revenue" based on the available information (the most useful numerical features were "year", "score", ... and the other categorical like "genre" don't seem to have much more added value in my opinion).

Please find the complete Jupyter Notebook here:

https://github.com/AndreasTraut/Machine-Learning-with-Python/blob/master/Movies%20Machine%20Learning%20-%20Predict%20NaNs.ipynb

If you want to run the code immediately without installing the required "Jupyter environment" then you can use this Deepnote-Link:

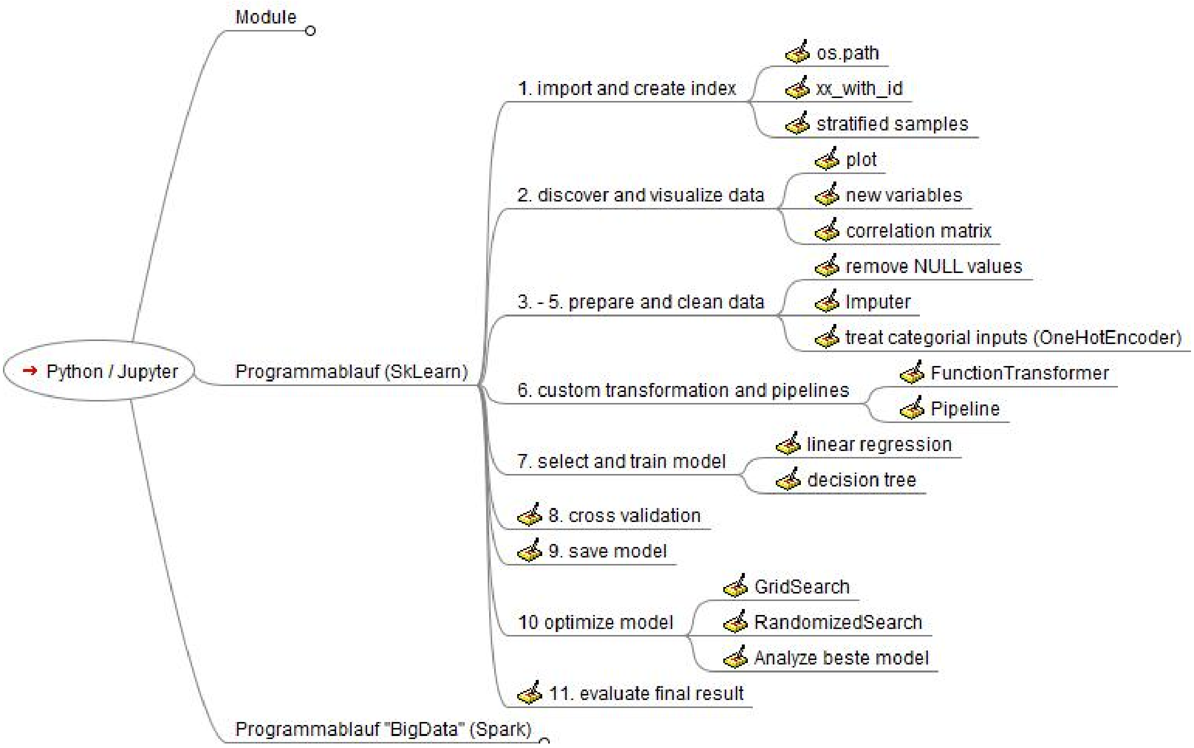
https://beta.deepnote.com/project/754094f0-3c01-4c29-b2f3-e07f507da460

II. "Small Data" Machine Learning using "Scikit-Learn"

In my opinion Jupyter Notebooks are **not** always the best environment for learning to code! I agree, that Jupyter Notebooks are nice for doing documentation of python code. It really looks beautiful. But I prefer debugging in an IDE instead of a Jupyter Notebook: having the possibility to set a breakpoint can be a pleasure for my nerves, specially if you have longer programs. Some of my longer Jupyter Notebooks feel from the hundreds line of code onwards more like pain than like anything helpful. And I also prefer having a "help window" or a "variable explorer", which is smoothly integrated into the IDE user interface. And there are a lot more advantages why getting familiar with an IDE is a big advantage compared to the very popular Jupyter Notebooks! I am very surprised, that everyone is talking about Jupyter Notebooks but IDEs are only mentioned very seldom. But maybe my preferences are also a bit different, because I grew up in a MS-DOS environment. :-)

I choose in this *second example* the Spyder-IDE and worked on "Scikit-Learn", a very popular python machine learning library. The basis of this code are some Jupyter-Notebooks, which Aurelien Geron provided (under the Apache License 2.0) in his book "Machine Learning with Scikit-Learn & Tensorflow". But as I didn't like at all that his code are Jupyter Notebooks (how can you re-use it efficiently for your own purposes?), so I wanted to work on it: I extracted the most essential parts of chapter 2, then sorted, arranged and modified the code fragments and created the following structured Python code. The structure of the Python code is a bit similar to the steps, which I followed in the Movies Database example above (you will find these sections also in the ".py" file).

So let's start with the "scikit-learn" ("SmallData", if you want). I will align this structure to the Spark "Big Data" mind map below in order to learn from each of this two approaches.



1. create index

1.1 Alternative 1: generate id with static data

```
# 1.1 Alternative 1: generate id with static data
housing_with_id["id"] = housing["longitude"] * 1000 + housing["latitude"]
train_set, test_set = split_train_test_by_id(housing_with_id, 0.2, "id")
```

1.2 Alternative 2: generate stratified sampling

```
# from sklearn.model_selection import StratifiedShuffleSplit
split = StratifiedShuffleSplit(n_splits=1, test_size=0.2, random_state=42)
for train_index, test_index in split.split(housing, housing["income_cat"]):
    strat_train_set = housing.loc[train_index]
    strat_test_set = housing.loc[test_index]
```

1.3 verify if stratified example is good

```
compare_props["Rand. %error"] = 100 * compare_props["Random"] / compare_props["Overall"] - 100
compare_props["Strat. %error"] = 100 * compare_props["Stratified"] / compare_props["Overall"] - 100
```

