Zadanie 2 - Analýza dát a regresory
I-SUNS: Strojové učenie a neurónové siete

Pavol Ďuďák
xdudakp@stuba.sk

18. novembra 2021

1 Úvod

Po použití neurónových sietí na jednoduché klasifikačné úlohy si v tomto zadaní predstavíme alternatívu, regresory. Regresory sú veľa krát šikovnejšie pri jednoduhších úlohách pri ktorých používame štruktúrované (napr. csv) dáta. Regresory nemajú toľko konfiguračných parametrov ako neurónové siete a ich použitie je všeobecne jednoduchšie.

Cieľom zadania je oboznámiť sa s použitím rôznych regresorov a ich využitím pri úlohe predpovedi hlasitosti skladieb na základe ich rôznych atribútov. Zároveň nad poskytnutými dátami bola vypracovaná exploratívna analýza (EDA).

2 Spracovanie a analýza dát

2.1 Spracovanie dát

Pred analýzou je nutné dáta spracovať. Spracovanie zahŕňa odstránenie nepotrebných stĺpcov, kódovanie nečíselných hodnôt, odstránenie null hodnôt a odstránenie hraničných hodnôt.

Dáta si načítame z poskytnutého .csv

```
df_raw: DataFrame = pd.read_csv('sample_data/spotify_train.csv')
```

Listing 1: Načítanie trénovaích dát

následne upravíme dátum vydania iba na rok a extrahujeme všetky žánre a roky vydania ktoré budú použité pre zakódovanie vo forme masky.

```
df_raw['release_date'] = df_raw['release_date'].apply(lambda d: d[0:4])
genres: Set[str] = extract_unique_genres(df_raw)
release_dates: Set[str] = collect_release_data(df_raw)
df_raw['artist_genres'] = df_raw['artist_genres'].apply(str_to_list)
df_raw['index'] = df_raw.index
```

Listing 2: Úprava rokov a extrakcia žánrov a rokov vydania

Z extrahovaných hodnôt žánrov a rokov vydania vytvoríme masku.

```
# vytvorim bool masku top 50 udajov
genres_mask = create_bool_mask(df_raw, genres, 'artist_genres')
genres_mask['index'] = genres_mask.index
genres_index = genres_mask.index

release_mask = create_bool_mask_date(df_raw, release_dates, 'release_date')
release_mask['index'] = release_mask.index
releases_index = release_mask.index
```

Listing 3: Vytvorenie masky

Aby sme mali v datasete iba číselné hodnoty tak všetky boolean hodnoty zmeníme na 0 alebo 1.

```
genres_mask.replace({False: 0, True: 1}, inplace=True)
release_mask.replace({False: 0, True: 1}, inplace=True)
```

Listing 4: Kódovanie bool hodnôt na čísla

Aby náš dataset nemal príliš veľa stĺpcov (keďže nečíselné údaje maskujeme do stĺpcov) tak vyberieme top 50 žánrov a 50 rokov vydania a tieto údaje zamaskujeme do nášho datasetu.

```
top_genres = genres_mask.sum(axis=0).drop(labels=['index']).nlargest(50)
top_years = release_mask.sum(axis=0).drop(labels=['index']).nlargest(50)
# maskovanie
for genre in top_genres.keys():
  df_raw[genre] = df_raw['artist_genres'].apply(lambda x: 1 if genre in x else 0)
for year in top_years.keys():
  df_raw[year] = df_raw['release_date'].apply(lambda x: 1 if x == year else 0)
```

Listing 5: Maskovanie

Pre ďalšie spracovanie sú uvedené stĺpce nerelevantné tak ich môžeme vyhodiť.

```
df_raw.drop(['url', 'playlist_id', 'playlist_description', 'playlist_name', '
    playlist_url', 'query'], axis=1, inplace=True)
```

Listing 6: Odstánenie nerelevantných stĺpcov

Dôležitým krokom pri spracovaní dát je odstránenie duplikátov. Pri poskytnutom datasete sme nemali žiaden jednoznačný ukazovateľ ktorý by hovoril že ide o duplikát takže sme museli vymyslieť stratégiu ktorou budeme odstraňovať duplikáty.

