SMART BUILD

INSIGHTS GAINED REPORT

Data Science and Machine Learning

By:Zalan, Shanaya, Jiri, Phat

INDEX

01

TASK AND GOALS

- task is to be implemented with Python
- Presentation of the prediction models
- Implications on SmartBuild's business.
- Give a viable solution

02

Q1 PREDICTIVE MODEL

- Goal
- Models used
- Input and output features
- Model 1 – XGB Regressor
- Model 2 - Linear Regression
- Model 3 - Polynomial Regression
- Model comparison and result

GOAL:

Make reliable predictions and compare the three models

Models Used:

- **XGBRegressor from xgboost**
- **LinearRegression from sklearn**
- **Poly1d from numpy**

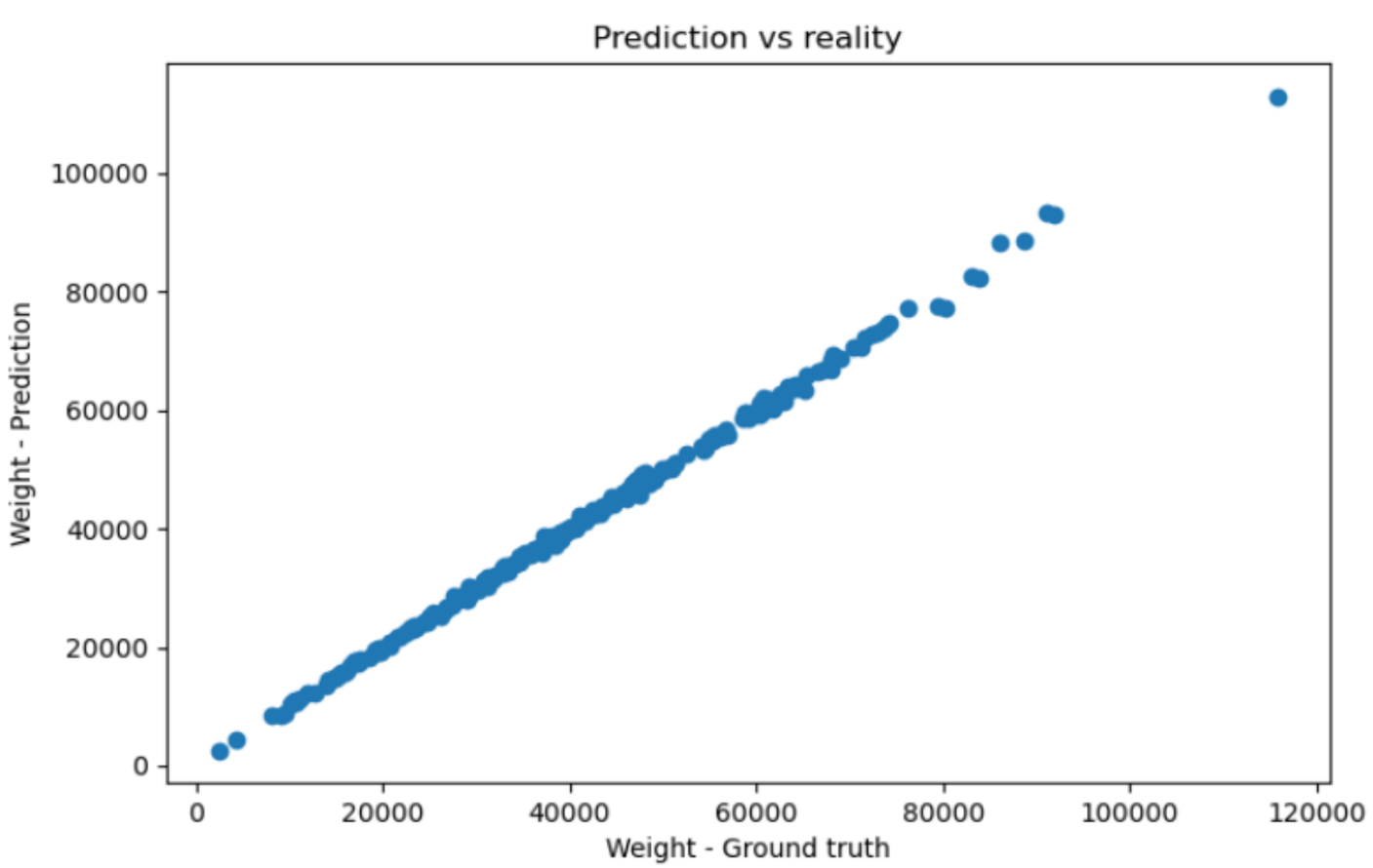
Input features *- width, height, ionizationclass, FluxCompensation, pressure, karma, modulation*

Output *- Predicted weight*

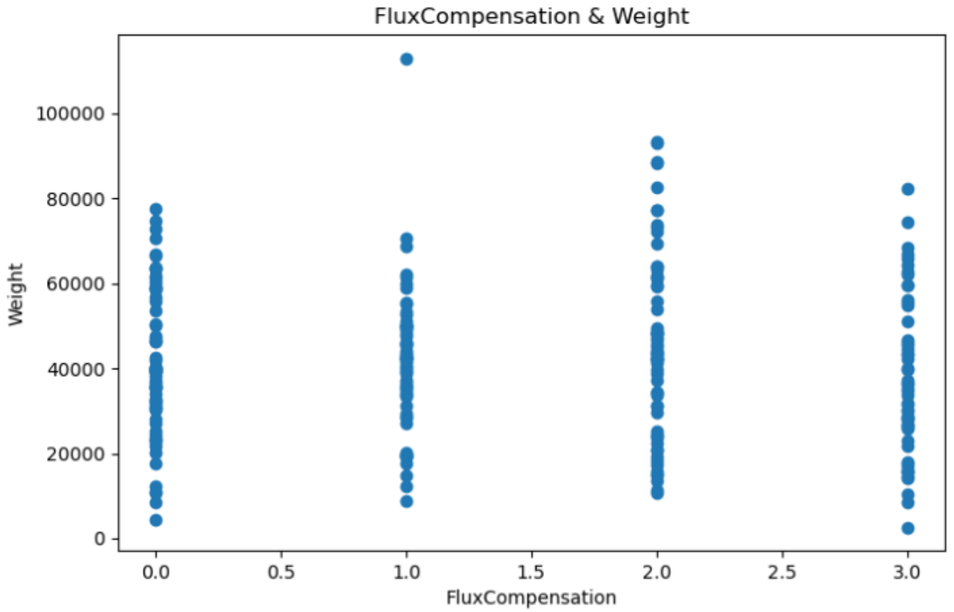
MODEL 1 – XGB REGRESSOR

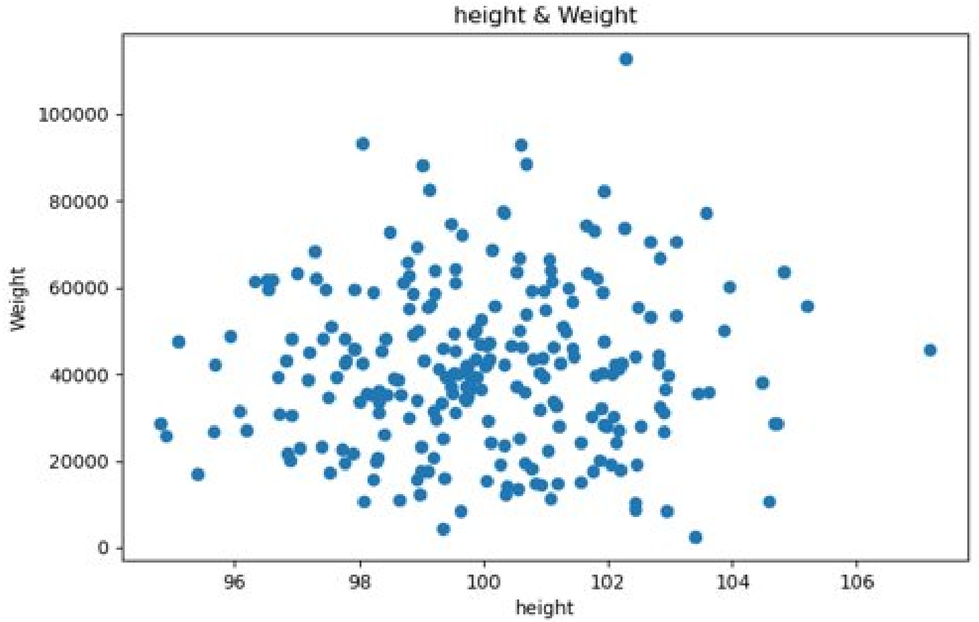
MAIN INSIGHTS

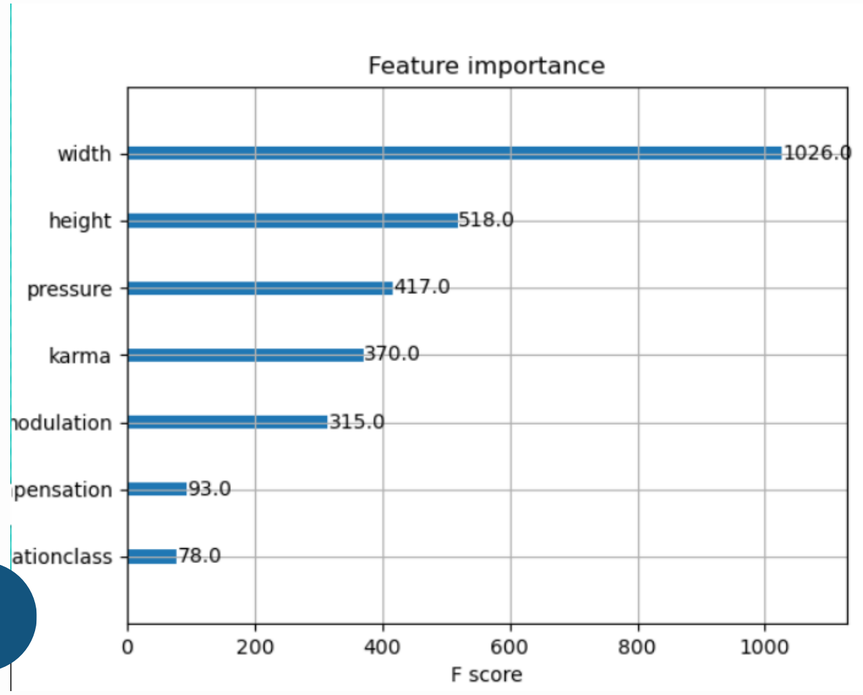
- units: kg
- Model 1 error: 458.53
- Standard dev.: 20300.77
- Baseline error: 14870.99
- Baseline: mean value

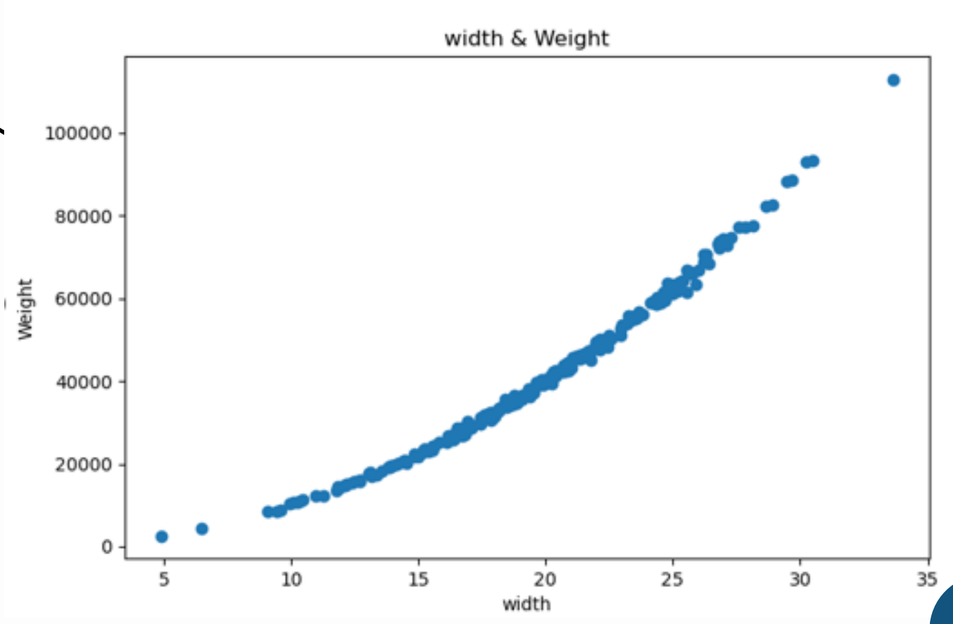


DATA


FluxCompensation & Weight
Weight
FluxCompensation
0
20000
40000
60000
80000
100000
0.0
0.5
1.0
1.5
2.0
2.5
3.0


height & Weight
Weight
height
0
20000
40000
60000
80000
100000
96
98
100
102
104
106