2. Discover and visualize the data to gain insights

```
# from pandas.plotting import scatter_matrix
attributes = ["median_house_value", "median_income", "total_rooms",
              "housing_median_age"]
scatter_matrix(housing[attributes], figsize=(12, 8))
plt.suptitle("scatter_matrix_plot")
save_fig("scatter_matrix_plot")
```

3. prepare for Machine Learning

3.1 find all NULL-values

```
print("Are there nans in column total_bedrooms?\n", housing["total_bedrooms"].isnull().any())
print("Show rows with nan:\n", housing[housing["total_bedrooms"].isnull()])
```

3.2 remove all NULL-values

```
sample_incomplete_rows = housing[housing.isnull().any(axis=1)].head()
# sample_incomplete_rows.dropna(subset=["total_bedrooms"])    # option 1
# sample_incomplete_rows.drop("total_bedrooms", axis=1)       # option 2

median = housing["total_bedrooms"].median()
sample_incomplete_rows["total_bedrooms"].fillna(median, inplace=True) # option 3
```

4. Use "Imputer" to clean NaNs

```
imputer = SimpleImputer(strategy="median")
# Remove all text attributes because median can only be calculated on numerical attributes:
housing_num = housing.select_dtypes(include=[np.number])
imputer.fit(housing_num)
print("imputer.strategy\n", imputer.strategy)
print("imputer.statistics_\n", imputer.statistics_)
print("housing_num.median\n", housing_num.median().values)  # <- Check that this is the same as \
                                                            # manually computing the median of \
                                                            # each attribute
print("housing_num.mean\n", housing_num.mean().values)      # <- Check that this is the same as \
                                                            # manually computing the median of \
                                                            # each attribute
X = imputer.transform(housing_num) # Transform the training set:
housing_tr = pd.DataFrame(X, columns=housing_num.columns,
                          index=housing.index)
```

5. treat "categorial" inputs

```
cat_encoder = OneHotEncoder()
housing_cat_1hot = cat_encoder.fit_transform(housing_cat)
```

6. custom transformer and pipelines

6.1 custom transformer

```
def add_extra_features(X, add_bedrooms_per_room=True):
    rooms_per_household = X[:, rooms_ix] / X[:, household_ix]
    population_per_household = X[:, population_ix] / X[:, household_ix]
    if add_bedrooms_per_room:
        bedrooms_per_room = X[:, bedrooms_ix] / X[:, rooms_ix]
        return np.c_[X, rooms_per_household, population_per_household,
                     bedrooms_per_room]
    else:
        return np.c_[X, rooms_per_household, population_per_household]

# from sklearn.preprocessing import FunctionTransformer
attr_adder = FunctionTransformer(add_extra_features, validate=False,
                                 kw_args={"add_bedrooms_per_room": False})
housing_extra_attribs = attr_adder.fit_transform(housing.values)

housing_extra_attribs = pd.DataFrame(
    housing_extra_attribs,
    columns=list(housing.columns)+["rooms_per_household", "population_per_household"],
    index=housing.index)
print("housing_extra_attribs.head()\n", housing_extra_attribs.head())
```

6.2 pipelines

```
num_pipeline = Pipeline([
        ('imputer', SimpleImputer(strategy="median")),
        ('attribs_adder', FunctionTransformer(add_extra_features,
                                              validate=False)),
        ('std_scaler', StandardScaler()),
    ])
housing_num_tr = num_pipeline.fit_transform(housing_num)
print("housing_num_tr\n", housing_num_tr)

try:
    from sklearn.compose import ColumnTransformer
except ImportError:
    from future_encoders import ColumnTransformer # Scikit-Learn < 0.20

num_attribs = list(housing_num)
cat_attribs = ["ocean_proximity"]

full_pipeline = ColumnTransformer([
        ("num", num_pipeline, num_attribs),
        ("cat", OneHotEncoder(), cat_attribs),
    ])
housing_prepared = full_pipeline.fit_transform(housing)
print("housing_prepared\n", housing_prepared)
```

7. select and train model

7.1 LinearRegression model

```
# from sklearn.linear_model import LinearRegression
lin_reg = LinearRegression()
lin_reg.fit(housing_prepared, housing_labels)
# let's try the full preprocessing pipeline on a few training instances
some_data = housing.iloc[:1]
some_labels = housing_labels.iloc[:1]
some_data_prepared = full_pipeline.transform(some_data)
print("Predictions:\n", lin_reg.predict(some_data_prepared))
print("Labels:\n", list(some_labels)) # Compare against the actual values:

# from sklearn.metrics import mean_squared_error
housing_predictions = lin_reg.predict(housing_prepared)
lin_mse = mean_squared_error(housing_labels, housing_predictions)
lin_rmse = np.sqrt(lin_mse)
print("lin_rmse\n", lin_rmse)
```

7.2 DecisionTreeRegressor model

```
# from sklearn.tree import DecisionTreeRegressor
tree_reg = DecisionTreeRegressor(random_state=42)
tree_reg.fit(housing_prepared, housing_labels)
housing_predictions = tree_reg.predict(housing_prepared)

tree_mse = mean_squared_error(housing_labels, housing_predictions)
tree_rmse = np.sqrt(tree_mse)
print("tree_rmse\n", tree_rmse)
```

8. crossvalidation

8.1 for DecisionTreeRegressor

```
# from sklearn.model_selection import cross_val_score
scores = cross_val_score(tree_reg, housing_prepared, housing_labels,
                         scoring="neg_mean_squared_error", cv=10)
```

8.2 for LinearRegression

```
lin_scores = cross_val_score(lin_reg, housing_prepared, housing_labels,
                             scoring="neg_mean_squared_error", cv=10)
```

8.3 for RandomForestRegressor

```
# from sklearn.ensemble import RandomForestRegressor
forest_reg = RandomForestRegressor(n_estimators=10, random_state=42)
forest_reg.fit(housing_prepared, housing_labels)

housing_predictions = forest_reg.predict(housing_prepared)
forest_mse = mean_squared_error(housing_labels, housing_predictions)
forest_rmse = np.sqrt(forest_mse)
print(forest_rmse)
# from sklearn.model_selection import cross_val_score
forest_scores = cross_val_score(forest_reg, housing_prepared, housing_labels,
                                scoring="neg_mean_squared_error", cv=10)
forest_rmse_scores = np.sqrt(-forest_scores)
display_scores(forest_rmse_scores)
```

