Machine Learning Basics:

Customer Retention Project

Elton Grivith D Souza 16 September 2021

Machine learning tasks can be gathered into the four following categories:

	Supervised learning	Unsupervised learning
Discrete	Classification or categorization	Clustering
Continuous	Regression	Dimensionality reduction

This article mainly focuses on regression. Specifically, this article describes the basis of this task and Illustrates the main concepts onto the Customer Retention dataset.

Note : The complete code is provided on GitHub. This document only gives an overview of the procedure

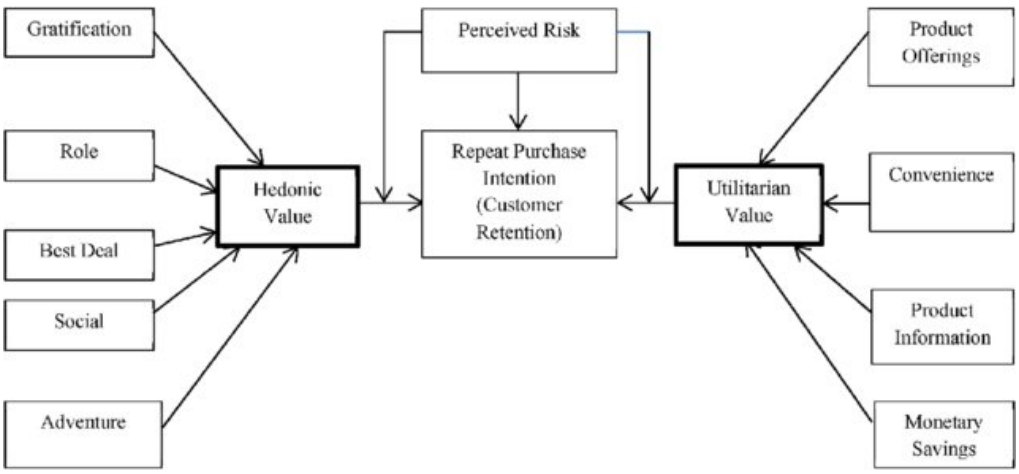
The structure of this article is as following:

- Problem Definition
- Data Analysis
- Pre-processing
- Building Machine Learning (ML) models
- Hyper - Parameter Tuning

Problem Definition

Customer satisfaction has emerged as one of the most important factors that guarantee the success of online store; it has been posited as a key stimulant of purchase, repurchase intentions and customer loyalty. A comprehensive review of the literature, theories and models have been carried out to propose the models for customer activation and customer retention.

Five major factors that contributed to the success of an e-commerce store have been identified as: service quality, system quality, information quality, trust and net benefit. The research furthermore investigated the factors that influence the online customers repeat purchase intention. The combination of both utilitarian value and hedonistic values are needed to affect the repeat purchase intention (loyalty) positively. The data is collected from the Indian online shoppers. Results indicate the e-retail success factors, which are very much critical for customer satisfaction.



Data Analysis

Step 1: Importing the libraries

We first import most of the necessary libraries.

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from scipy import stats
from sklearn.preprocessing import LabelEncoder
from sklearn.preprocessing import PowerTransformer
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import cross_val_score,cross_validate
from sklearn.tree import DecisionTreeRegressor
from sklearn.neighbors import KNeighborsRegressor
from sklearn.ensemble import
RandomForestRegressor,AdaBoostRegressor,GradientBoostingRegressor
from sklearn.linear_model import ElasticNet
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import
mean_absolute_error,mean_squared_error,r2_score
from sklearn.model_selection import train_test_split
from sklearn.svm import SVR
from lightgbm import LGBMRegressor
from xgboost import XGBRegressor
from sklearn.metrics import SCORERS
from sklearn.model_selection import GridSearchCV,RandomizedSearchCV
import warnings

%matplotlib inline
warnings.filterwarnings('ignore')
```

Step 2: Reading the Dataset

Our dataset is in the csv format so we use the read_csv function to read our dataset.

You can then use `df.head()` or `df.tail()` to view some of the data in the dataframe.

```
# Reading Dataset
df=pd.read_csv('customer_retention_dataset.csv')
```

Step 3: Dataset Information

This step allows us to view basic information about our dataframe. We can then plan our approach to exploring, processing and modelling our data based on this understanding.