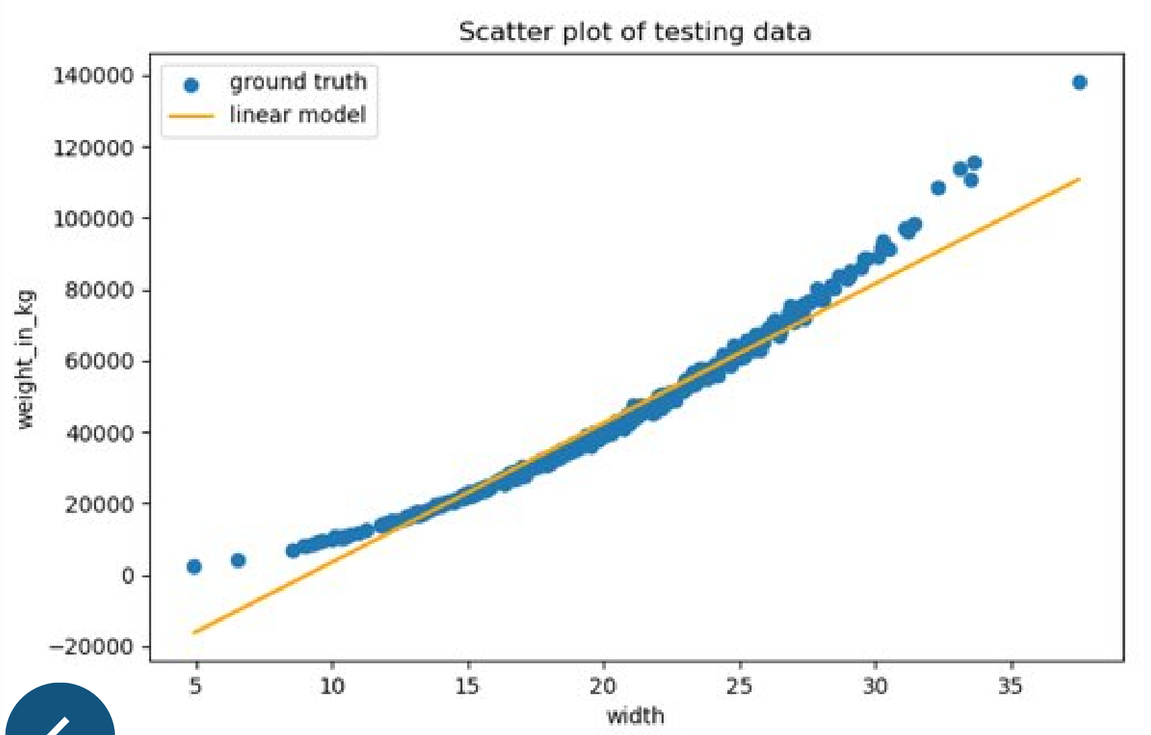

Feature importance
width
height
pressure
karma
odulation
pensation
ationclass
1026.0
518.0
417.0
370.0
315.0
93.0
78.0
0
200
400
600
800
1000
F score

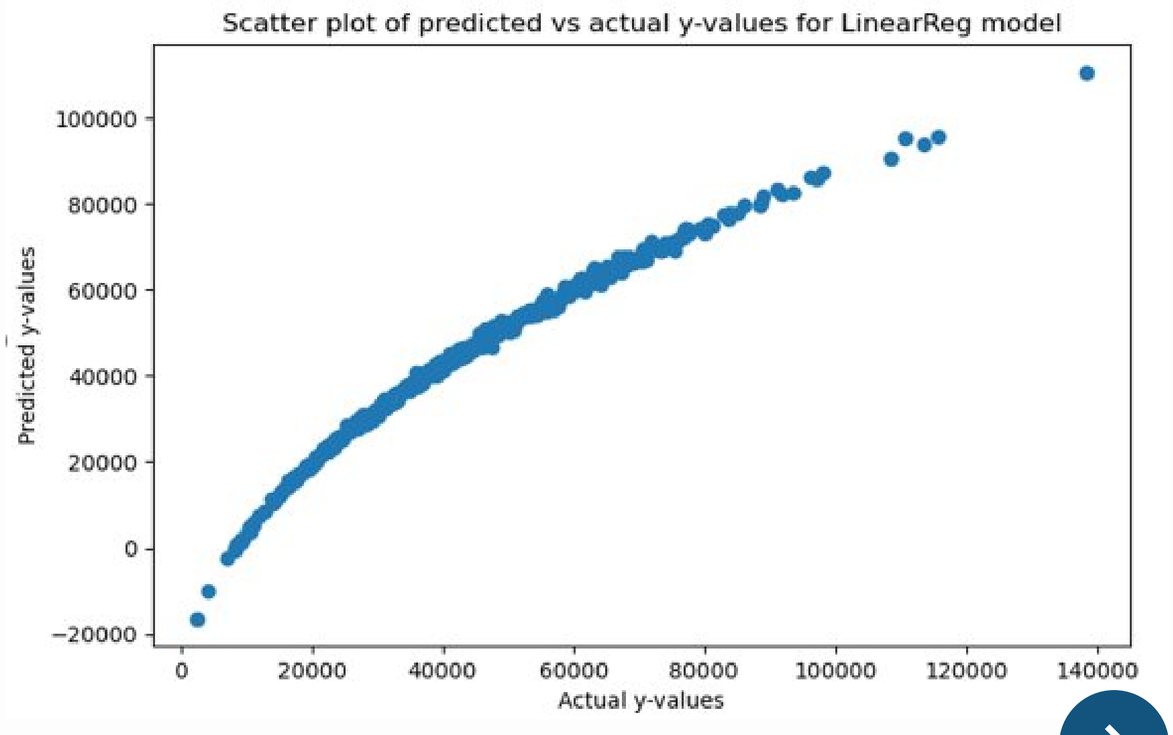

width & Weight
Weight
width
0
20000
40000
60000
80000
100000
5
10
15
20
25
30
35

MODEL 2 - LINEAR REGRESSION

MAIN INSIGHTS

Mean Abs. Error = 2554.70 **Baseline error** = 15906.73 ***Linear_model.LinearRegression()***

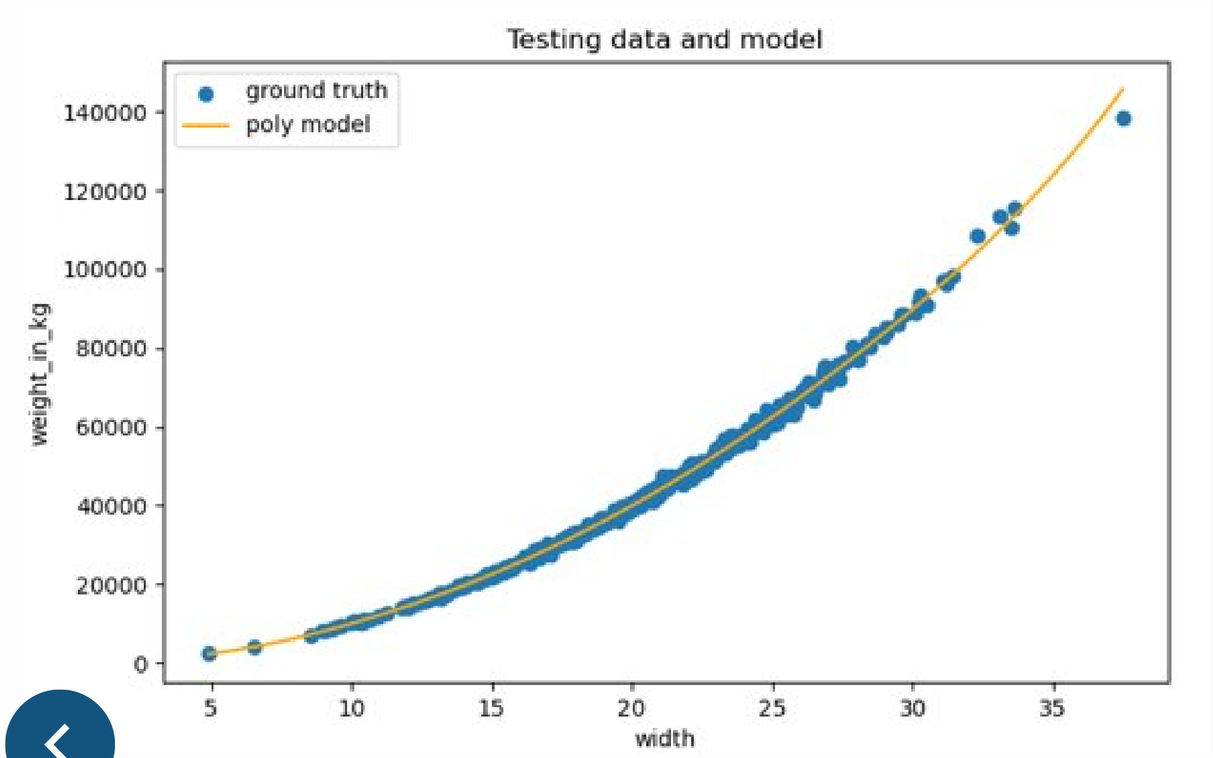


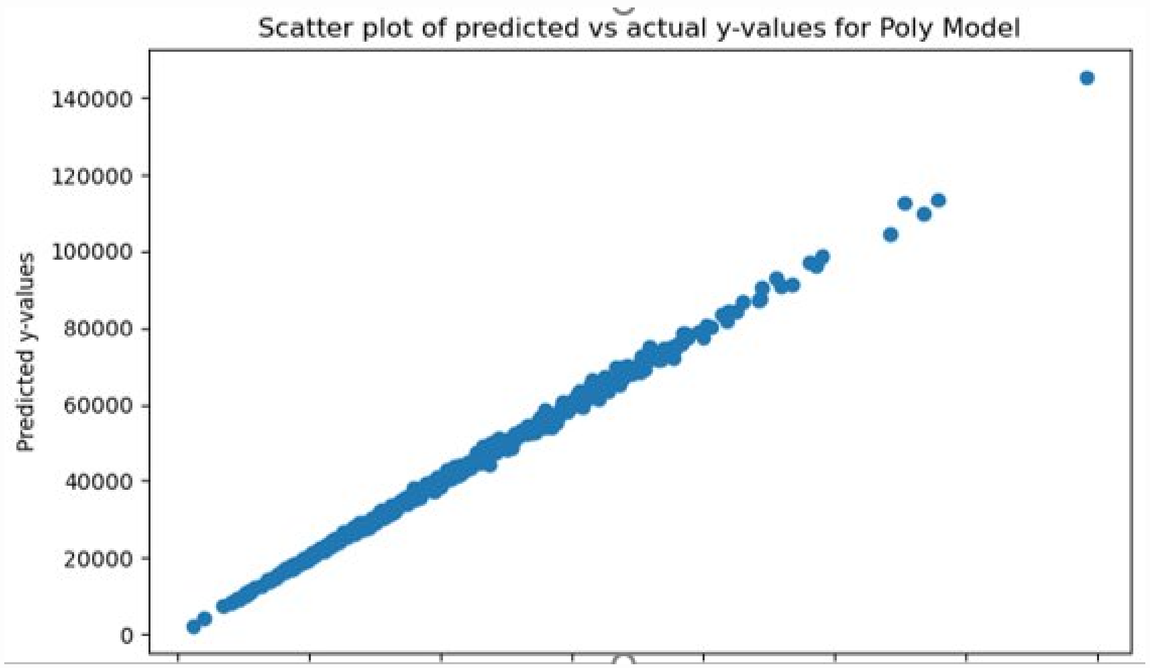


MODEL 3 - POLYNOMIAL REGRESSION

MAIN INSIGHTS

Mean Abs. Error = 705.85 **Baseline error** = 15906.73 ***np.poly1d(np.polyfit(X,Y,degree))***





MODEL COMPARISON

Baseline error: 14870

M1 = XGB Regressor
M2 = Linear Regression
M3 = Polynomial Regression

BEST MODEL
XGB REGRESSOR

03

Q2 PREDICTIVE MODEL

- Goal
- Models used
- Input and output features
- Model 1 – XGB Classifier
- Model 2 - DecisionTreeClassifier
- Model comparison and result

GOAL:

Make reliable predictions and compare the two models

Models Used:

- **XGB regressor from xgboost**
- **DecisionTreeClassifier from sklearn**

Input features *- width, height, ionizationclass, FluxCompensation, pressure, karma, modulation*
Output *- predicted error type*

MODEL 1 - XGB CLASSIFIER

MAIN INSIGHTS

- Accuracy = 0.592
- Reliably detects the presence of an error (0.92 accuracy)
- Problems with the type of error
- Second model for detecting type of error:
  Accuracy = 0.339