1KRpJeeB2JEG6AW6Uri3QX	1dfeR4HaWDbWqFHLkxsg1d	Queen	Bohemian Rhapsody	0	1975-11-21	354346	FALSE
6fPm2XMi51I5GCEp19HscL	1dfeR4HaWDbWqFHLkxsg1d	Queen	Bohemian Rhapsody	0	1975-11-21	354840	FALSE
3z8h0TU7ReDPLibEnYhWZb	1dfeR4HaWDbWqFHLkxsg1d	Queen	Bohemian Rhapsody	72	2018-10-19	354947	FALSE
1pwE0JB6MTtTWx8fzqriAj	1dfeR4HaWDbWqFHLkxsg1d	Queen	Bohemian Rhapsody - 2011 Remaster	0	1981-10-26	355466	FALSE
64uO28GXj2aA7DYcvnEoCl	1dfeR4HaWDbWqFHLkxsg1d	Queen	Bohemian Rhapsody - Live	40	2007-10-09	328933	FALSE
7xHATAMD7ezTZGYsNAMr5R	1dfeR4HaWDbWqFHLkxsg1d	Queen	Bohemian Rhapsody - Live Aid	59	2018-10-19	147840	FALSE
0U49xOQCWh6zDWcu5r8Why	1dfeR4HaWDbWqFHLkxsg1d	Queen	Bohemian Rhapsody - Live At The Montreal Forum / November 1981	42	2007-10-29	328933	FALSE
5YJIdKE76GacbcTxuoOf93	1dfeR4HaWDbWqFHLkxsg1d	Queen	Bohemian Rhapsody - Live At Wembley Stadium / July 1986	0	1992-05-26	350573	FALSE
2XbcujvemK0hGh0Ob4HAXQ	1dfeR4HaWDbWqFHLkxsg1d	Queen	Bohemian Rhapsody - Live At Wembley Stadium / July 1986	51	1992-05-26	350573	FALSE
2vwqOagUdKs4iPG4awFt7Y	1dfeR4HaWDbWqFHLkxsg1d	Queen	Bohemian Rhapsody - Remastered 2011	42	1975-11-21	354320	FALSE
2GAlycsMaDVtMtdvxzR2xI	1dfeR4HaWDbWqFHLkxsg1d	Queen	Bohemian Rhapsody - Remastered 2011	52	2011-01-01	354320	FALSE
1AhDOtGVPSOmsWgNW0BEY	1dfeR4HaWDbWqFHLkxsg1d	Queen	Bohemian Rhapsody - Remastered 2011	2	1975-11-21	354320	FALSE
5w5XPGYKMycWUQEZb0UFEH	1dfeR4HaWDbWqFHLkxsg1d	Queen	Bohemian Rhapsody - Remastered 2011	0	2011-01-01	355466	FALSE
2OBofMJx94NryV2SK8g8Zf	1dfeR4HaWDbWqFHLkxsg1d	Queen	Bohemian Rhapsody - Remastered 2011	51	1981-10-26	355466	FALSE
7tFiyTwD0nx5a1eklYtX2J	1dfeR4HaWDbWqFHLkxsg1d	Queen	Bohemian Rhapsody - Remastered 2011	74	1975-11-21	354320	FALSE
4u7EnebtmKWzUH433cf5Qv	1dfeR4HaWDbWqFHLkxsg1d	Queen	Bohemian Rhapsody - Remastered 2011	82	1975-11-21	354320	FALSE

Obr. 1: Ukážka duplikátov

V datasete nastávali často situácie uvedené na obrázku 1. V tomto prípade môžeme vidieť nie len že jedna skladba má viacero rôznych verzií no aj to že skladba s rovnakým názvom sa vyskytuje v datasete viac krát. Tento problém sme sa rozhodli riešiť vybraním najpopulárnejšej skladby

zo skupiny skladieb z rovnakým názvom. Túto operáciu vykonávame pre každého interpreta a jeho skladby. Algoritmus zoskupuje skladby podľa interpreta, potom všetky skladby interpreta zoskupí podľa názvu (dostaneme množinu skladieb interpreta z rovnakým názvom) a z týchto vyberie len tú najpopulárnejšiu podľa stĺpca *popularity*.

```
# vycistim duplikaty songov
artist_groups = create_groups(df_raw, 'artist')
filtered_artists: List[DataFrame] = []
for artist in artist_groups:
    grouped_by_song: List[DataFrame] = create_groups(artist, 'name')
    filtered_artists.append(pd.concat(filter_only_relevant_song(grouped_by_song))) #
    songy ktore su uz odfiltrovane zlucim do jedneho DF
df_filtered = pd.concat(filtered_artists)
```