```
# (Rows, Columns in df)
> df.shape
# Memory size of df
> df.size
# Base information of df
> df.info()
# Statistical description of numerical columns in df
> df.describe()
```

Step 4: Exploring the data

We now understand the dataframe well enough to explore it. This step involves univariate and multivariate analysis that can then be used to pre-process the data to give us better models. A few suggestions have been provided below. Feel free to further explore and visualise the data.

```
# Unique values in `column_name`
> df.`column_name`.unique()

# value count in `column_name`
> df.`column_name`.value_counts
```

Machine learning models require all input and output variables to be numeric. This means that if your data contains categorical data, you must encode it to numbers before you can fit and evaluate a model. It is also easier to visualise numeric data. The snippet given below can be used to encode your labels. We will be encoding each label independently.

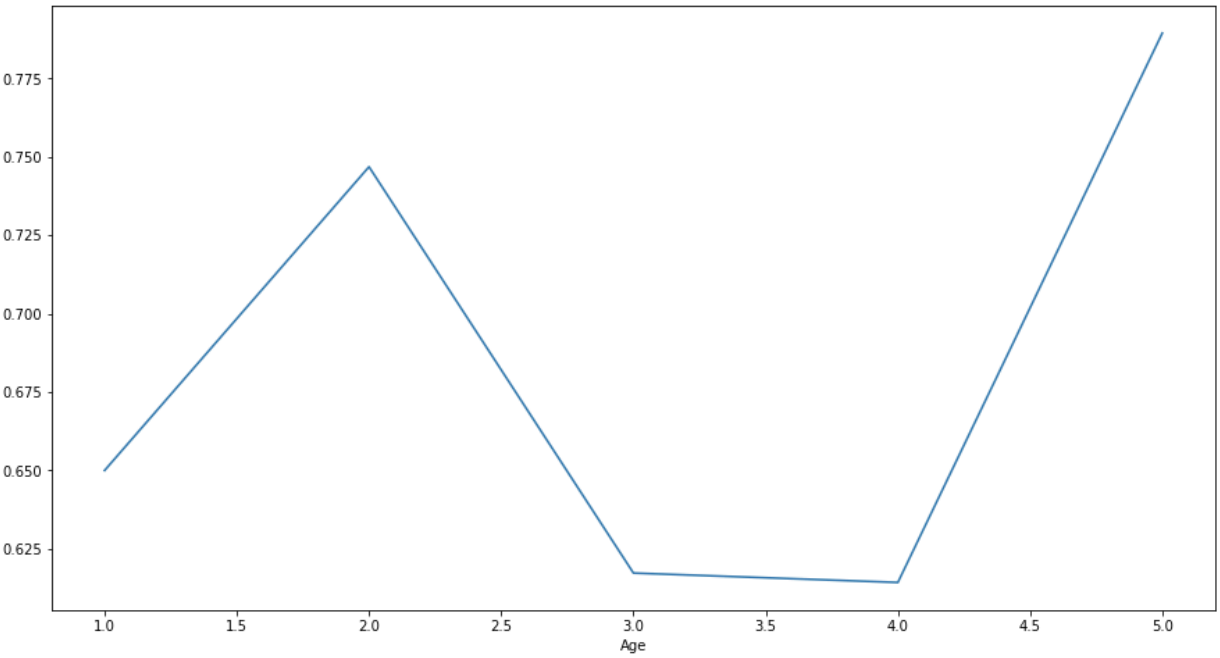
```
> lab_enc=LabelEncoder()