Actual	0	1	2	3
0	112	8	5	2
1	2	10	14	10
2	2	17	13	10
3	1	12	17	10

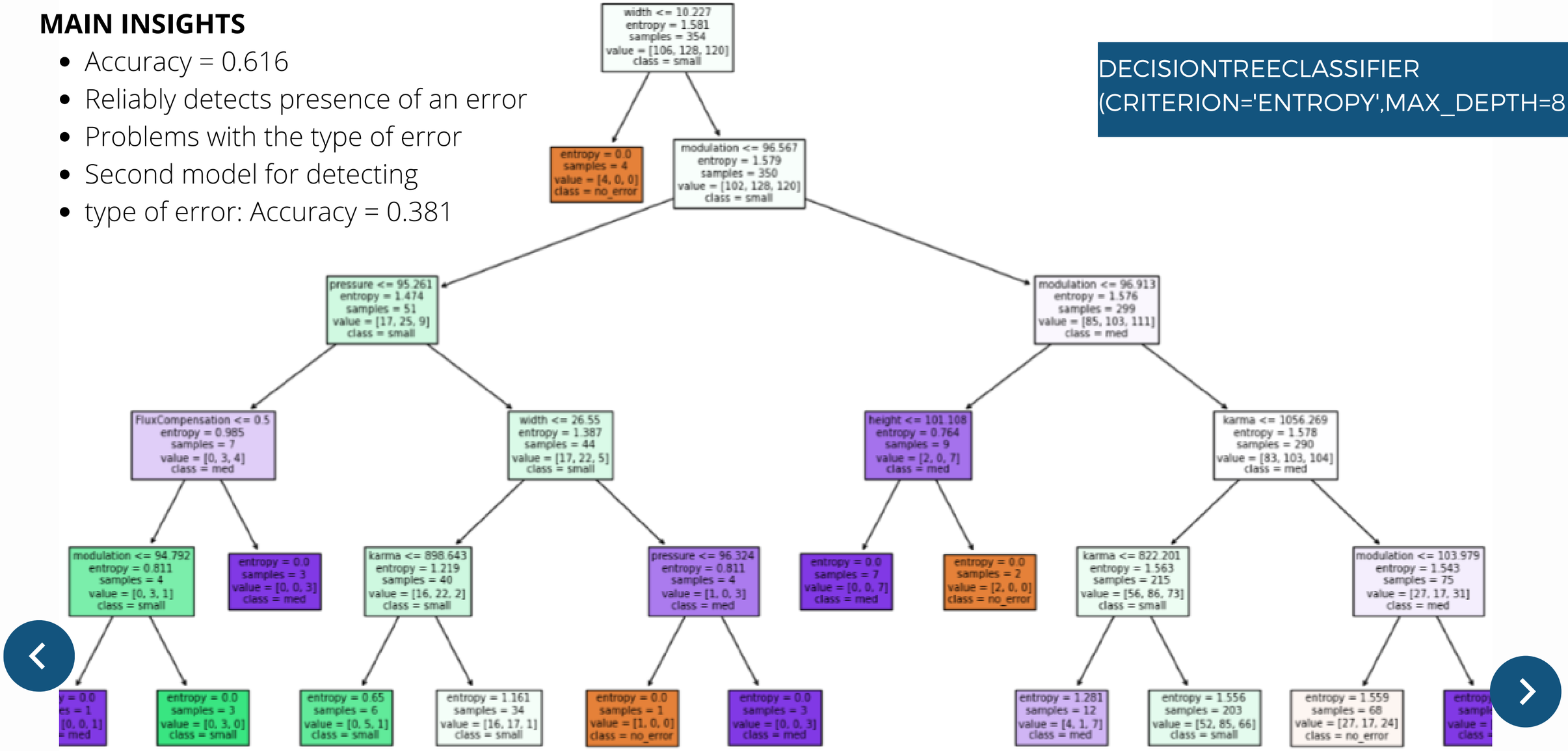
MODEL = XGBCLASSIFIER()

MODEL 2 - DECISIONTREECLASSIFIER

MAIN INSIGHTS

- Accuracy = 0.616
- Reliably detects presence of an error
- Problems with the type of error
- Second model for detecting
- type of error: Accuracy = 0.381

DECISIONTREECLASSIFIER (CRITERION='ENTROPY',MAX_DEPTH=8)



04

QUALITY PREDICTION

- Goal
- Models used
- Input and output features
- Exploring the data
- Visualizing data
- Model comparison and result

GOAL:

Make reliable predictions and compare the two models

Models Used:

- **Linear Regression from sklearn**
- **Poly1d from numpy**

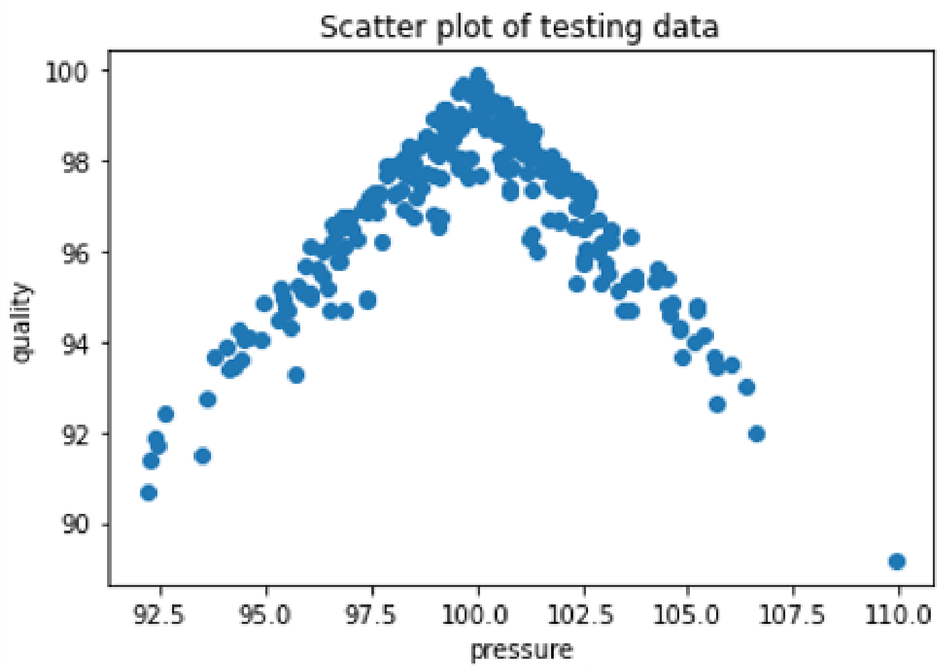
Input features *- pressure*
Output *- quality*

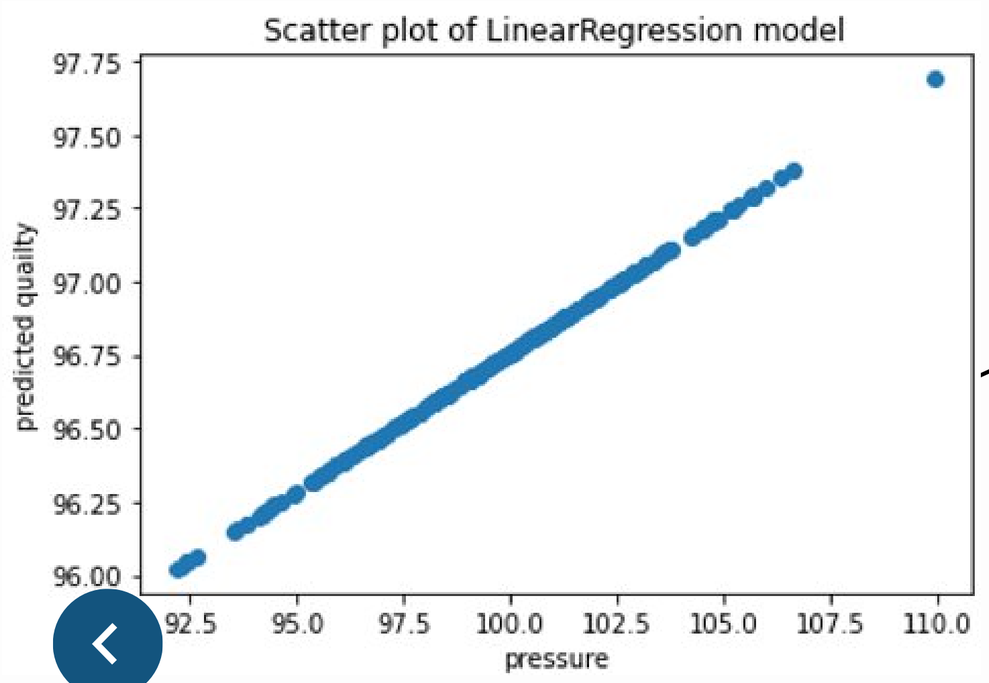
EXPLORING THE DATA

Pressure

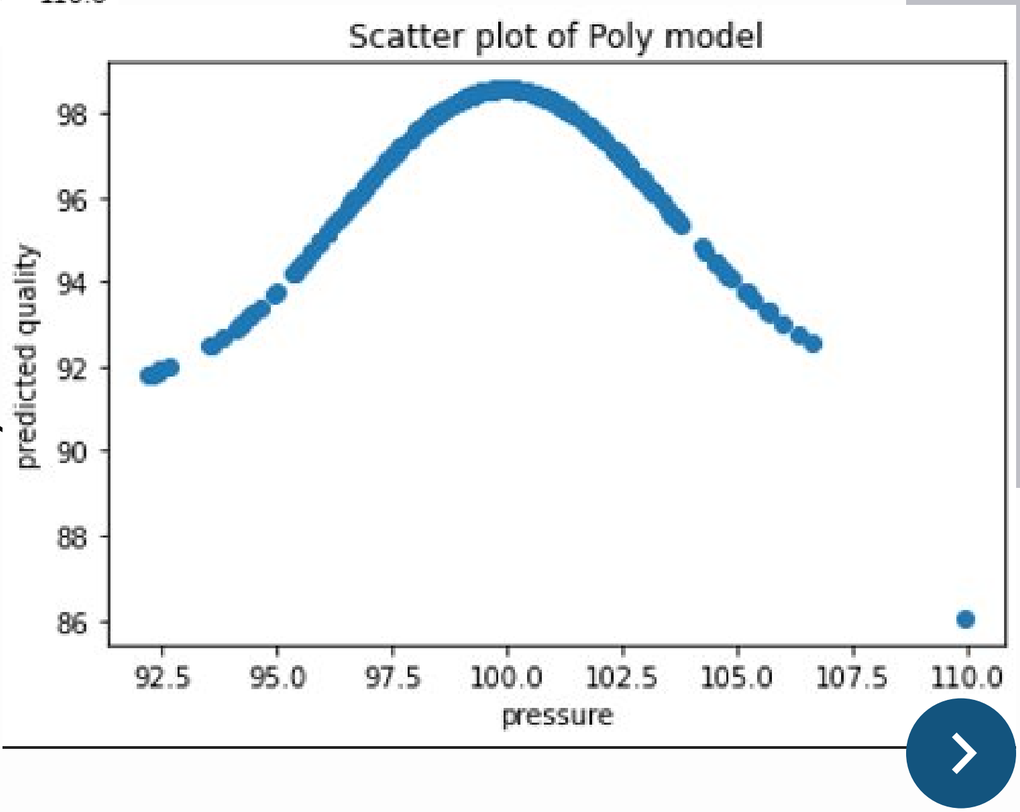
```
          pressure
count  1000.000000
mean     99.788820
std       3.059445
min      90.279123
25%      97.666845
50%      99.903188
75%     101.880290
max     109.913957

count    1000.000000
mean       96.753492
std         1.976024
min        88.932730
25%        95.586433
50%        97.170427
75%        98.202438
max        99.904697
```

```
inputColumns = ['pressure']

output = 'Quality’


X = df[inputColumns]

Y = df[output]


print(X.describe())

print()

print(Y.describe())
```

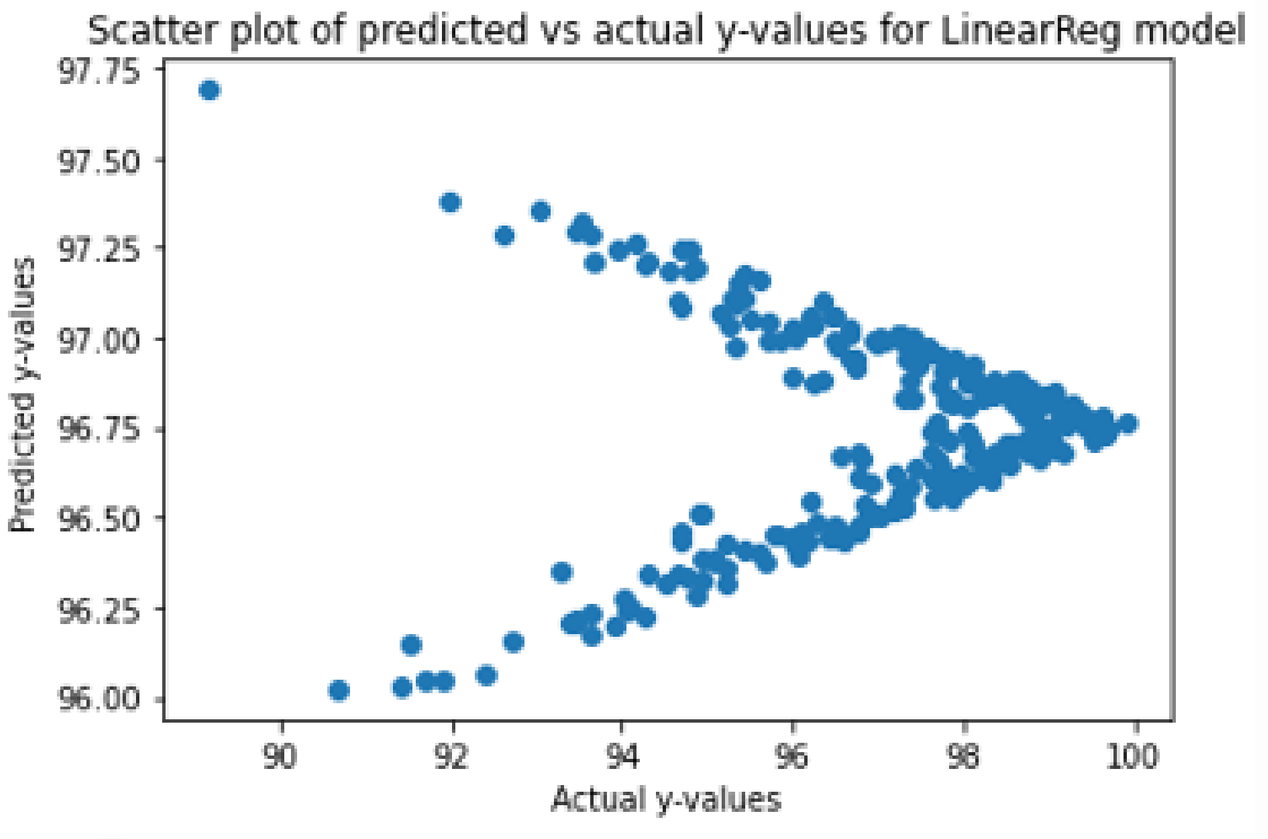

Scatter plot of testing data
quality
pressure
100
98
96
94
92
90
92.5
95.0
97.5
100.0
102.5
105.0
107.5
110.0