8.4 for ExtraTreesRegressor

```python
from sklearn.ensemble import ExtraTreesRegressor
extratree_reg = ExtraTreesRegressor(n_estimators=10, random_state=42)
extratree_reg.fit(housing_prepared, housing_labels)

housing_predictions = extratree_reg.predict(housing_prepared)
extratree_mse = mean_squared_error(housing_labels, housing_predictions)
extratree_rmse = np.sqrt(extratree_mse)
print(extratree_rmse)
extratree_scores = cross_val_score(extratree_reg, housing_prepared,
                                   housing_labels,
                                   scoring = "neg_mean_squared_error", cv=10)
extratree_rmse_scores = np.sqrt(-extratree_scores)
display_scores(extratree_rmse_scores)
```

9. Save Model

```python
# from sklearn.externals import joblib
joblib.dump(forest_reg, "forest_reg.pkl")
# und später zum Laden des Modells...
my_model_loaded = joblib.load("forest_reg.pkl")
```

10. Optimize Model

10.1 GridSearchCV

10.1.1 GridSearchCV on RandomForestRegressor

```python
# from sklearn.model_selection import GridSearchCV
param_grid = [
    # try 12 (3×4) combinations of hyperparameters
    {'n_estimators': [30, 40, 50], 'max_features': [2, 4, 6, 8, 10]},
    # then try 6 (2×3) combinations with bootstrap set as False
    {'bootstrap': [False], 'n_estimators': [3, 10], 'max_features': [2, 3, 4]},
  ]

forest_reg = RandomForestRegressor(random_state=42)
# train across 5 folds, that's a total of (12+6)*5=90 rounds of training
grid_search = GridSearchCV(forest_reg, param_grid, cv=5,
                           scoring='neg_mean_squared_error',
                           return_train_score=True)
grid_search.fit(housing_prepared, housing_labels)
print(grid_search.best_params_)
print(grid_search.best_estimator_)
cvres = grid_search.cv_results_
for mean_score, params in zip(cvres["mean_test_score"], cvres["params"]):
    print(np.sqrt(-mean_score), params)
```

10.1.2 GridSearchCV on LinearRegressor

```
# from sklearn.model_selection import GridSearchCV
param_grid = [
    # try 12 (3×4) combinations of hyperparameters
    {'fit_intercept': [True], 'n_jobs': [2, 4, 6, 8, 10]},
    # then try 6 (2×3) combinations with bootstrap set as False
    {'normalize': [False], 'n_jobs': [3, 10]},
  ]

lin_reg = LinearRegression()
# train across 5 folds, that's a total of (12+6)*5=90 rounds of training
lin_grid_search = GridSearchCV(lin_reg, param_grid, cv=5,
                           scoring='neg_mean_squared_error',
                           return_train_score=True)
lin_grid_search.fit(housing_prepared, housing_labels)
# print(lin_grid_search.best_params_)
print(lin_grid_search.best_estimator_)
cvres = lin_grid_search.cv_results_
for mean_score, params in zip(cvres["mean_test_score"], cvres["params"]):
    print(np.sqrt(-mean_score), params)
```

10.2 Randomized Search

```
# from sklearn.model_selection import RandomizedSearchCV
# from scipy.stats import randint
param_distribs = {
        'n_estimators': randint(low=1, high=200),
        'max_features': randint(low=1, high=8),
    }

forest_reg = RandomForestRegressor(random_state=42)
rnd_search = RandomizedSearchCV(forest_reg,
                                param_distributions=param_distribs,
                                n_iter=10, cv=5,
                                scoring='neg_mean_squared_error',
                                random_state=42)
rnd_search.fit(housing_prepared, housing_labels)
cvres = rnd_search.cv_results_
for mean_score, params in zip(cvres["mean_test_score"], cvres["params"]):
    print(np.sqrt(-mean_score), params)
```

10.3 Analyze best models

```
feature_importances = grid_search.best_estimator_.feature_importances_
feature_importances
extra_attribs = ["rooms_per_hhold", "pop_per_hhold", "bedrooms_per_room"]
cat_encoder = full_pipeline.named_transformers_["cat"]
cat_one_hot_attribs = list(cat_encoder.categories_[0])
attributes = num_attribs + extra_attribs + cat_one_hot_attribs
sorted(zip(feature_importances, attributes), reverse=True)
```

11. Evaluate final model on test dataset

```python
final_model = grid_search.best_estimator_

X_test = strat_test_set.drop("median_house_value", axis=1)
y_test = strat_test_set["median_house_value"].copy()

X_test_prepared = full_pipeline.transform(X_test)
final_predictions = final_model.predict(X_test_prepared)

final_mse = mean_squared_error(y_test, final_predictions)
final_rmse = np.sqrt(final_mse)

print ("final_predictions\n", final_predictions )
print ("final_rmse \n", final_rmse )

confidence = 0.95
squared_errors = (final_predictions - y_test) ** 2
mean = squared_errors.mean()
m = len(squared_errors)

# from scipy import stats
print("95% confidence interval: ",
      np.sqrt(stats.t.interval(confidence, m - 1,
                               loc=np.mean(squared_errors),
                               scale=stats.sem(squared_errors)))
      )
```