# Feature Encoding
> data=lab_enc.fit_transform(df['column_name'])
  df['column_name']=data
```

The snippet above converts the input column into `*int64*` *d*type using the Label Encoder instance.

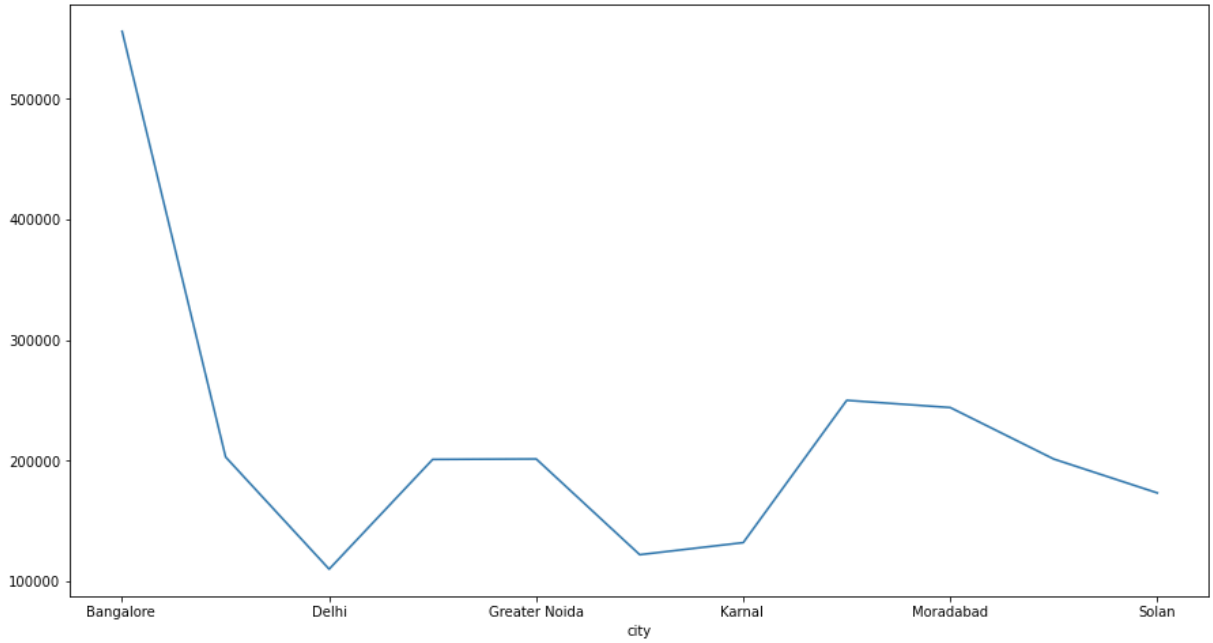
Snippets given below can be used to obtain different plots. We can use this to observe different behaviour in our data. Examples of such observations have been provided below.

```
# Relation of Gender vs Age
D=df.groupby('Age').mean()
plt.figure(figsize=(15,8))
D['Gender'].plot()
```



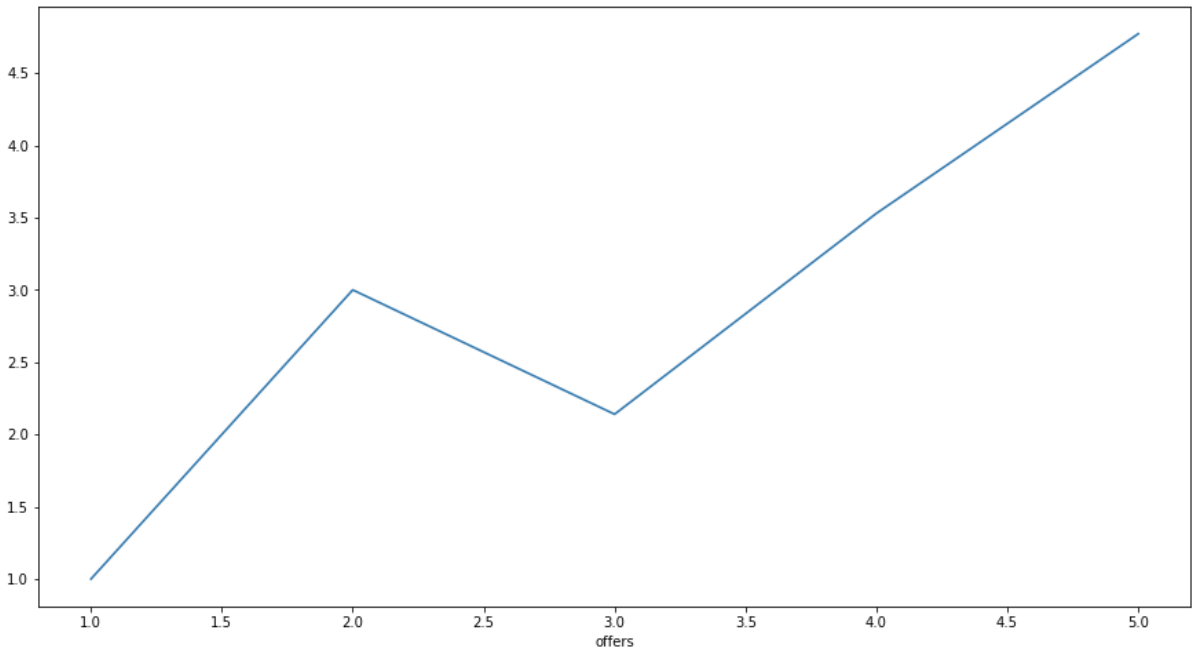
The graph above shows the distribution of the gender of the survey participants in regard to their age. We can see that our data is biased towards certain genders in certain ages. This means that our final model might be biased toward the interests of people in certain gender based age groups.