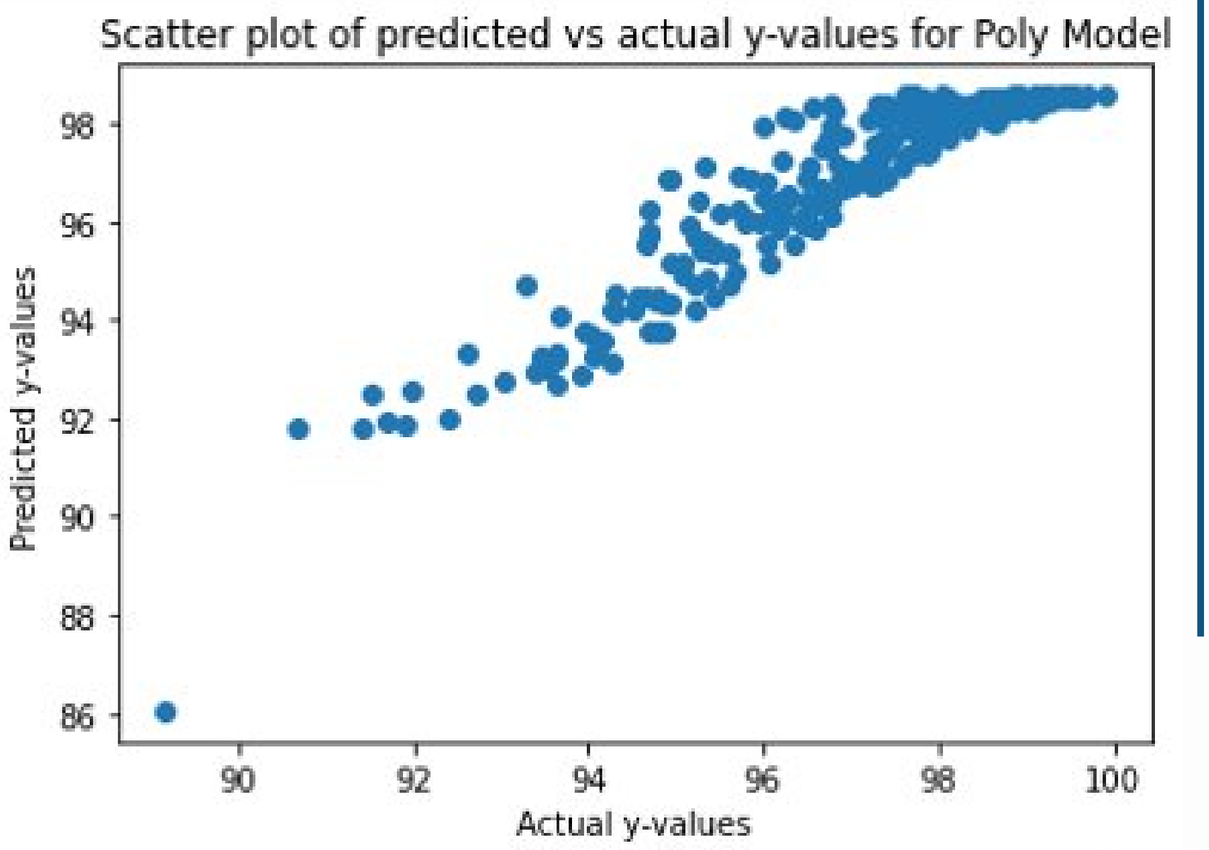

Scatter plot of LinearRegression model
predicted quality
pressure
97.75
97.50
97.25
97.00
96.75
96.50
96.25
96.00
92.5
95.0
97.5
100.0
102.5
105.0
107.5
110.0

DATA


Scatter plot of Poly model
predicted quality
pressure
98
96
94
92
90
88
86
92.5
95.0
97.5
100.0
102.5
105.0
107.5
110.0

VISUALIZING AND COMPARING



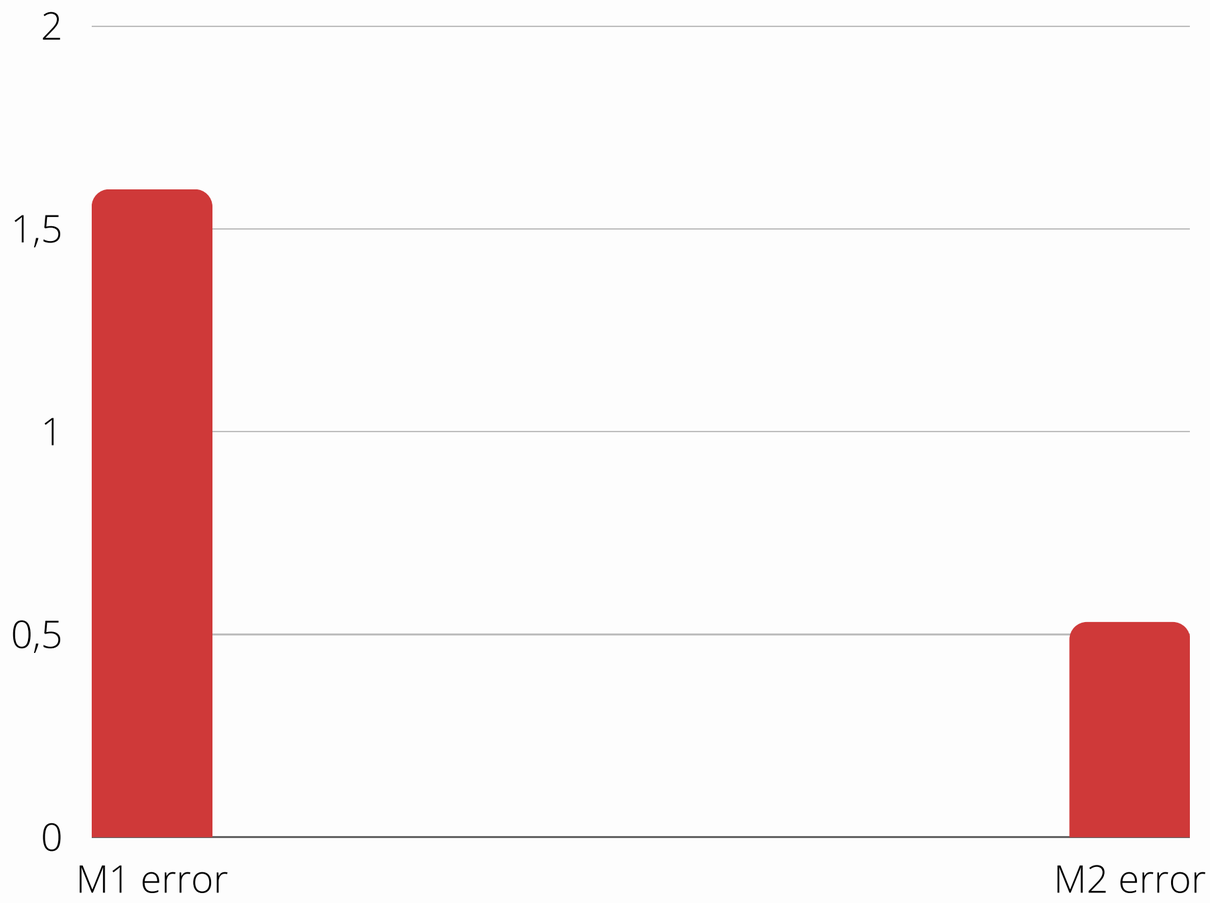


MODEL COMPARISON

- Mean absolute error for Linear Regression = 1.596348
- Mean absolute error for Poly1d = 0.52938295

M1 = Linear Regression
M2 = Poly1d Regression

BEST MODEL
POLY1D REGRESSION



DETAILS, CODE, DATA PREP

```
#splitting the data into training and testing variables
X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=0.25,
                                                    random_state=42)
#Creating the actual model
model = linear_model.LinearRegression()
#Training the model with the training data
model.fit(X_train, y_train)

deg = 6
X_train_pol = X_train.values.reshape(1,-1)[0]
#Creating and training the model
model_pol = np.poly1d(np.polyfit(X_train_pol,y_train, deg))

#Making predictions using the testing data
y_pred = model.predict(X_test)
y_pred_pol = model_pol(X_test)
```

05

DISTORTION PREDICTION

- Goal
- Models used
- Input and output features
- Exploring the data
- Visualizing data
- Model comparison and result

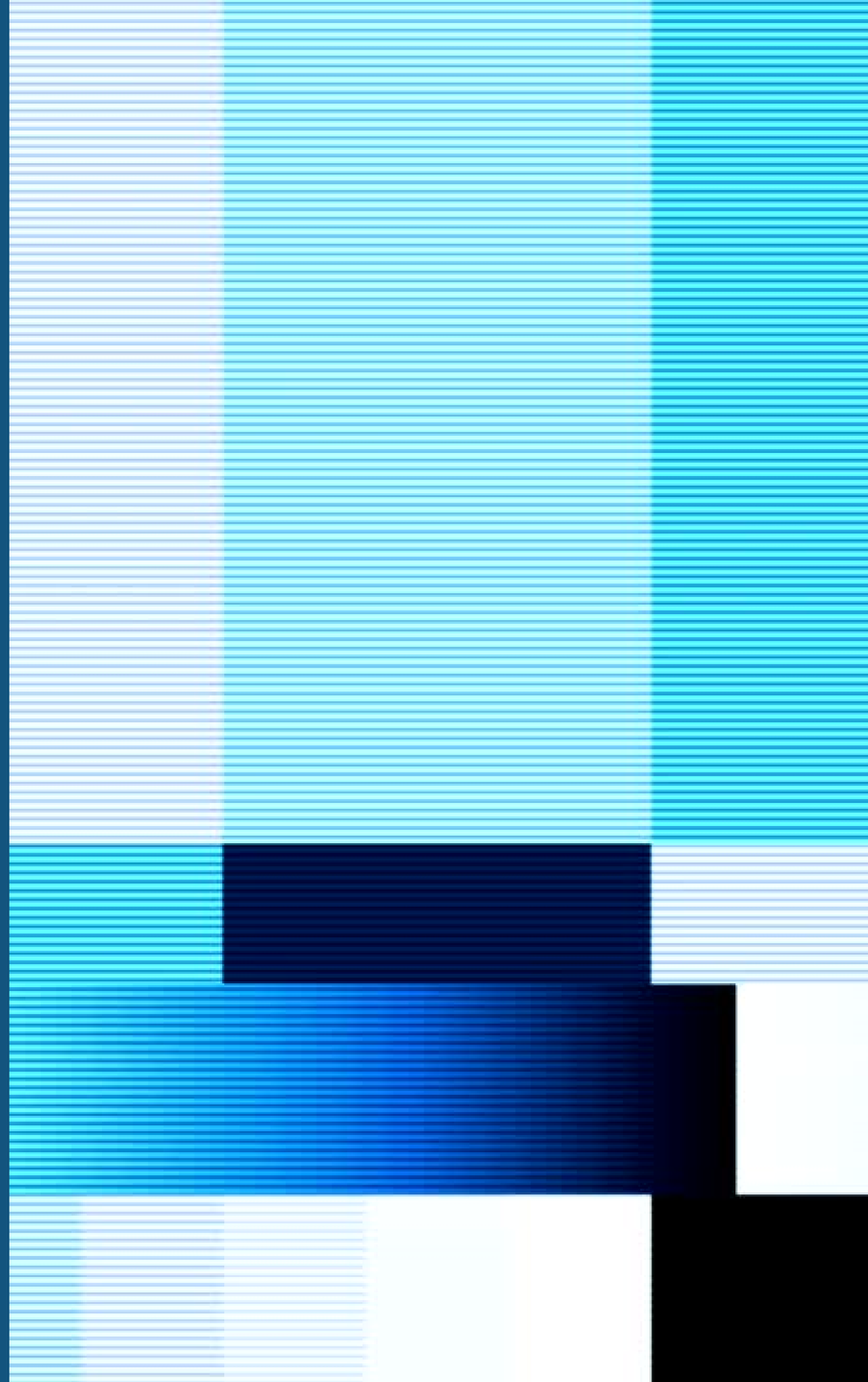
GOAL:

Make reliable predictions

Models Used:

- **XGBRegressor from XGB**

Input features *- karma, pressure, modulation*
Output *- distortion*



EXPLORING THE DATA

```
inputColumns = ['karma','pressure','modulation']
output = ['distortion']

X = df[inputColumns]
Y = df[output]

print(X.describe())
print()
print(Y.describe())
```

	karma	pressure	modulation
count	1000.000000	1000.000000	1000.000000
mean	999.583930	99.788820	99.966848
std	99.018368	3.059445	3.087174
min	670.085902	90.279123	88.319052
25%	932.811626	97.666845	97.955718
50%	999.672417	99.903188	100.091202
75%	1062.986446	101.880290	101.985798
max	1341.389436	109.913957	111.364415