III. "Big Data" Machine Learning using the "Spark ML Library"

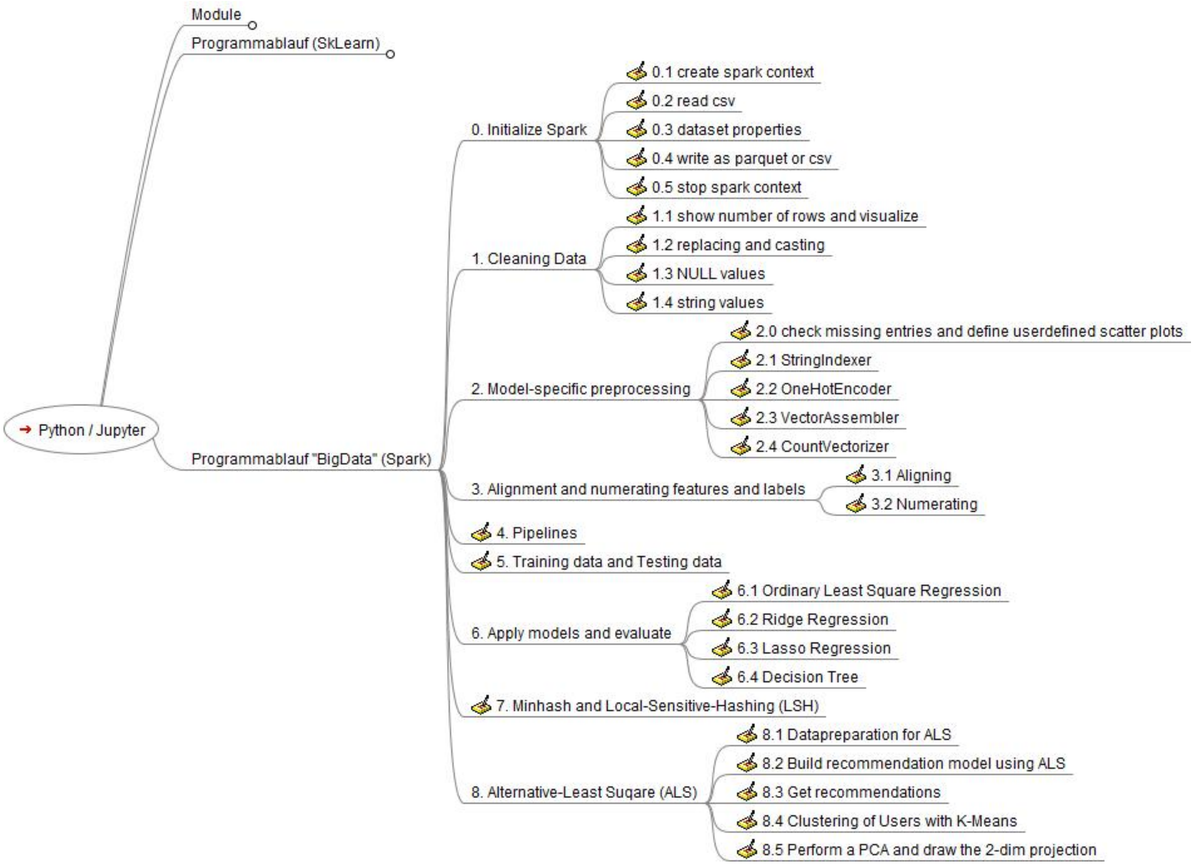
This will be an example for a "Big-Data" environment and uses the "Apache MLib" scalable machine learning library. Various tutorials, documentation, "code-fragments" and guidelines can be found in the internet **for free** (at least for your private use). The best is in my opinion the official documentation. A few more helpful sources are the following GitHub repositories:

- tirthajyoti/Spark-with-Python (MIT license)
- Apress/learn-pyspark (Freeware License)
- mahmoudparsian/pyspark-tutorial (Apache License v2.0)

Concerning the topic **"Big Data"** I want to add the following: I passed a certification as *"Data Scientist Specialized in Big Data Analytics"*. I must say: Understanding the concept of "Big-Data" and how to differentiate "standard" machine learning from a "scalable" environment is not easy. I recommend a separate training! Some steps are a bit similar to "scikit-learn" (e.g. data-cleaning, preprocessing), but the technical environment for running the code is different and also the code itself is different.

I added a **"Digression (Excurs)"** at the end of this document which covers the topics *"Big Data Visualization"*, *"K-Means-Clustering in Spark"* and *"Map-Reduce"* (one of the powerful programming models for Big Data).

Let's start with the structure, which I put into a mind map (you can download it from this repository). I aligned the structure to the SkLearn mind map above in order to learn from each of this two approaches.



There are different ways to approach the Apache Spark and Hadoop environment: you can install it on your own computer (which I found rather difficult because of lack of user-friendly and easy understandable documentation). Or you can dive into a Cloud environment, like e.g. Microsoft Azure or Amazon EWS or Google Cloud and try to get a virtual machine up and running for your purposes. Have a look at my documentation, where I shared my experiences, which I had with Microsoft Azure here.

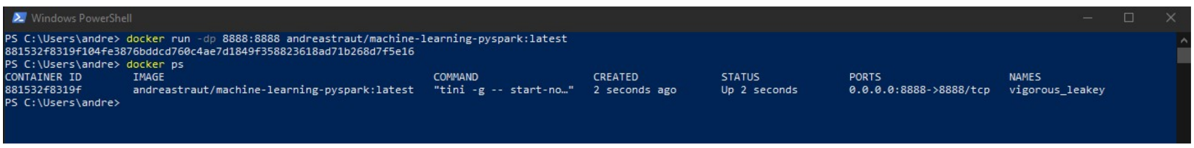
For the following explanation I decided to use Docker. What is Docker? Docker is *"an open-source project that automates the deployment of software applications inside containers by providing an additional layer of abstraction and automation of OS-level virtualization on Linux."* Learn from the Docker-Curriculum how it works. I found an container, which had Apache Spark Version 3.0.0 and Hadoop 3.2 installed and built my machine-learning code (using pyspark) on top of this container.