```
# Relation of City vs pin
D=df.groupby('city').mean()
plt.figure(figsize=(15,8))
D['pin'].plot()
```



From the figure above, we can see that participant records are imbalanced. There are too many records from certain cities and too less from others. This means the final model will perform better with participants from a certain given region more than others.

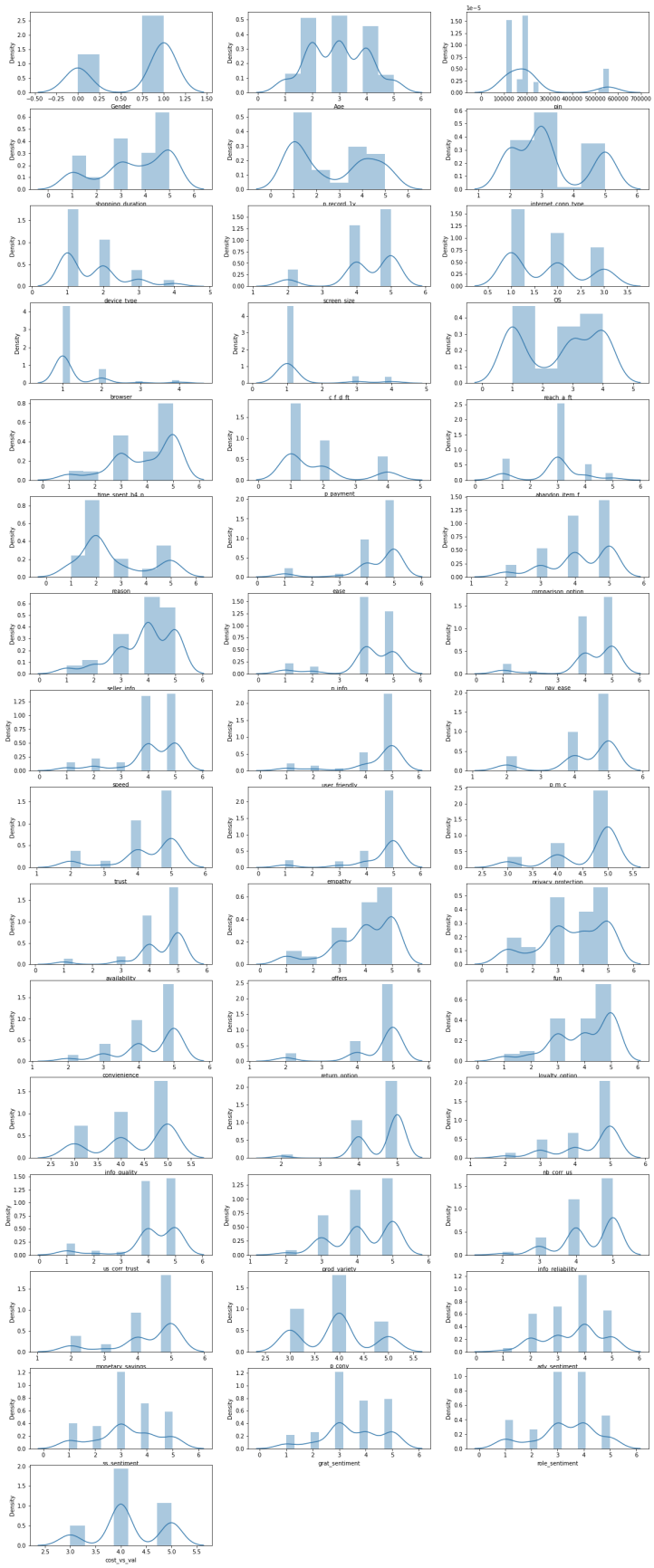
```
# Relation of offers and fun
D=df.groupby('offers').mean()
plt.figure(figsize=(15,8))
D['fun'].plot()
```



We can see that people have more fun using the website when there are more offers present. For more relations like these, connect to the code on GitHub.

We can also plot the distribution of our dataset. This can be used to check for skew, outliers, value scarcity etc.