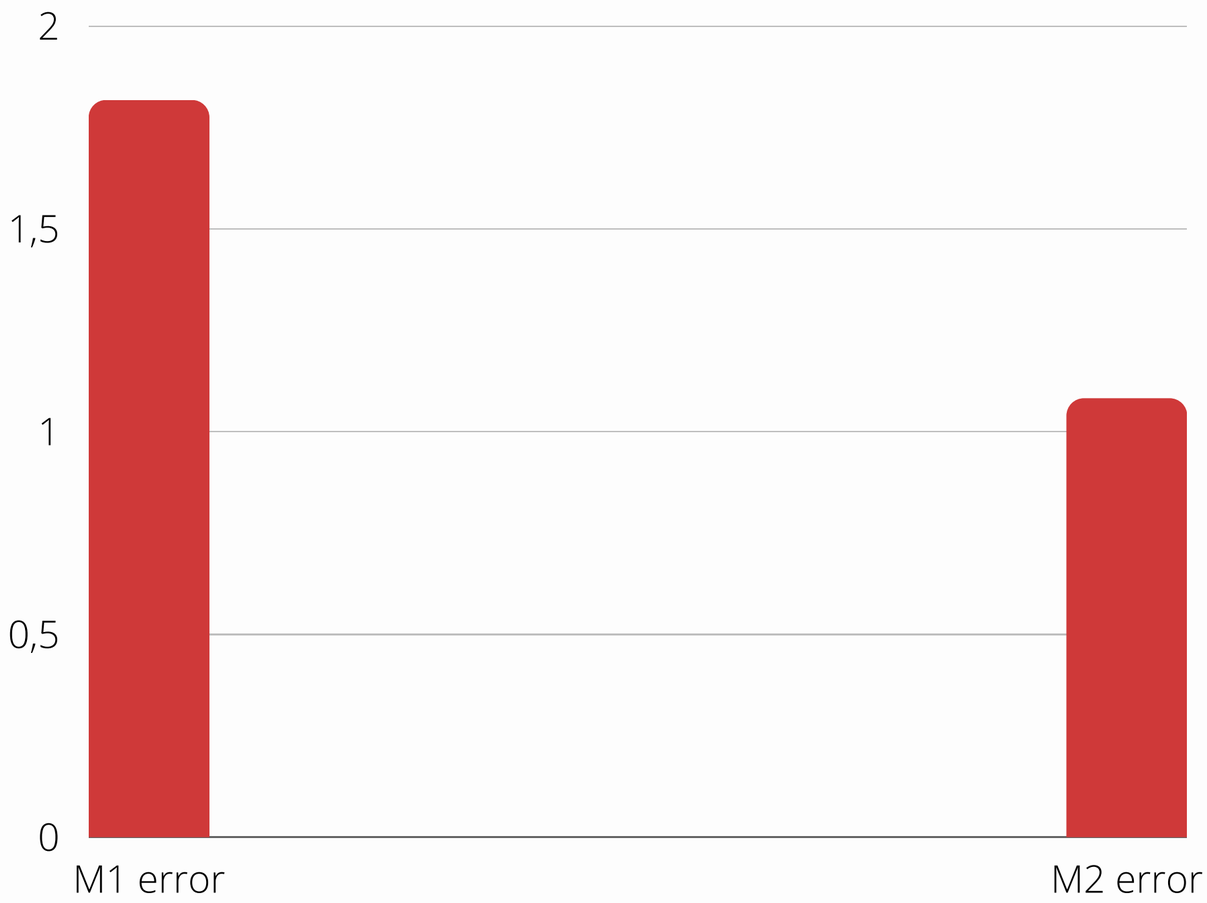
	distortion
count	1000.000000
mean	49.940813
std	1.978294
min	43.536144
25%	48.700090
50%	49.876750
75%	51.325000
max	55.762223

DATA COMPARISON

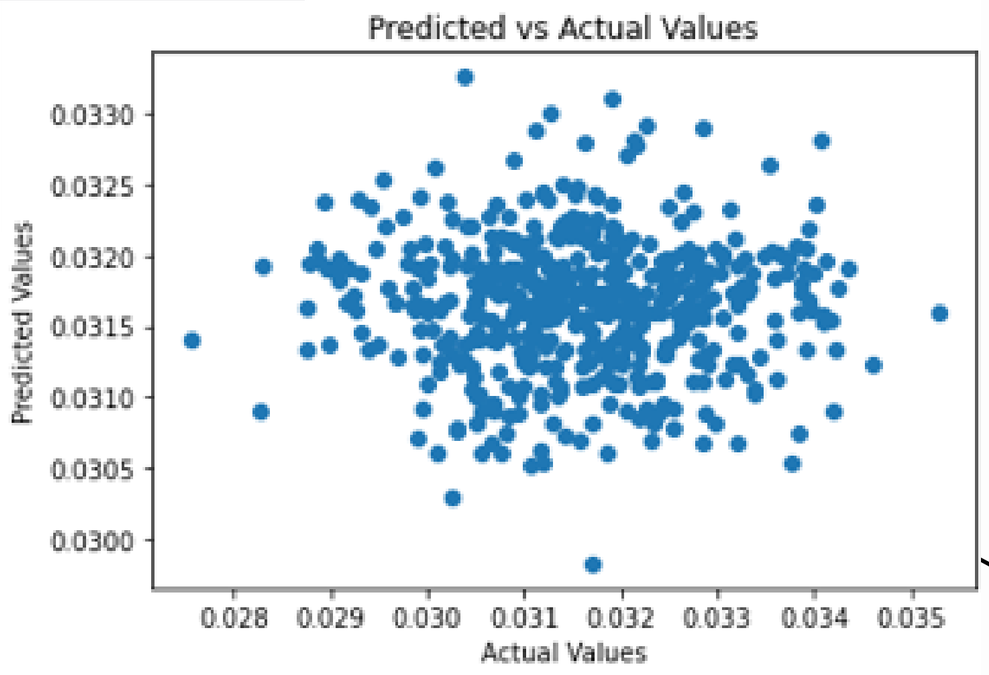
- Mean absolute error for XGBRegressor = 1.81646033
- After normalization = 1.081028238

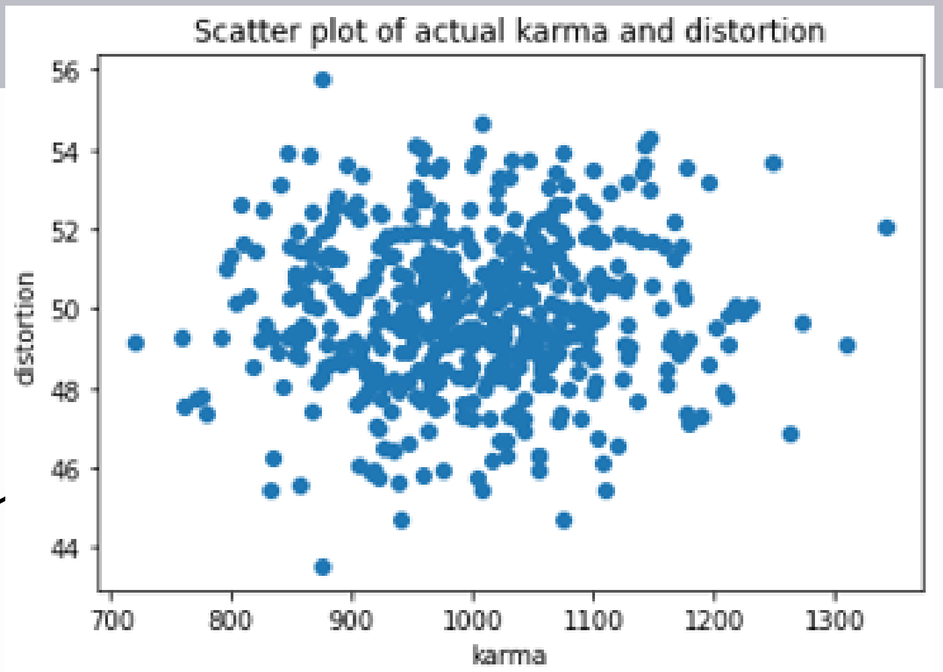
M1 = Linear Regression
M2 = L Regression (Normalization)

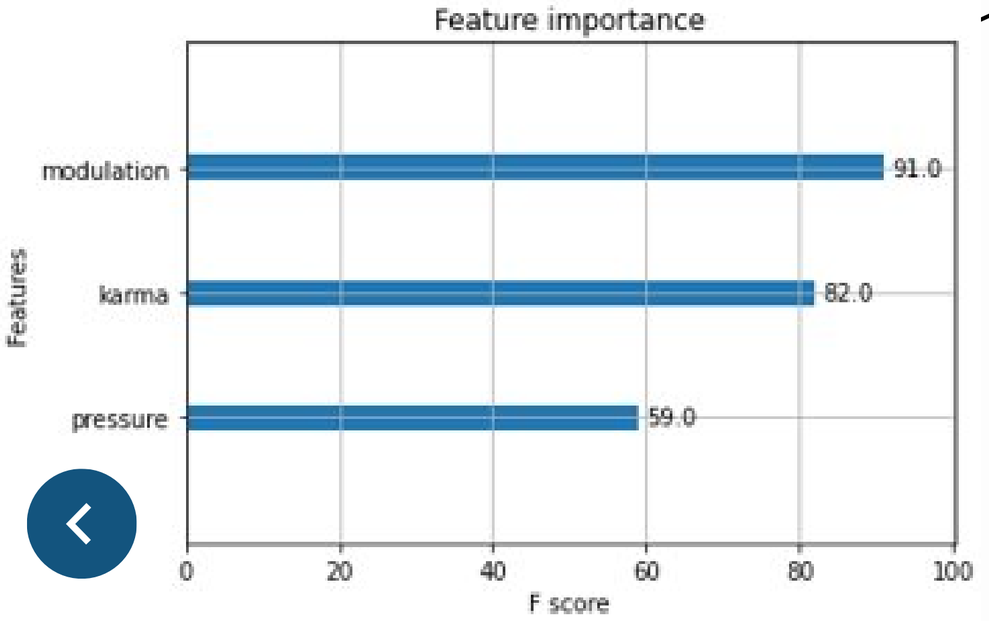
BEST NORMALIZED DATA

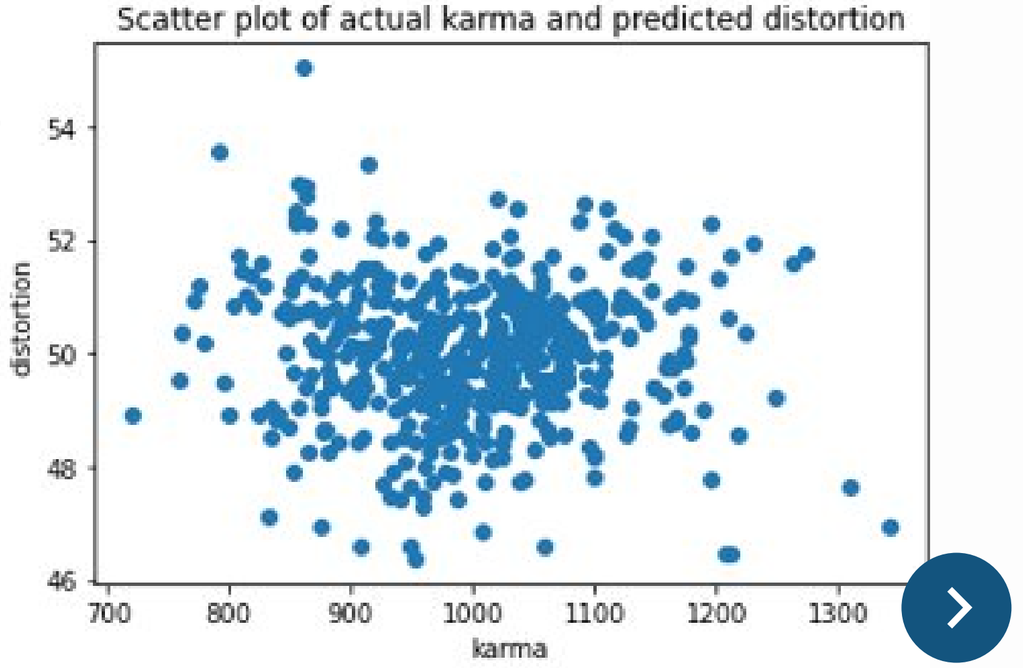


DATA


Predicted vs Actual Values
Predicted Values
Actual Values
0.0330
0.0325
0.0320
0.0315
0.0310
0.0305
0.0300
0.028
0.029
0.030
0.031
0.032
0.033
0.034
0.035


Scatter plot of actual karma and distortion
distortion
karma
56
54
52
50
48
46
44
700
800
900
1000
1100
1200
1300


Feature importance
Features
F score
modulation
karma
pressure
91.0
82.0
59.0
0
20
40
60
80
100


Scatter plot of actual karma and predicted distortion
distortion
karma
54
52
50
48
46
700
800
900
1000
1100
1200
1300

DETAILS, CODE, DATA PREP

```
#splitting the data into training and testing variables
X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=0.5, random_state=42)

#Creating the actual model
model = xgb.XGBRegressor()
#Training the model with the training data
model.fit(X_train, y_train)

#Making predictions using the testing data
y_pred = model.predict(X_test)
```

06

NICESNESS PREDICTION

- Goal
- Models used
- Input and output features
- Exploring the data
- Visualizing data
- Model comparison and result

GOAL:

To make a linear regresion analisis and compere the actual values with the predict ones and check their errors

Models Used:

- **Linear Model from Sklearn**
- **XGBRegressor from XGB**

Input features *- width, height, ionizationclass, FluxCompensation, pressure, karma, modulation*
Output *- predicted nicesness*