I shared my code and developments on Docker-Hub in the following repository here. After having installed the Docker application you will need to pull my "machine-learning-pyspark" image to your computer:

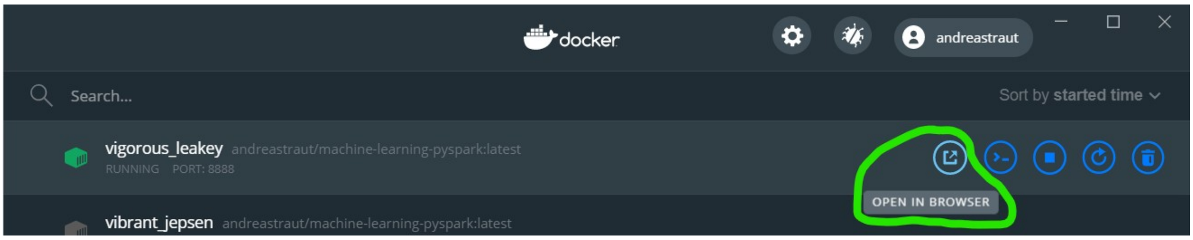
`docker pull andreastraut/machine-learning-pyspark`

Then open Windows Powershell and type the following:

`docker run -dp 8888:8888 andreastraut/machine-learning-pyspark:latest`

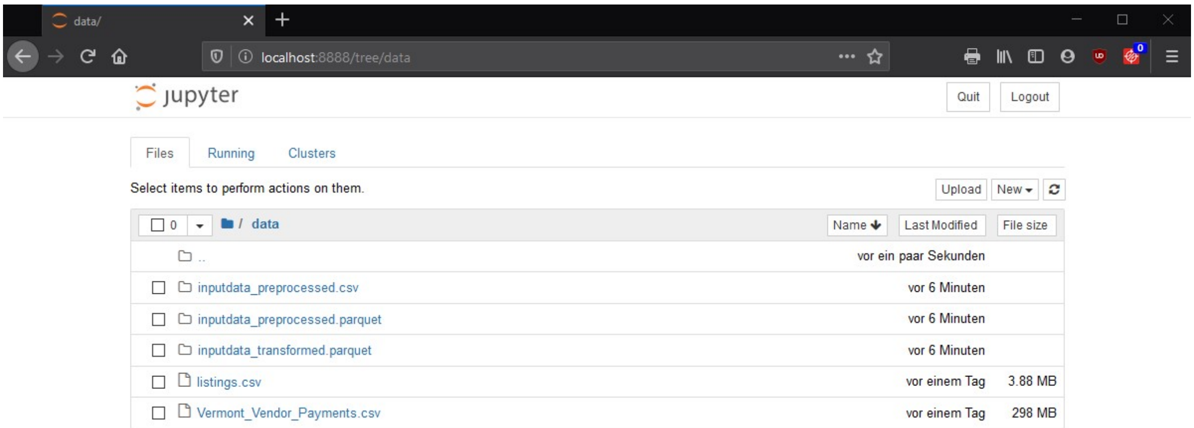


You will see in your Docker Dashborad that a container is running:

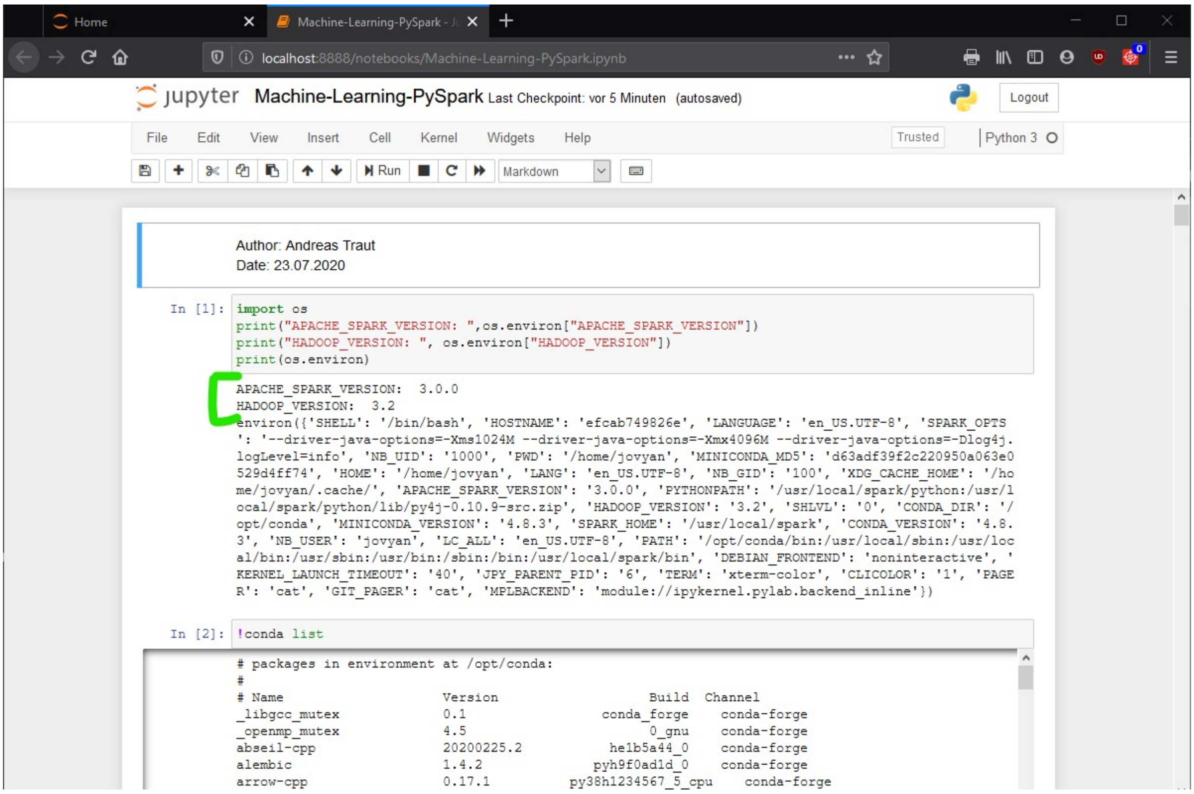


After having opened your browser (e.g. Firefox-Browser), navigate to "localhost:8888" (8888 is the port, which will be opened).

The folder "data" contains the datasets. If you would like to do further analysis or produce alternate visualizations of the Airbnb-data, you can download them from [here](). It is available below under a [Creative Commons CC0 1.0 Universal (CC0 1.0) "Public Domain Dedication"]() license. The data for the Vermont-Vendor-Payments can be downloaded from [here]() and are available under the [Open Data Commons Open Database License]().