Listing 7: Čistenie duplikátov

V tento moment máme dataset skladieb ktoré neobsahujú duplikáty. Poslednými krokmi sú odstránenie null hodnôt a outliers (outliers sa odstraňujú až neskôr).

```
import numpy as np
from scipy import stats
df_filtered = df_filtered.dropna()
X[(np.abs(stats.zscore(X)) < 3).all(axis=1)] # tento krok je realne v kode az neskor
```

Listing 8: Mazanie null hodnôt a outliers

2.2 Exploratívna analýza dát

Po načítaní a úprave dát môžeme robiť analýzu. V tejto analýze sa po pohľade na dáta spýtame niekoľko otázok na ktoré si v zápätí grafmi odpovieme.

1. Ktoré žánre majú najväčšie zastúpenie?
2. Ktorý z interpretov sú najsledovanejší?
3. Ako energicky pôsobila hudba počas rokov?
4. Ako hlasito pôsobila hudba počas rokov?

Manipuláciou vyčistených dát dokážeme na tieho otázky jednoducho a graficky odpovedať.

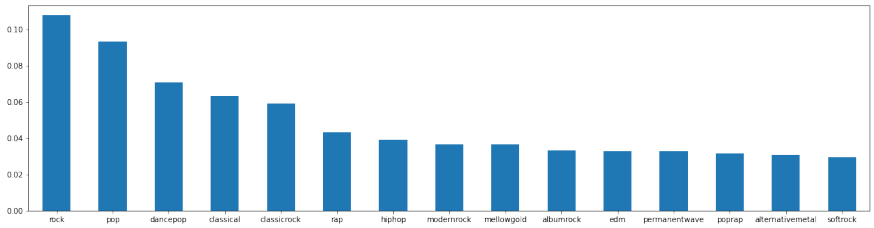
Ktoré žánre majú najväčšie zastúpenie?

Pre každý žáner vypočítame pomer jeho zastúpenia v datasete a vyberieme prvých 15 záznamov ktoré si zobrazíme do stĺpcového diagramu.

```
# hladame najviac zastupene
genres = top_genres.keys()
genre_percent = { genre: len(df_filtered.loc[df_filtered[genre]==True]) / len(
    df_filtered) for genre in list(genres)}
s_genres: Series = Series(genre_percent)
s_genres.sort_values(ascending=False, inplace=True)

s_genres.head(15).plot.bar(rot=0, figsize=(20,5))
```

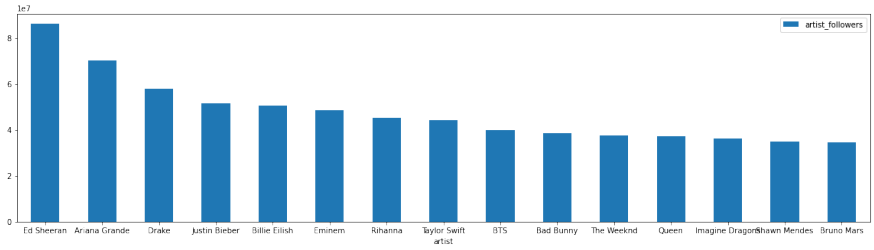
Listing 9: Hľadanie top žánrov



Obr. 2: Zastúpenie žánrov v datasete

Ktorý z interpretov sú najsledovanejší

Pre získanie interpretov ktorý sú najsledovanejší potrebujeme spraviť jednoduchú transformáciu dát. Zoskupíme si dáta podľa interpreta, zoberieme jeho náhodnú prvú skladbu a tieto skladby zoradíme podľa počtu sledovateľov. Tento počet sa viaže na interpreta takže nám je jedno ktorú jeho skladbu vyberieme. Následne tieto dáta zobrazíme.