EXPLORING THE DATA

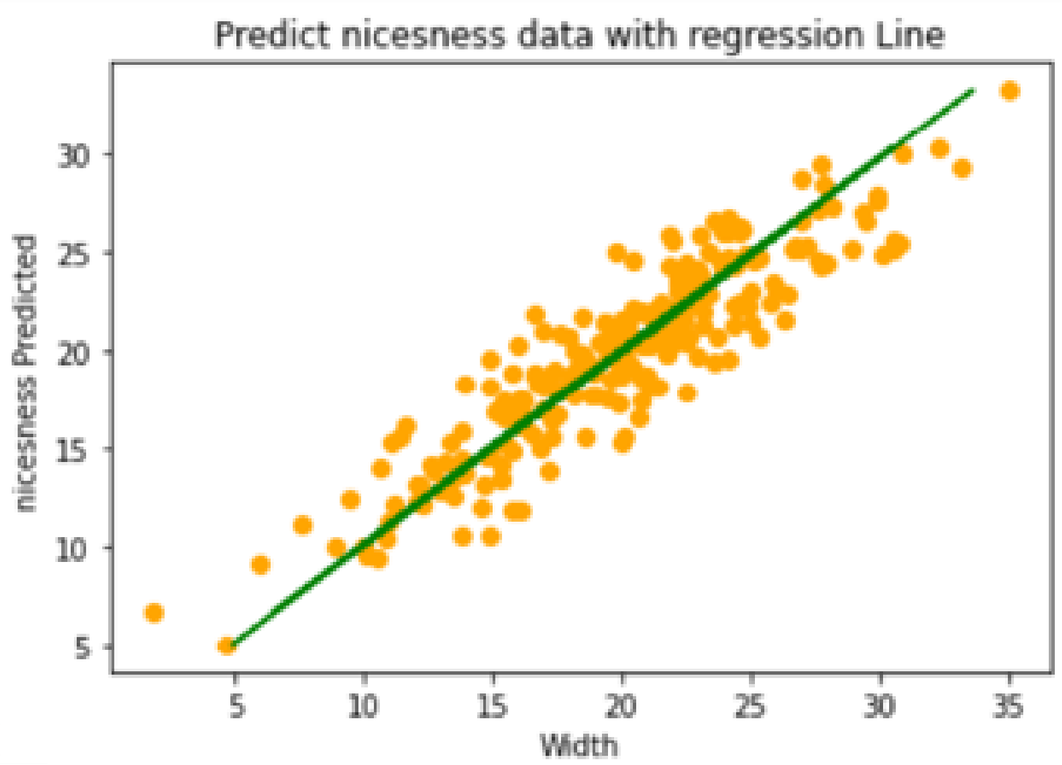
```
inputColumns = ["width","height"]
output = ['niceness']

X = df[inputColumns]
Y = df[output]

print(X.describe())
print()
print(Y.describe())
```

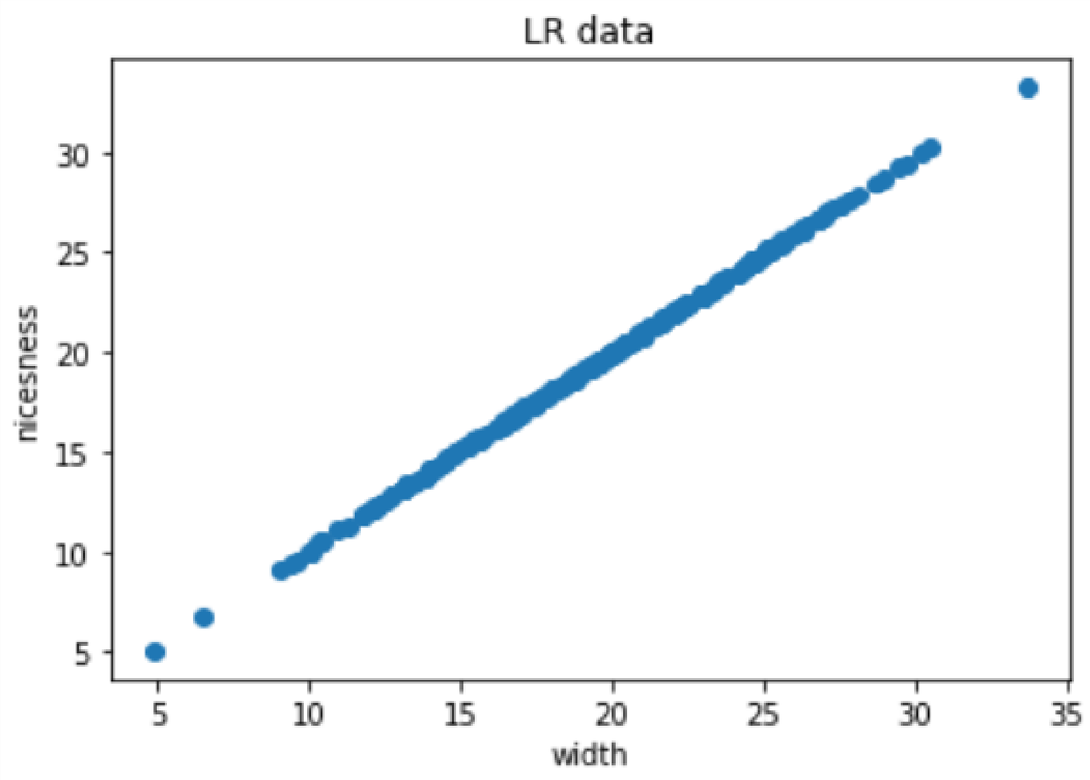
```
          nicesness
count   980.000000
mean      0.030847
std       0.008304
min       0.002870
25%       0.025658
50%       0.030934
75%       0.036430
max       0.060767

            width      height
count   980.000000  980.000000
mean      0.030975    0.031938
std       0.007811    0.000632
min       0.004895    0.030069
25%       0.025967    0.031514
50%       0.031064    0.031930
75%       0.036343    0.032365
max       0.058400    0.034229
```

MODEL 1 - LINEAR REGRESSION

MAIN INSIGHTS

Mean Abs. Error = 0.002703266 **Baseline error** = 0.00651012 ***Linear_model.LinearRegression()***



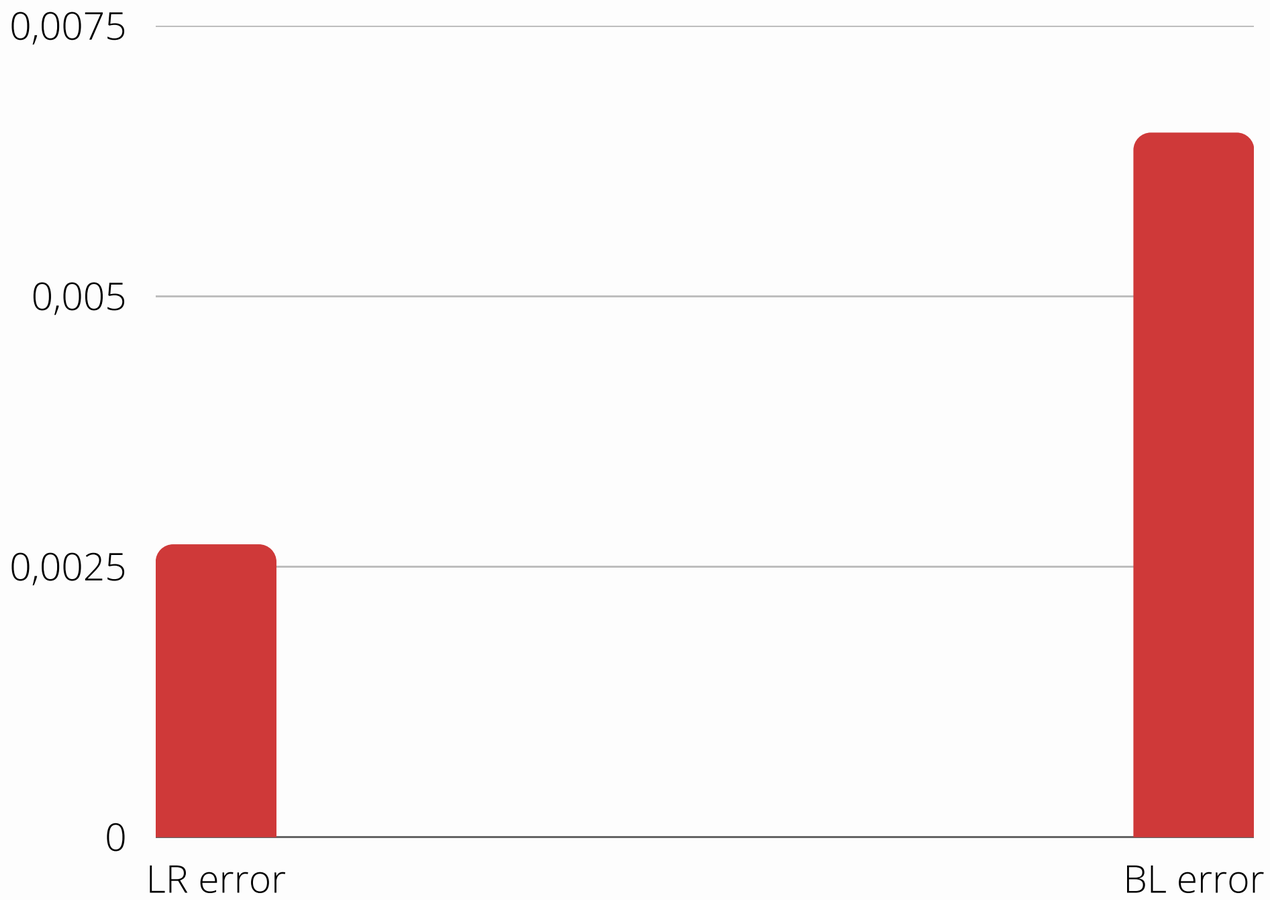


DATA COMPARISON

- Mean absolute error for Linear R = 0.002703266
- Baseline error = 0.0065101225

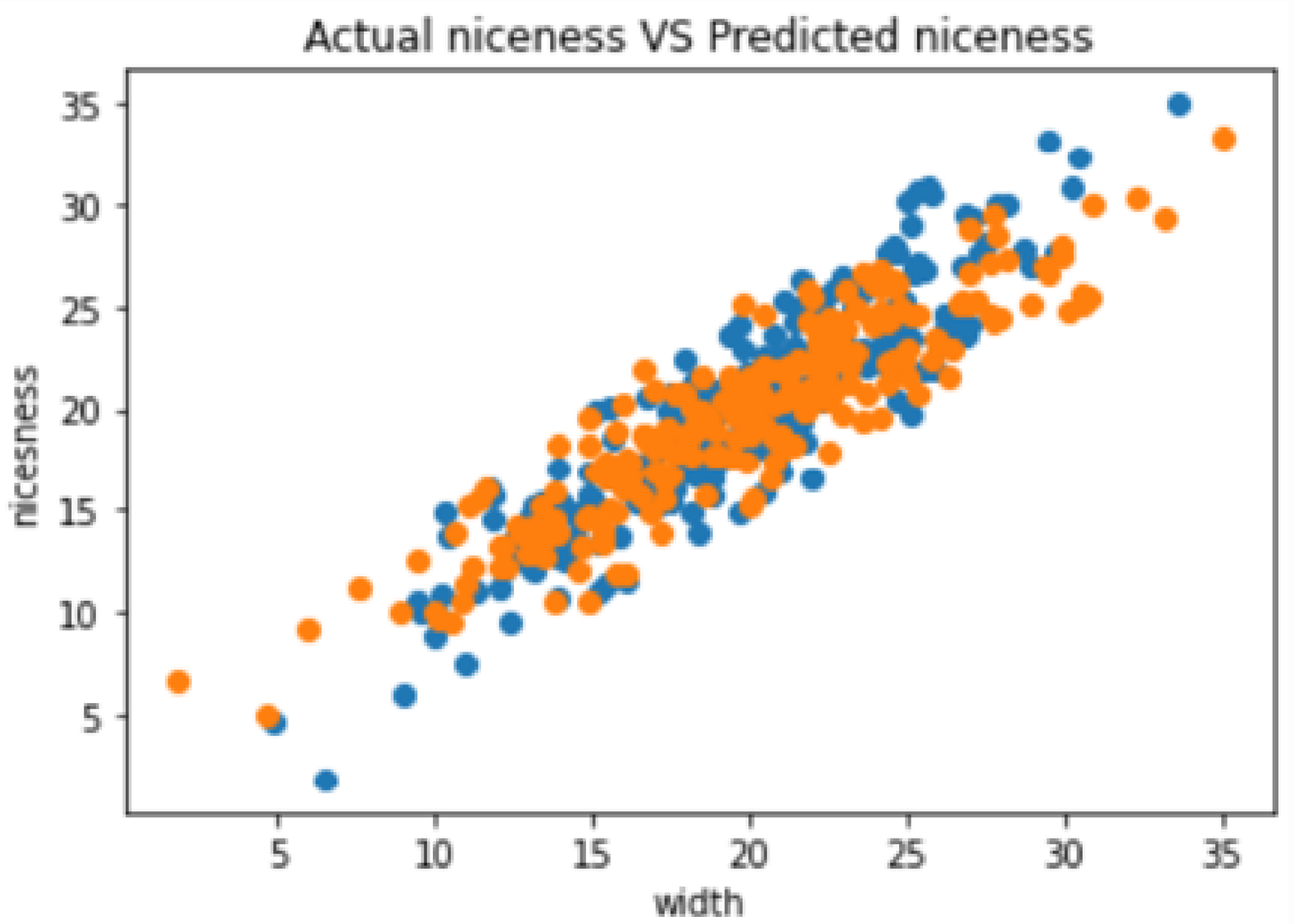
LR = Linear Regression
BL = Baseline

BEST MODEL TRAIN



Comparison

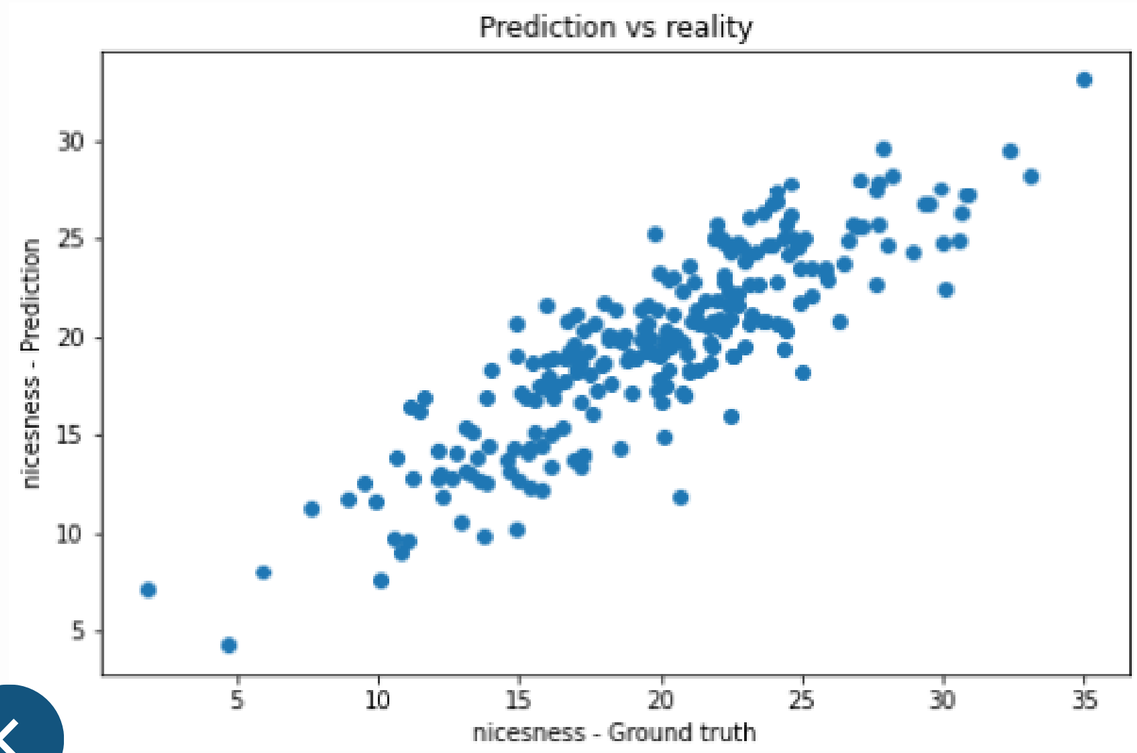
By looking at the dispersion of the points, one can assess the quality of the predictions made by the linear regression model.