When you open the Jupyter-Notebook, you will see, that Apache Spark Version 3.0.0 and Hadoop Version 3.2 is installed:



0. Initialize Spark

Initializing a Spark sessions works and reading a CSV file can by done with the following commands (see more documentation here and also have a look at a "Get Started Guide"):

```
In [3]: import pyspark
        from pyspark.sql import SparkSession
        from pyspark.sql import functions as F
```

0.1 Create Spark Context and Spark Session

```
In [4]: sc = pyspark.SparkContext(appName='Spark Modelling Context')
```

```
In [5]: spark = SparkSession.builder \
                .appName('Spark Modelling Session') \
                .config('spark.executor.memory','5g') \
                .config('spark.executor.cores','4') \
                .getOrCreate()
```

0.2 Read CSV

```
In [6]: import os
        datapath = os.environ['PWD']
        filename = datapath + "/data/listings.csv"
        #read in data from csv
        data = spark.read.csv(path=filename,
                              sep=',',
                              encoding='utf-8',
                              header=True,
                              inferSchema=True)
```

0.3 Dataset Properties and some Select, Group and Aggregate Methods

After then the data-cleaning and data preparation (eliminating of null values, visualization techniques) work pretty similar to the "Small data" (Sklearn) approach.

0.4 Write as Parquet or CSV

If you want to persist (=save) your intermediate you can do it as follows:

Persisting the preprocessed data

In [22]:
```python
data.select(*data.columns[:]).write.format("parquet") \
.save("data/inputdata_preprocessed.parquet", mode='overwrite')

data.select(*data.columns[:]).write.csv('data/inputdata_preprocessed.csv', mode='overwrite', header=
```

In [23]:
```python
filename = "data/inputdata_preprocessed.parquet"
data = spark.read.parquet(filename)
data.show(5)
```

```
+----+--------------------+-------+----------+--------------------+--------------------+--------+-
--------+---------------+-----+--------------+-----------------+-----------+-----------------+----
--------------------------+----------------+
|  id|                name|host_id| host_name| neighbourhood_group|       neighbourhood|latitude|l
ongitude|      room_type|price|minimum_nights|number_of_reviews|last_review|reviews_per_month|calc
ulated_host_listings_count|availability_365|
+----+--------------------+-------+----------+--------------------+--------------------+--------+-
--------+---------------+-----+--------------+-----------------+-----------+-----------------+----
--------------------------+----------------+
|3176|Fabulous Flat in ...|   3718|    Britta|              Pankow|Prenzlauer Berg S...|  52.535|
13.41758|Entire home/apt| 90.0|            62|              145| 2019-06-27|             1.11|
1.0|             140|
|3309|BerlinSpot Schöne...|   4108|      Jana|Tempelhof - Schön...|     Schöneberg-Nord|52.49885|
13.34906|   Private room| 28.0|             7|               27| 2019-05-31|             0.34|
1.0|             320|
|6883|Stylish East Side...|  16149|   Steffen|Friedrichshain-Kr...|Frankfurter Allee...|52.51171|
13.45477|Entire home/apt|125.0|             3|              133| 2020-02-16|             1.08|
1.0|               0|
|7071|BrightRoom with s...|  17391|BrightRoom|              Pankow|      Helmholtzplatz|52.54316|
13.41509|   Private room| 33.0|             1|              292| 2020-03-06|             2.27|
2.0|              45|
|9991|Geourgeous flat -...|  33852|   Philipp|              Pankow|Prenzlauer Berg S...|52.53303|
13.41605|Entire home/apt|180.0|             6|                8| 2020-01-04|             0.14|
1.0|               8|
+----+--------------------+-------+----------+--------------------+--------------------+--------+-
--------+---------------+-----+--------------+-----------------+-----------+-----------------+----
--------------------------+----------------+
only showing top 5 rows
```

0.5 Read Parquet

See jupyter notebook.

0.6 How to stop a Spark Session and Spark Context

See jupyter notebook.

1. Cleaning the data

1.1 Show number of rows and columns and do some visualizations

1.2 Replacing and Casting

1.3 Null-Values

1.4 String Values

2. Model-specific preprocessing

2.0 Check missing entries and define userdefined scatter plot

2.1 StringIndexer

I included some examples of how features can be extracted, transformed and selected in the Jupyter-Notebook (see more documentation here). Just to mention a few here: the "StringIndexer", "OneHotEncoder" and "VectorAssembler" work as follows:

```
In [25]: data.select('neighbourhood_group').distinct().count()
Out[25]: 38
```

```
In [26]: from pyspark.ml.feature import StringIndexer
         neighbourhood_indexer = StringIndexer(inputCol='neighbourhood_group', outputCol='neighbourhood_group_
         neighbourhood_indexer_model = neighbourhood_indexer.fit(data)
         data = neighbourhood_indexer_model.transform(data)
```

```
In [27]: data.groupby('neighbourhood_group').agg(F.collect_set('neighbourhood_group_index').alias('neighbourh
```

```
+--------------------+------------------------+
| neighbourhood_group|neighbourhood_group_index|
+--------------------+------------------------+
|Friedrichshain-Kr...|                   [0.0]|
|               Mitte|                   [1.0]|
|              Pankow|                   [2.0]|
|           Neukölln|                   [3.0]|
|Charlottenburg-Wilm.|                   [4.0]|
|Tempelhof - Schön...|                   [5.0]|
|        Lichtenberg|                   [6.0]|
|  Treptow - Köpenick|                   [7.0]|
|Steglitz - Zehlen...|                   [8.0]|
|       Reinickendorf|                   [9.0]|
|Marzahn - Hellers...|                  [10.0]|
|             Spandau|                  [11.0]|
| Downtown Apartments|                  [12.0]|
|Downtown Apartmen...|                  [13.0]|
|     Neue Kantstraße|                  [14.0]|
|      Alexanderplatz|                  [15.0]|
|Prenzlauer Berg Nord|                  [16.0]|
|     Alt-Lichtenberg|                  [17.0]|
|          Barstraße|                  [18.0]|
|Blankenfelde/Nied...|                  [19.0]|
|    Brunnenstr. Nord|                  [20.0]|
|Frankfurter Allee...|                  [21.0]|
|           Grunewald|                  [22.0]|
|      Kurfürstendamm|                  [23.0]|
|                Lisa|                  [24.0]|
```