```
#Distribution of numerical columns in df
plt.figure(figsize=(20,80),facecolor='white')
plotnumber=1
for i in df:
    if df[i].dtype != 'object':
        if plotnumber<=75:
            ax=plt.subplot(25,3,plotnumber)
            sns.distplot(df[i])
            plt.xlabel(i)
        plotnumber+=1
```



We can see that our data has some skew, feature value imbalance, data imbalance, outliers etc. We will need to process our data to treat these conditions.

More examples of data exploration can be found in the code.

Data Preprocessing

Data preprocessing can refer to manipulation or dropping of data before it is used in order to ensure or enhance performance, and is an important step in the data mining process. The phrase "garbage in, garbage out" is particularly applicable to data mining and machine learning projects. Thus we need to ensure that we fit our model with the best possible version of our data.

In our case, we first start this process by renaming the columns to make the data easier to work with:

```
df.rename(columns = {'def_col_name':'revised_col_name'}, inplace = True)
```

We then split the columns that are permutations of user input.

```
col_l = ['col_name_1','col_name_2',,,'col_name_n']

for i in col_l:
    amazon=df[i].str.contains(pat = 'Amazon.in')
    flipkart = df[i].str.contains(pat = 'Flipkart.com')
    Myntra = df[i].str.contains(pat = 'Myntra.com')
    paytm = df[i].str.contains(pat = 'Paytm.com')
    sd = df[i].str.contains(pat = 'snapdeal.com')

    df['Amazon_' +str(i)] = amazon
    df['Flipkart_' +str(i)] = flipkart
    df['Myntra_' +str(i)] = Myntra
    df['Paytm_' +str(i)] = paytm
    df['Snapdeal_' +str(i)] = sd
```

We also need to remove the outliers. For this, we have used Z-score method.

The Z-score method for outlier detection helps to understand if a data value is greater or lesser than mean and how far away it is from the mean. More specifically, Z score tells how many standard deviations away a data point is from the mean.

The snippet used in our code is provided below

```
#Removing outliers using zscore
z=np.abs(stats.zscore(df))
np.where(z>4)
```

This provides a numpy array where z is greater than 4. We can then remove that data using

```
index=(np.where(z>4)[0])
df=df.drop(df.index[index])
```

This will drop the rows where z is greater than 4. You can verify this step by using `df.shape()`. The difference in rows will tell you how many rows were dropped and the percentile can be calculated to monitor data loss.

To fix the skew in our data, we first need to separate our target and feature columns. We can do this by

```
# separating target and features
x=df.drop(columns=['Amazon_r','Flipkart_r','Myntra_r','Paytm_r',
                   'Snapdeal_r'],axis=1)
y=df[['Amazon_r','Flipkart_r','Myntra_r','Paytm_r','Snapdeal_r']]
```

We can then call the PowerTransformer method to fix the skew. You can execute `x.skew()` to monitor skew values of each feature. We have used the 'yeo-johnson' method to fix the skew in our data.

```
power = PowerTransformer(method='yeo-johnson', standardize=True)
data_new = power.fit_transform(x)
data_new=pd.DataFrame(data_new,columns=x.columns)
print(data_new.skew())
x = data_new
```

We can then plot the features to check their distributions. The snippet can be found in the data analysis section above.

Building Machine Learning (ML) models

Now that our feature set looks satisfactory, we can start building our model.