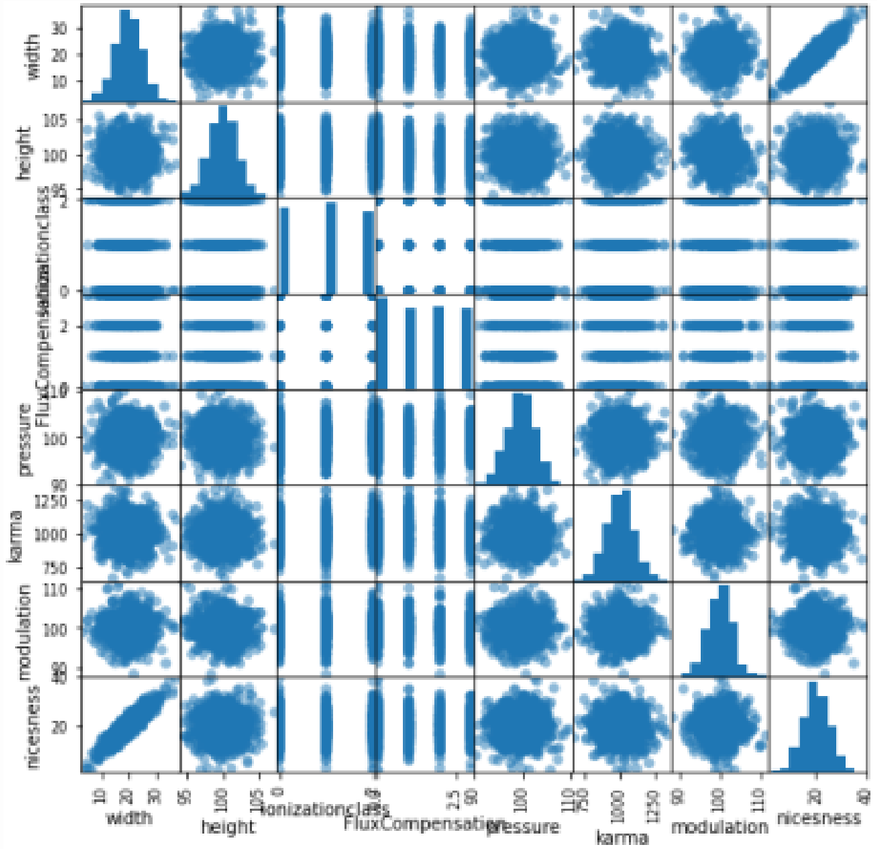


MODEL 2 - XGB REGRESSOR

MAIN INSIGHTS

Mean Abs. Error = 2.1072055 **Baseline error** = 4.1506495 ***XGBRegressor()***



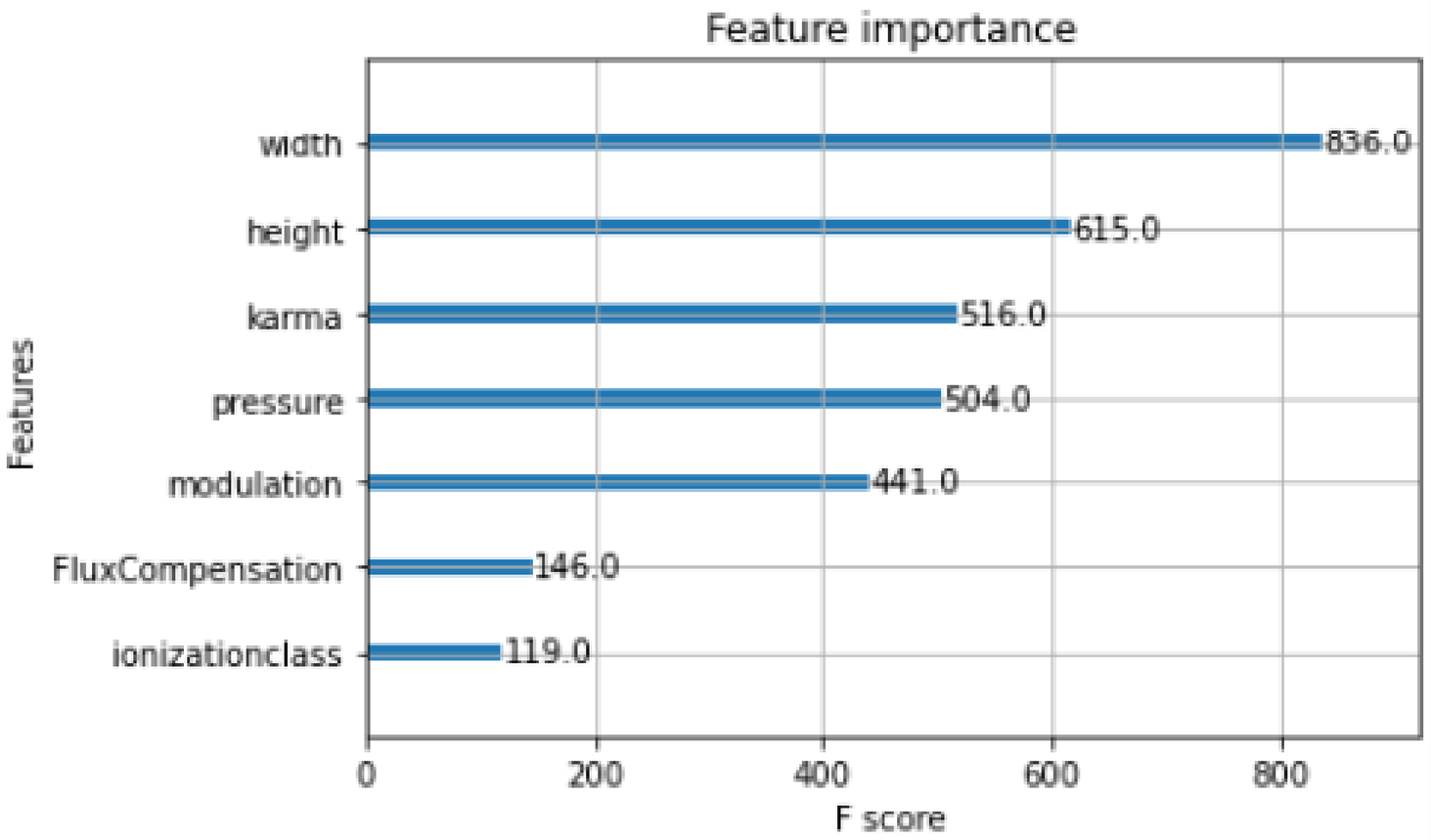


Feature Importance

A graph representing the importance of features.

The higher the importance of a feature, the more it influences the model's prediction

```
# FEATURE IMPORTANCE
XGB.PLOT_IMPORTANCE(MODEL)
```