2.2 OntHotEncoder

```
In [28]: from pyspark.ml.feature import OneHotEncoder
         one_hot_encoder = OneHotEncoder(
             inputCol = 'neighbourhood_group_index',
             outputCol = 'one_hot_neighbourhood_group',
             dropLast=False)
         one_hot_encoder_model = one_hot_encoder.fit(data)
         data = one_hot_encoder_model.transform(data)
```

2.3 VectorAssembler

```
In [29]: from pyspark.ml.feature import VectorAssembler
         data = data.withColumn('number_of_reviews', data['number_of_reviews'].cast('double'))
         data.select('number_of_reviews').show()
```

```
+-----------------+
|number_of_reviews|
+-----------------+
|            145.0|
|             27.0|
|            133.0|
|            292.0|
|              8.0|
|             24.0|
|             48.0|
|            262.0|
|             86.0|
|             60.0|
|             86.0|
|            307.0|
|            130.0|
|             21.0|
|              5.0|
|            188.0|
|             31.0|
|             74.0|
|            296.0|
|             39.0|
+-----------------+
only showing top 20 rows
```

2.4 CountVectorizer

3. Aligning and numerating Features and Labels

3.1 Aligning

3.2 Numerating

4. Pipelines

5. Training data and Testing data

6. Apply models and evaluate

6.1 Ordinary Least Square Regression

After having extracted, transformed and selected features you will want to apply some models, which are documented here, for example the "OLS Regression":

```
In [38]: from pyspark.ml.regression import LinearRegression
         lr = LinearRegression(featuresCol='num_features', labelCol='label',maxIter=1000, fitIntercept=True)

In [39]: lr_model = lr.fit(data_train)
         lr_model.coefficients

Out[39]: DenseVector([-0.4689])

In [40]: pred = lr_model.transform(data_test)
```

6.2 Ridge Regression

6.3 Lasso Regression

6.4 Decision Tree

7. Minhash und Local-Sensitive-Hashing (LSH)

see example: https://github.com/AndreasTraut/Deep_learning_explorations

8. Alternative-Least-Square (ALS)

8.1. Datapreparation for ALS

8.2 Build the recommendation model using alternating least squares (ALS)

8.3 Get recommendations

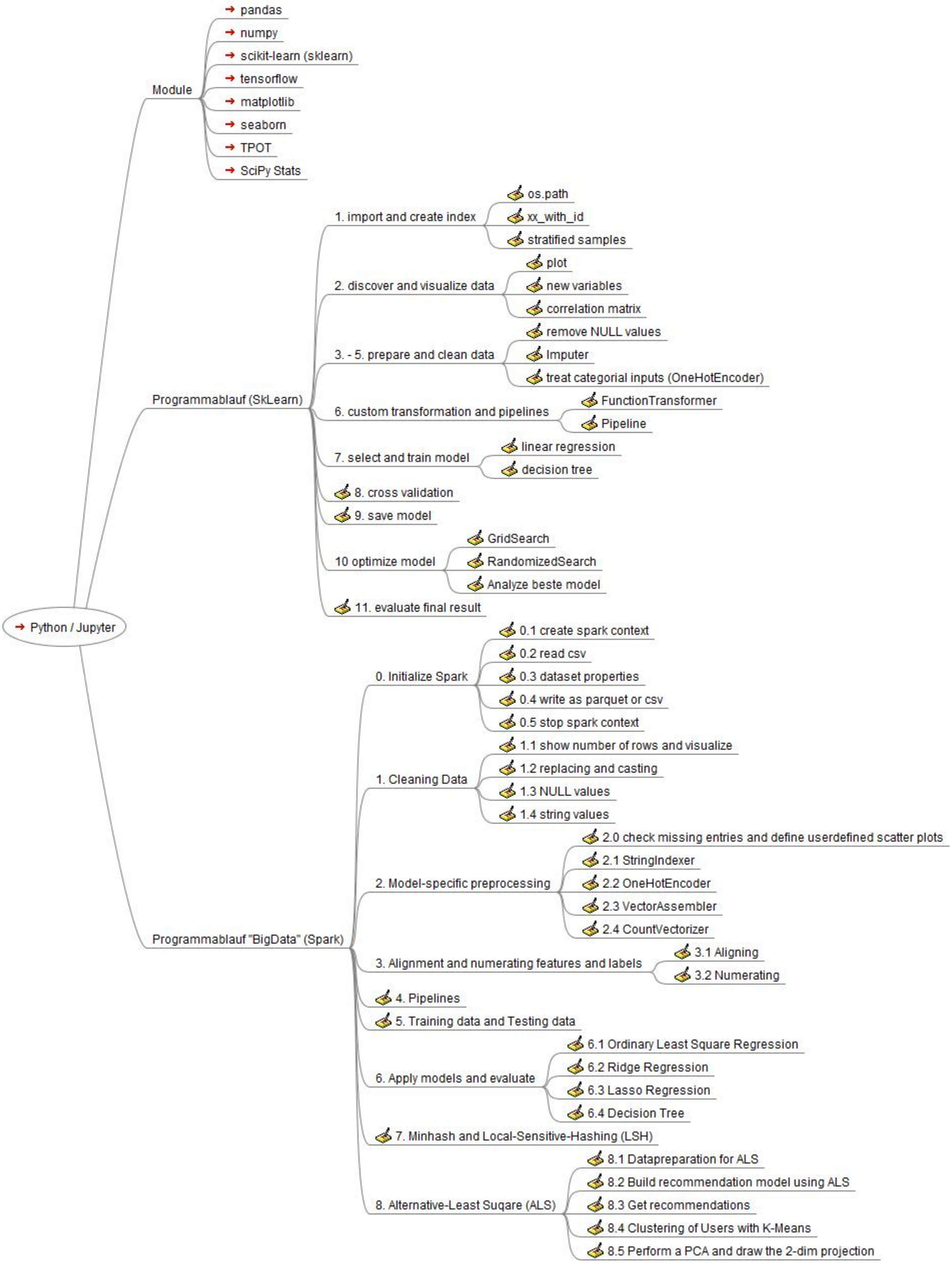
8.4 Clustering of Users with K-Means

see example: https://hub.docker.com/repository/docker/andreastraut/machine-learning-pyspark