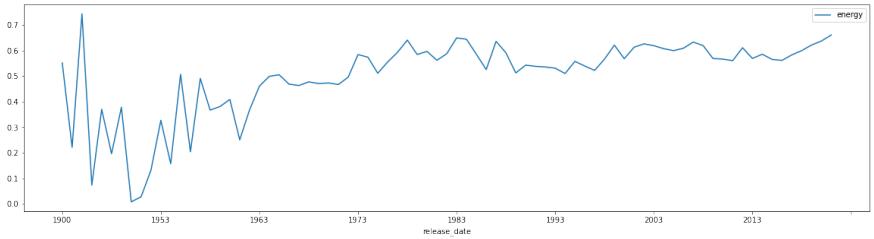
Obr. 3: Najsledovanejší interpreti

```
artist_leaderboard = df_filtered[['name', 'artist', 'artist_followers']].groupby(['
    artist']).first().reset_index().sort_values(by='artist_followers', ascending=False)
    .head(15).plot.bar(x='artist', y='artist_followers', rot=0, figsize=(20,5))
```

Listing 10: Hľadanie najsledovanejších interprétov

Ako energicky pôsobila hudba počas rokov?

Pre zobrazenie tohto atribútu jediné čo potrebujeme spraviť je usporiadať dáta podľa roku vydania a priemery atribútu *energy* zobraziť na grafe.



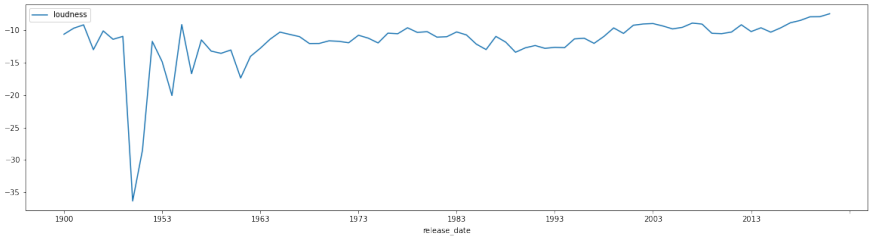
Obr. 4: Priebeh 'energičnosti' skladieb

```
df_filtered[['release_date', 'energy']].sort_values(by='energy', ascending=False).
    groupby('release_date').mean().reset_index().plot.line(x='release_date', y='energy'
    , rot=0, figsize=(20,5))
```

Listing 11: Hľadanie energičnosti

Ako hlasito pôsobila hudba počas rokov?

Rovnakým spôsobom vieme zobraziť aj priebeh pocitu hlučnosti skladieb počas rokov.



Obr. 5: Priebeh "hlučnosti"skladieb

```
df_filtered[['release_date', 'loudness']].sort_values(by='loudness', ascending=False).
    groupby('release_date').mean().reset_index().plot.line(x='release_date', y='
    loudness', rot=0, figsize=(20,5))
```

Listing 12: Hľadanie hlučnosti

3 SVM regresor

Pre prvotný pokus bol použitý regresor SVR z RBF kernelom. Tento kernel je ideálny pre nelineárne dáta. Pokiaľ by pri tom istom regresore bol použitý lineárny kernel tak trénovanie nie že by len dlho trvalo ale produkovalo by aj nepresné výsledky. V tom to prípade sú hodnoty hyperparametrov:

- kernel = rbf
- C = 1
- gamma = 1 / pocet stlpcov

```
regressor = SVR(kernel = 'rbf')
regressor.fit(X, y.to_numpy())
```

Listing 13: SVR regresor

3.1 Hyperparametre

Hyperparametre sú parametre ktorý ovplyvňujú trénovanie a pomocou ktorých môžeme ladiť svoj model.

3.1.0.1 kernel Kernel je možné si predstaviť ako funkciu ktorá bude prikovaná na dáta aby sa nelineárne dáta dali lineárne rozdeliť. Pri lineárnom kerneli sú dáta rozdelené priamkou, nič zvláštne. Ak ale priamka nestačí tak ako kernel sa môže zvoliť polynomiálna funkcia ktorá dokáže dáta rozdeliť krivkou. Pokiaľ ani toto nie je dostačujúce tak sa používa tzv. kernelový trik. V tomto prípade sa ako kernel používa 'rbf' - radiálna bázová funkcia. V prípade kernelového triku ide o transformáciu priestoru tak aby sa dali dáta rozdeliť lineárne hyperrovinami vo viac rozmernom priestore.

3.1.0.2 C Hodnota hyperparametra určuje akýsi rozostup medzi hyperrovinou a dátovými bodmi ktoré sú klasifikované do určenej triedy. čím nižši je C tým väčší rozostup má hyperrovina.