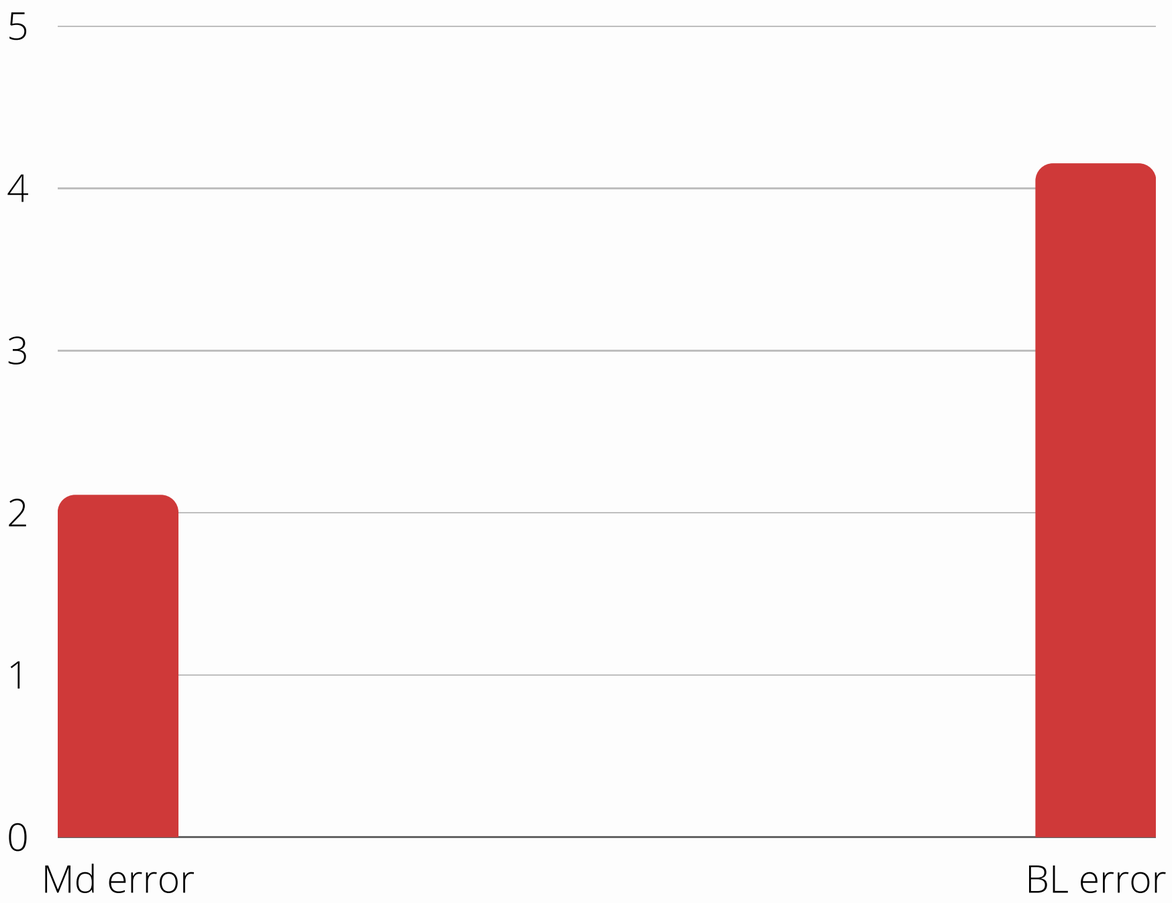


DATA COMPARISON

- Mean absolute error for Linear
  R = 2.10720550
- Baseline error = 4.1506495

Md = Model error
BL = Baseline error

BEST MODEL TRAIN

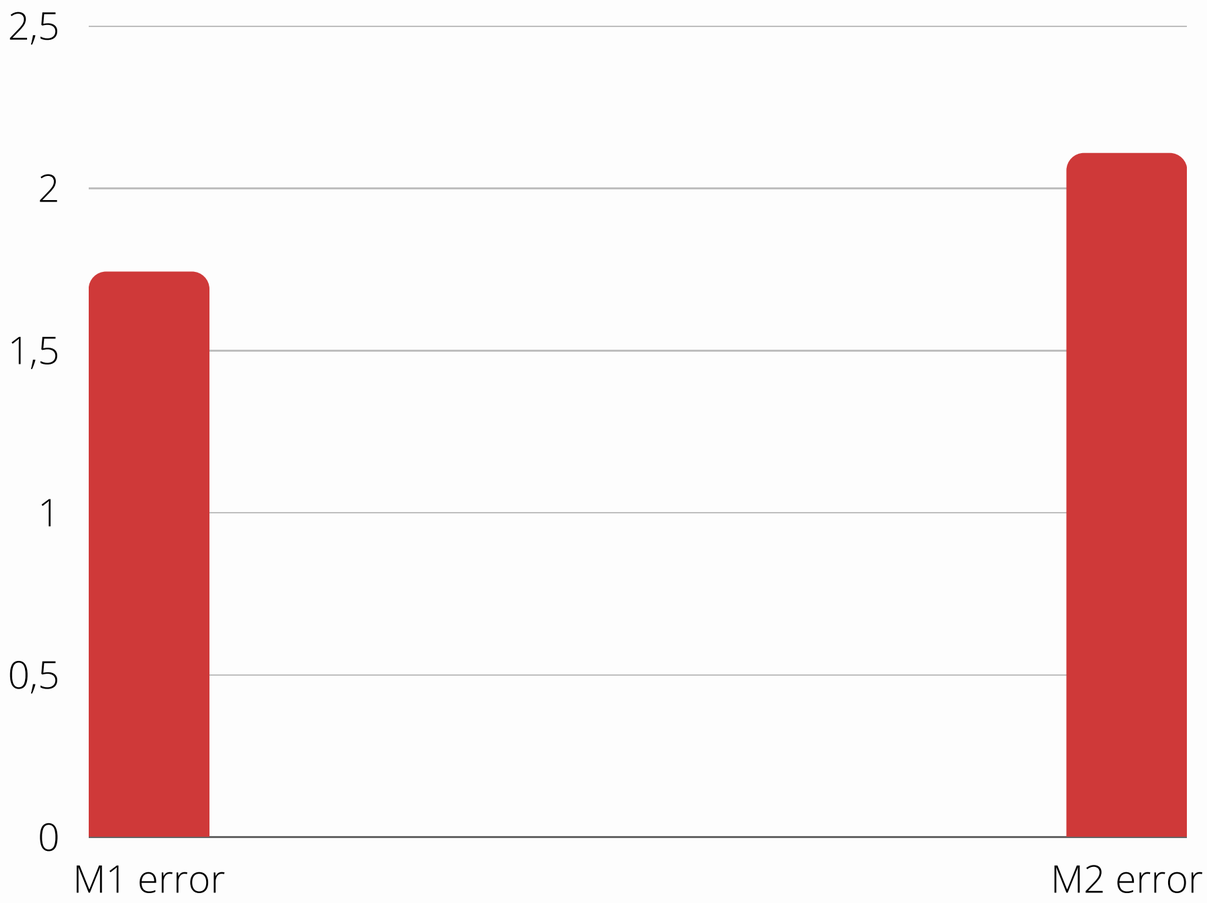


MODEL COMPARISON

- Mean absolute error for Linear Regression = 1.741689
- Mean absolute error for XGB Regressor = 2.1072055

M1 = Linear Regression
M2 = XGB Regressor

BEST MODEL
LINEAR REGRESSION



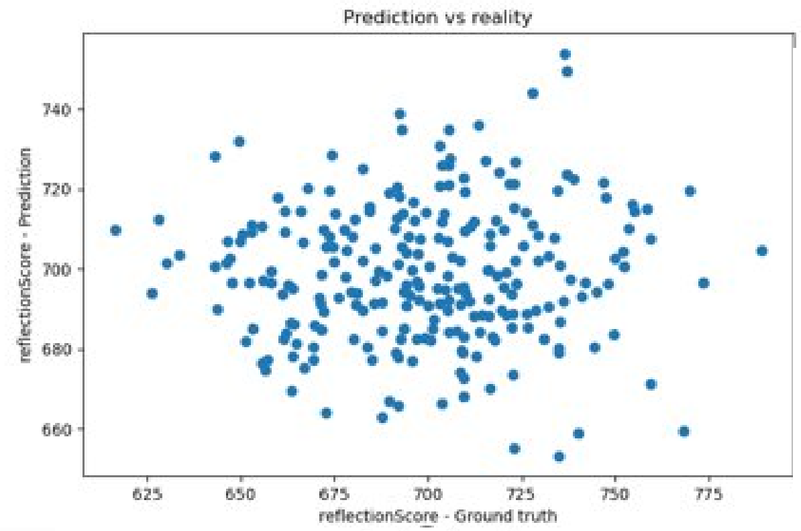
07

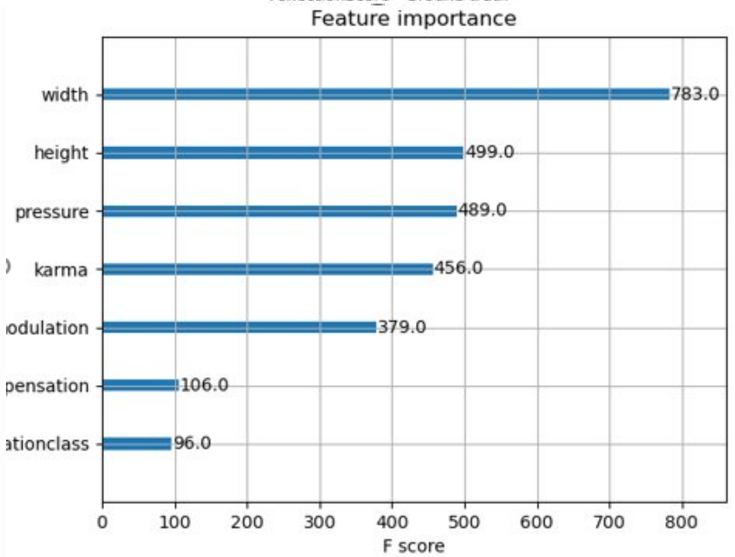
REFLECTION SCORE PREDICTION

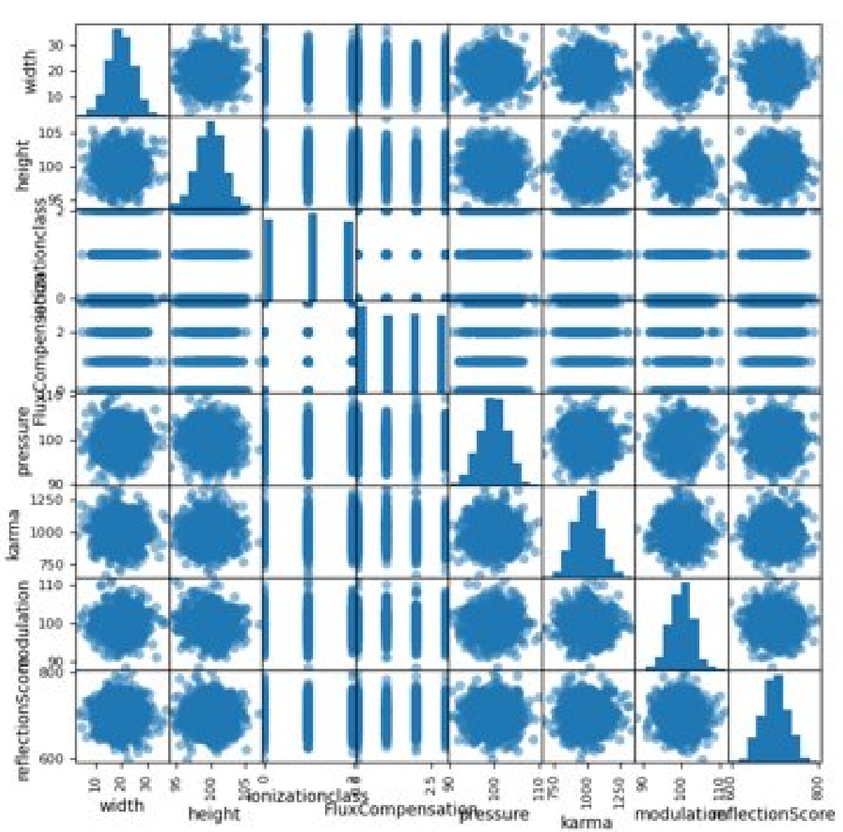
- Goal
- Models used
- Input and output features
- Model comparison and result

REFLECTIONSCORE PREDICTION

- Used models:
  - Poly1d (numpy)
  - XGBRegressor (xgboost)
  - LinearRegression (sklearn)
- XGB model error 28.26
- Linear model error: 24.84
- Poly1d model error: 24.88
- baseline error 24.88

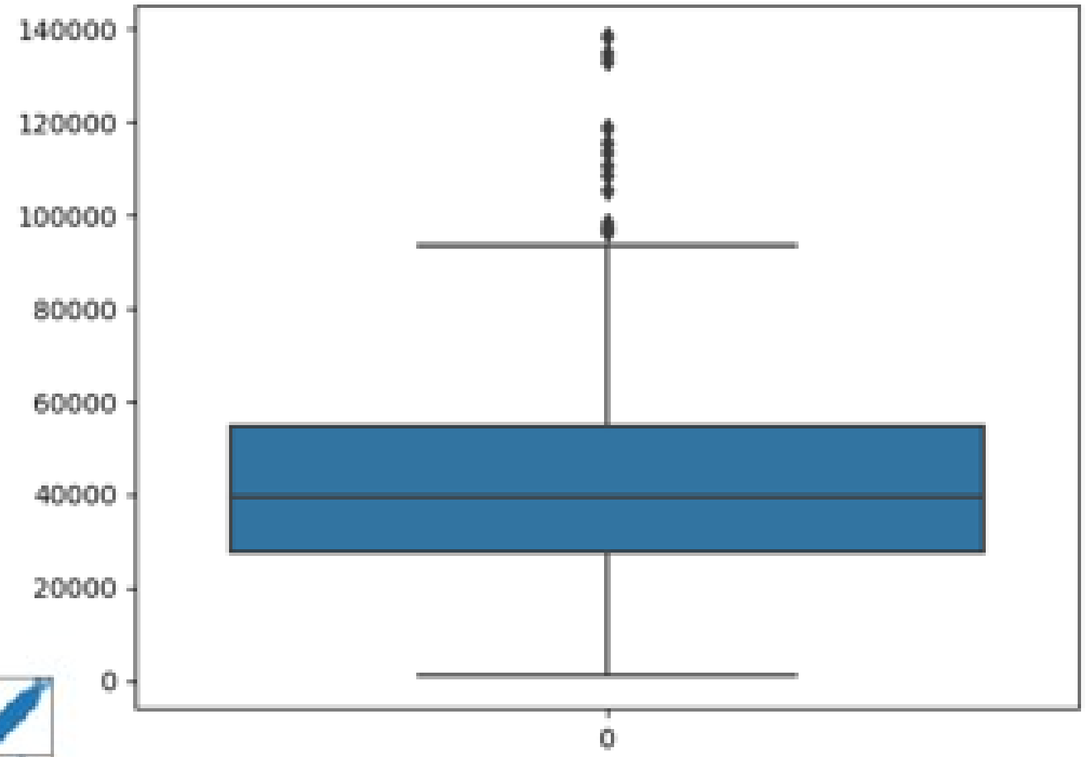


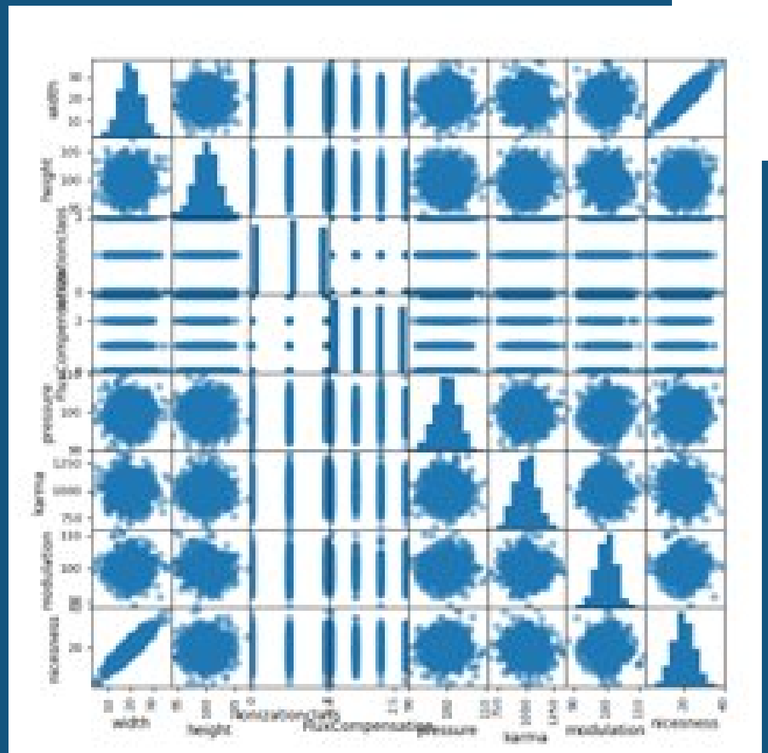




DETAILS, CODE, DATA PREP

```
pd.plotting.scatter_matrix(
df,figsize=(8,8),grid=True,
marker='o') # seeing
corelations

df.describe()

sns.boxplot(df['width']) #
seeing outliers

df.loc[df['width'] ==
10000000000,'width'] =
df.width.median() (just an
idea, unused)

df = df[df['width'] !=
10000000000] # removing
outliers
```





08

PHAT PREDICTIONS

09

REFLECTION ON RESULTS

- What value can we derive from your insights?
- Why was the analysis done in this particular way?
- What could be improved

RESULTS

Q1 weight_in_kg

Q2 error_type

Other features

What value can we derive from your insights?	Why was the analysis done in this particular way?	What could be improved
Some properties can be predicted at the input to optimize the production	The properties can be predicted at the input to optimize the production	Have more data to build more advanced models check the sensors, if they are working correctly

DISCUSSION

TIME

DETAILS, CODE, DATA PREP

outlier removal:
df = df[df['width'] != 10000000000]

Label encoding:

- labelEncoder, get_dummies

normalization:

- minMax, normalize

Neural Network models:

- possibly too little data
- for some features to complicated

MAE (MEAN ABSOLUTE ERROR)

- Before normalization 1.7416894630880386 average error on all data
- After normalization 0.002703266579790724 3 average error on all data

NN = No normalized
N = normalized

BEST NORMALIZED DATA