```
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.25)

# List of regressors and scorers
en = ElasticNet()
dtr = DecisionTreeRegressor(random_state=1,max_depth=3)
rfr = RandomForestRegressor(random_state=1)
m_list= [en,dtr,rfr]
s=SCORERS
s.keys()
```

In the snippet provided below, we have first split out data into training and testing data frames. The split size has been declared as 0.25, but, you can experiment and chose the

split ratio that best suits your requirements. We have then created instances of our classifiers and built a list consisting of their identifiers.

We can now begin to fit our models. The following snippet fits our data with each of the instances in our `m_list`. It then prints the training and testing scores with CV and difference between testing accuracy and CV.

```
# Training models and getting CV score
for m in m_list:
    m.fit(X_train,y_train)
    preds = m.predict(X_test)
    rmse= np.sqrt(mean_squared_error(y_test,preds))
    print(m)
    print('Root_mean_squared_error: ',rmse)
    print('cross validation scores below:--  \n',m)
    print('root_mean_squared_error:
',cross_val_score(m,x,y,cv=5,scoring='neg_root_mean_squared_error').mean(
))
    print('difference between model score and cross validation score: ',-
rmse-
cross_val_score(m,x,y,cv=5,scoring='neg_root_mean_squared_error').mean())
    print('\n')
    print('\n')
```

The output is provided below.

```
ElasticNet()
Root_mean_squared_error:  0.3865730602705447
cross validation scores below:--
 ElasticNet()
root_mean_squared_error:  -0.33597175829788295
difference between model score and cross validation score:
-0.05060130197266177



DecisionTreeRegressor(max_depth=3, random_state=1)
Root_mean_squared_error:  0.1088617151867933
cross validation scores below:--
 DecisionTreeRegressor(max_depth=3, random_state=1)
root_mean_squared_error:  -0.10338034809267685
difference between model score and cross validation score:
-0.005481367094116452



RandomForestRegressor(random_state=1)
Root_mean_squared_error:  0.010907131485472218
cross validation scores below:--
 RandomForestRegressor(random_state=1)
root_mean_squared_error:  -0.002272797835477087
difference between model score and cross validation score:
-0.008634333649995132
```

Hyper-parameter Tuning (HPT)

For this experiment, we have we have used GridSearchCV for HPT. The example is demonstrated with Decision Tree Regressor. The provided snippet is extensible to other regressors when the appropriate parameter grid is used. Check the documentation of the your regressor to build an appropriate parameter grid.

```
# Parameter Grid
grid_param = {'criterion':['mse','friedman_mse','mae','poisson'],
              'splitter': ['best','random'],
              'max_features': ['auto','sqrt','log2'],
              'min_samples_leaf':range(1,100,3),
              'max_leaf_nodes':range(1,100,3),
              'max_depth':range(1,3)}
```

```
grid_search=GridSearchCV(DecisionTreeRegressor(random_state=1),param_grid
=grid_param,cv=5,n_jobs=15,verbose = 1)
grid search.fit(X train,y train)
```

We can then fetch the best parameters from the conducted search by executing

```
grid_search.best_params_
```

We can then build our model using this optimal parameter configuration.

```
# Model Fitted with best Parameters
dt=KNeighborsClassifier(algorithm= 'auto',
 leaf_size= 10,
 n_neighbors= 2,
 p= 1,
 weights= 'uniform')
dt.fit(x_train,y_train)
y_pred=dt.predict(x_test)
```

We can now save and load our model accordingly.

```
import joblib

joblib.dump(dt,'DT_best.obj')
joblib.load('DT_best.obj')
```

References

1. https://numpy.org/doc/stable/
2. https://pandas.pydata.org/docs/
3. https://scikit-learn.org/stable/
4. https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.StandardScaler.html
5. https://www.geeksforgeeks.org/z-score-for-outlier-detection-python/
6. https://docs.scipy.org/doc/scipy/reference/generated/scipy.stats.yeojohnson.html
7. https://towardsdatascience.com/introduction-to-regression-analysis-9151d8ac14b3
8. https://towardsdatascience.com/understanding-auc-roc-curve-68b2303cc9c5
9. https://towardsdatascience.com/machine-learning-basics-random-forest-regression-be3e1e3bb91a
10. https://towardsdatascience.com/gridsearchcv-for-beginners-db48a90114ee