3.1.1 gamma

Tento hyperparameter určuje šírku funkcie kernelu. Inak povedané určuje aký dosah bude mať vzorka pri klasifikácií. Čím nižšia hodnota tým väčší dosah budú mať vzorky.

Pri spomínanej konfigurácie sme dostali výsledky:

```
Results of SVR:
MSE: 9.077949170492122
R-Squared: 0.7930420157292597
```

Vidíme že úspešnosť je 79% (podĺa metriky R^2). Ďalšia metrika pre nás zaujímavá je MSE čo je priemer reziduálov. Reziduál je rozdiel medzi predpovedanou a reálnou hodnotou. Z toho nám vyplýva že čím menšie je MSE tým lepší bol náš regresor. Interval hodnôt MSE záleží od problému ktorý regresor rieši keďže pracuje s predpovedanými hodnotami a reálnymi výstupmi množiny.

3.2 Grid search

Skúšanie kombinácií týchto hyperparametrov je časovo náročné a monotónne. Pre zjednodušenie práce bol použitý algoritmus GridSearch ktorému zadáme množinu hyperparametrov ktoré on skúša postupne metódou *každy s každým*. Tento proces je dlhotrvajúci no na základe jeho výstupu vieme dostať najlepšiu konfiguráciu hyperparametrov pre náš klasifikátor.

```
tuned_parameters = [
    {"kernel": ["rbf"], "gamma": [1e-3, 1e-4], "C": [1, 10, 100, 1000]},
]

scores = ["r2"]

print("# Tuning hyper-parameters for %s" % score)
print()

clf = GridSearchCV(SVR(), param_grid=tuned_parameters, cv=2)
clf.fit(X, y.to_numpy())

print("Best parameters set found on development set:")
```

```
print()
print(clf.best_params_)
print()
print("Grid scores on development set:")
print()
means = clf.cv_results_["mean_test_score"]
stds = clf.cv_results_["std_test_score"]
for mean, std, params in zip(means, stds, clf.cv_results_["params"]):
    print("%0.3f (+/-%0.03f) for %r" % (mean, std * 2, params))
print()

print("Detailed classification report:")
print()
print("The model is trained on the full development set.")
print("The scores are computed on the full evaluation set.")
print()
y_true, y_pred = yt, clf.predict(Xt)
```

Listing 14: GriSearch algoritmus

Výsledok algoritmu GridSearch spolu s cross validation bol následovný:

```
# Tuning hyper-parameters for r2

Best parameters set found on development set:

{'C': 1000, 'gamma': 0.001, 'kernel': 'rbf'}

Grid scores on development set:

0.626 (+/-0.077) for {'C': 1, 'gamma': 0.001, 'kernel': 'rbf'}
0.085 (+/-0.109) for {'C': 1, 'gamma': 0.0001, 'kernel': 'rbf'}
0.777 (+/-0.002) for {'C': 10, 'gamma': 0.001, 'kernel': 'rbf'}
0.627 (+/-0.077) for {'C': 10, 'gamma': 0.0001, 'kernel': 'rbf'}
0.795 (+/-0.017) for {'C': 100, 'gamma': 0.001, 'kernel': 'rbf'}
0.776 (+/-0.002) for {'C': 100, 'gamma': 0.0001, 'kernel': 'rbf'}
0.812 (+/-0.019) for {'C': 1000, 'gamma': 0.001, 'kernel': 'rbf'}
0.792 (+/-0.017) for {'C': 1000, 'gamma': 0.0001, 'kernel': 'rbf'}

Detailed classification report:

The model is trained on the full development set.
The scores are computed on the full evaluation set.
```

Vidíme že sme dostali najlepšie výsledky pri parametroch C=1000, gamma=0.001 a kernel: rbf pri ktorých bola úspešnosť 81% (podľa metriky R^2) Do algoritmu môžeme dosadiť aj viacero možností hyperparametrov. Čím viac hyperparametrov zadáme tým dlhšie bude algoritmus bežať no dokážeme teoreticky dostať lepšie výsledky.