8.5 Perform a PCA and draw the 2-dim projection

IV. Summary Mind-Map

To summarize the whole coding structure have a look at the following and also the provided mind-maps. My mind map below may help you to structure your code:

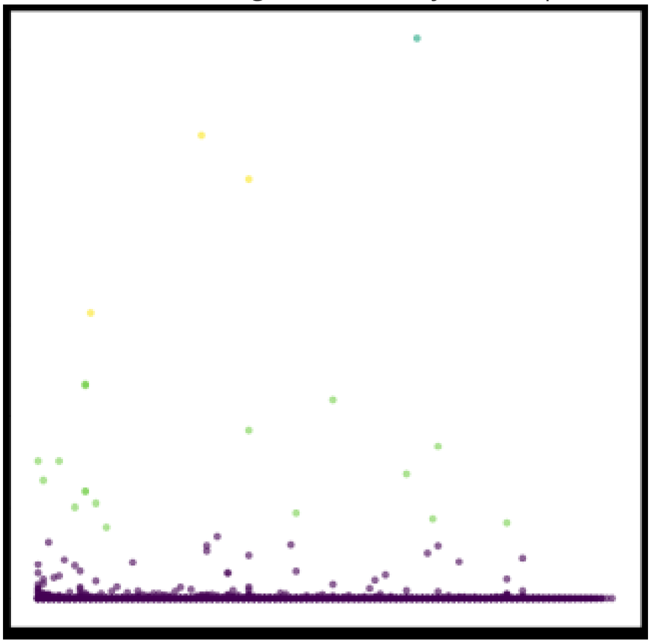


V. Digression (Excurs) to Big Data Visualization and K-Means Clustering Algorithm and Map-Reduce

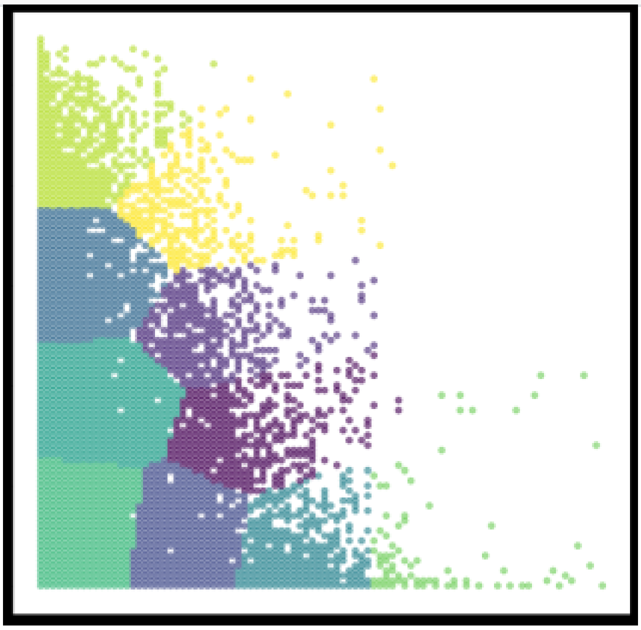
Digression (Excurs) to Big Data Visualization and K-Means Clustering Algorithm and Map-Reduce

(i) Big Data Visualization: You will see a Jupyter-Notebook (which contains the Machine-Learning Code) and a folder named "data" (which contains the raw-data and preprocessed data). As you can see: I also worked on a 298 MB big csv-file ("Vermont_Vendor_Payments.csv"), which I couldn't open in Excel, because of the huge size. This file contains a list of all state of Vermont payments to vendors (Open Data Commons License) and has more than 1.6 million lines (exactly 1'648'466 lines). I already mentioned in my repository "Visualization-of-Data-with-Python", that the **visualization of big datasets** can be difficult when using "standard" office tools, like Excel. If you are not able to open such csv-files in Excel you have to find other solutions. One is to use PySpark which I will show you here. Another solution would have been to use the Excel built-in connection, PowerQuery or something similar, maybe Access or whatever, which is not the topic here, because we also want to be able to apply machine-learning algorithms from the Spark Machine Learning Library. And there are more benefits of using PySpark instead of Excel: it can handle distributed processing, it's a lot faster, you can use pipelines, it can read many file systems (not only csv), it can process real-time data.

(ii) K-Means Clustering Algorithm: Additionally I worked on this dataset to show how the K-Means Clustering Algorithm can be applied by using the Spark Marchine-Learning Libary (see more documentation here). I will show how the "Vermont Vendor Payments" dataset can be clustered. In the images below every color represents a differents cluster:



Digression (Excurs) to Big Data Visualization and K-Means Clustering Algorithm and Map-Reduce



(iii) Map-Reduce: This is a programming model for generating big data sets with parallel distributed algorithm on a cluster. Map-Reduce is very important for Big Data and therefore I added some Jupyter-Notebooks to better understand how it works. Learn the basis of the *Map-Reduce* programming model from here and then have a look into my jupyter notebook for details. I used the very popular "Word Count" example in order to explain Map-Reduce in detail.

In another application of Map-Reduce I found the very popular **term frequency–inverse document frequency** (short **TF-idf**) very interesting (see Wikipedia). This is a numerical statistic, which is often used in text-based recommender systems and for information retrieval. In my example I used the texts of "Moby Dick" and "Tom Sawyer". The result are two lists of most important words for each of these documents. This is what the TF-idf found:
Moby Dick: WHALE, AHAB, WHALES, SPERM, STUBB, QUEEQUEG, STARBUCK, AYE
Tom Sawyer: HUCK, TOMS, BECKY, SID, INJUN, POLLY, POTTER, THATCHER
Applications for using TF-idf are in the information retrieval or to classify documents.

Have a look into my notebook here to learn more about Big Data Visualization, K-Means Clustering Algorithm, Map-Reduce and TF-idf.

Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International License