3.3 Metóda bagging

Pri metóde bagging vytvárame viacero regresorov ktoré dostanú náhodné podmnožiny trénovacích dát ktorých výsledky sú priemerované do konečného výsledku. Metóda sa dá jednoducho popísať ako kombinácia viacerých slabých regresovov nám dá presnejší model regresora a tým vytvára jeden silný regresor.

```
bagging_reg = BaggingRegressor(base_estimator=regressor, n_estimators=27, random_state
    =255)
bagging_reg.fit(X, y.to_numpy())
y_pred_bagging = bagging_reg.predict(Xt)
mse = metrics.mean_squared_error(yt, y_pred_bagging)
r2 = metrics.r2_score(yt,y_pred_bagging)
print("Results of sklearn.metrics:")
print("MSE:", mse)
print("R-Squared:", r2)
```

Listing 15: Metóda bagging

Podľa výsledkou bohužial vidíme že metóda bagging nebola úspešejšia ako SVC regresor s vyladenými parametrami podľa Grid Seach (podľa metriky R^2).

```
Results of bagging:
MSE: 9.065703067088416
R-Squared: 0.793321200909525
```

3.4 Metóda boosting

Podobne ako pri metóde bagging sa používa viacero slabých regresorov ktoré sú ale vyhodnocované sekvenciálne po sebe a každý ďalší dostane upravené parametre z predchádajúceho. Táto metóda sa dá popísať ako vylepšovanie slabých regresorov na základe prechádzajúcich pre vytvorenie jedného silného regresora.

```
boosting_reg = GradientBoostingRegressor(random_state=255)
boosting_reg.fit(X, y.to_numpy())
y_pred_boosting = boosting_reg.predict(Xt)
```

Listing 16: Metóda boosting

V tomto prípade bola metóda boosting úspešnejšia ako metóda bagging (podľa metriky R^2).

```
Results of boosting:
MSE: 5.122653856629292
R-Squared: 0.8832143586206862
```

Celkovô môžeme zhodnotiť že najlepší regresor s úspešnosťou 88% (podľa metriky R^2) bol pri boosting regresor a to presnejšie Gradient boosting regressor.

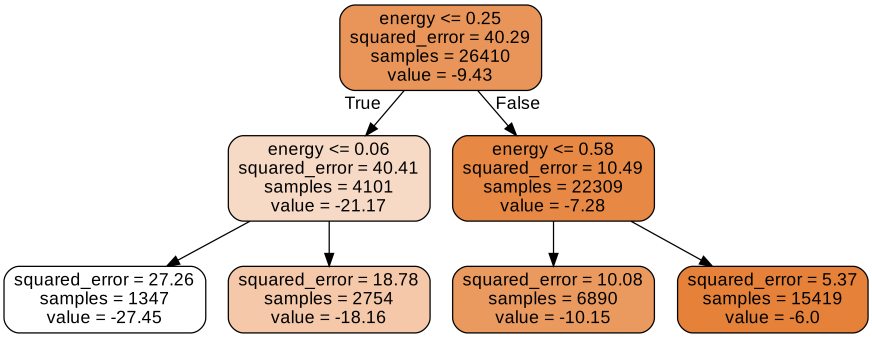
3.5 Random forrest

Posledný regresor ktorý bol použitý je RandomForrest regresor ktorý si ľahko dokážeme aj vizualizovať.

```
rf = RandomForestRegressor(max_depth=2, random_state=0)
rf.fit(X, y)
```

Listing 17: Random forrest regresor

Každe 42 je odpoveď na základnú otázku života, vesmíru a všetkého tak si zoberieme 42. estimátor (strom nášho náhodného lesa) a vizualizujeme si ho:



Obr. 6: Strom náhodného lesa

Na strome vidíme zároveň že stĺpec energy vo vstupnom dataseťe má veľkú váhu pri vyhodnocovaní predikovaného výsledku.

Literatúra

[1] https://scikit-learn.org/stable/auto_examples/model_selection/plot_grid_search_digits.html

[2] https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.BaggingRegressor.html

[3] https://towardsdatascience.com/how-to-visualize-a-decision-tree-from-a-random-forest-in-python-using-scikit-learn-38ad2d75f21c

[4] https://towardsdatascience.com/ensemble-learning-bagging-boosting-3098079e5422

[5] https://www.analyticsvidhya.com/blog/2020/03/support-vector-regression-tutorial-for-machine-learning/

[6] https://towardsdatascience.com/train-test-split-and-cross-validation-in-python-80b61beca4b6

[7] https://www.datatechnotes.com/2019/10/accuracy-check-in-python-mae-mse-rmse-r.html

[8] https://stackoverflow.com/questions/23199796/detect-and-exclude-outliers-in-pandas-data-frame