Projekt Andromeda

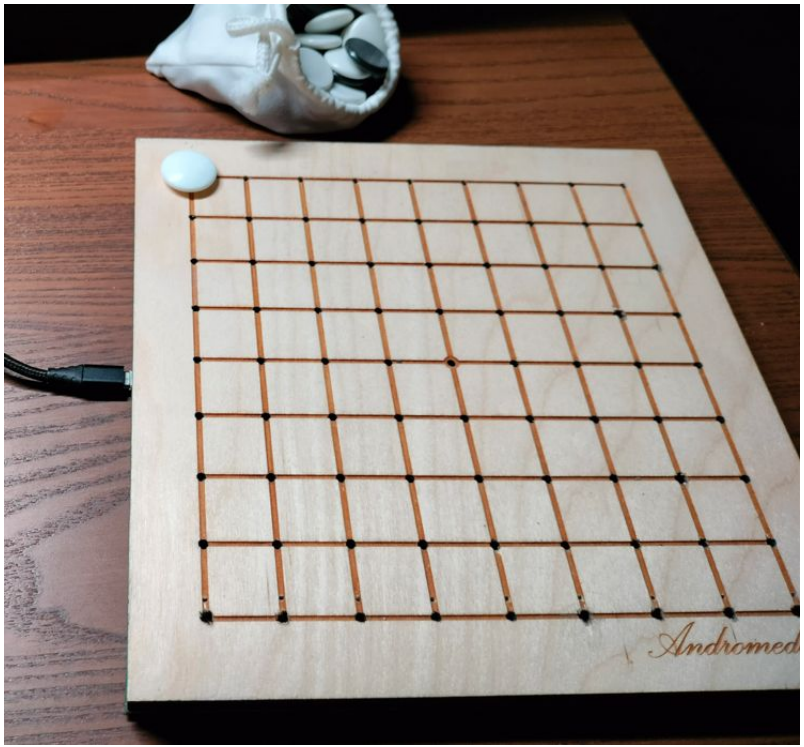
Automatyczna detekcja kamieni na planszy Go

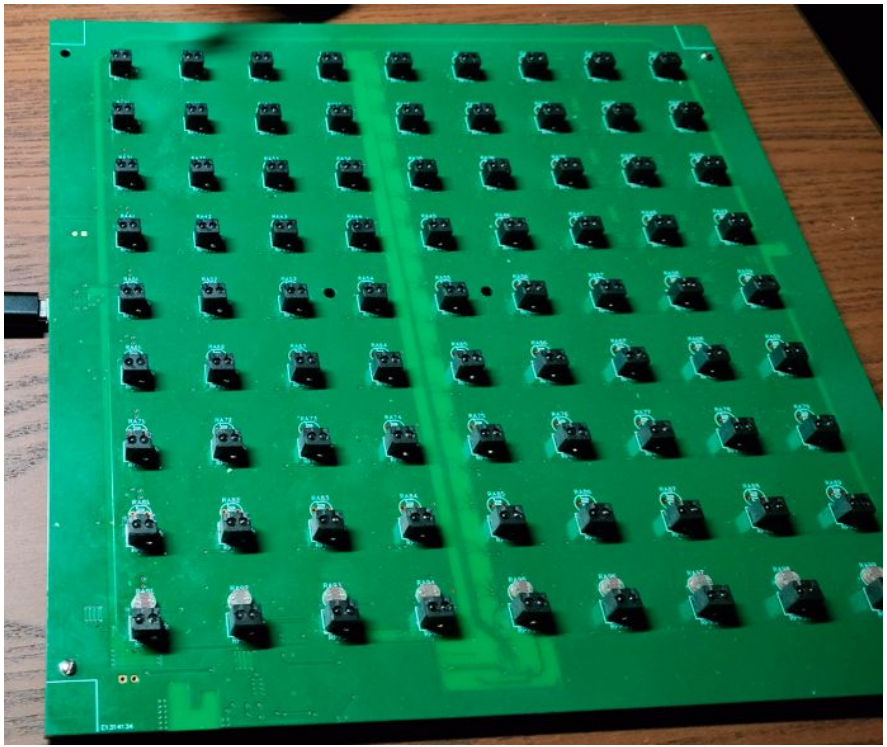
Opis projektu

Projekt składa się z kilku części:

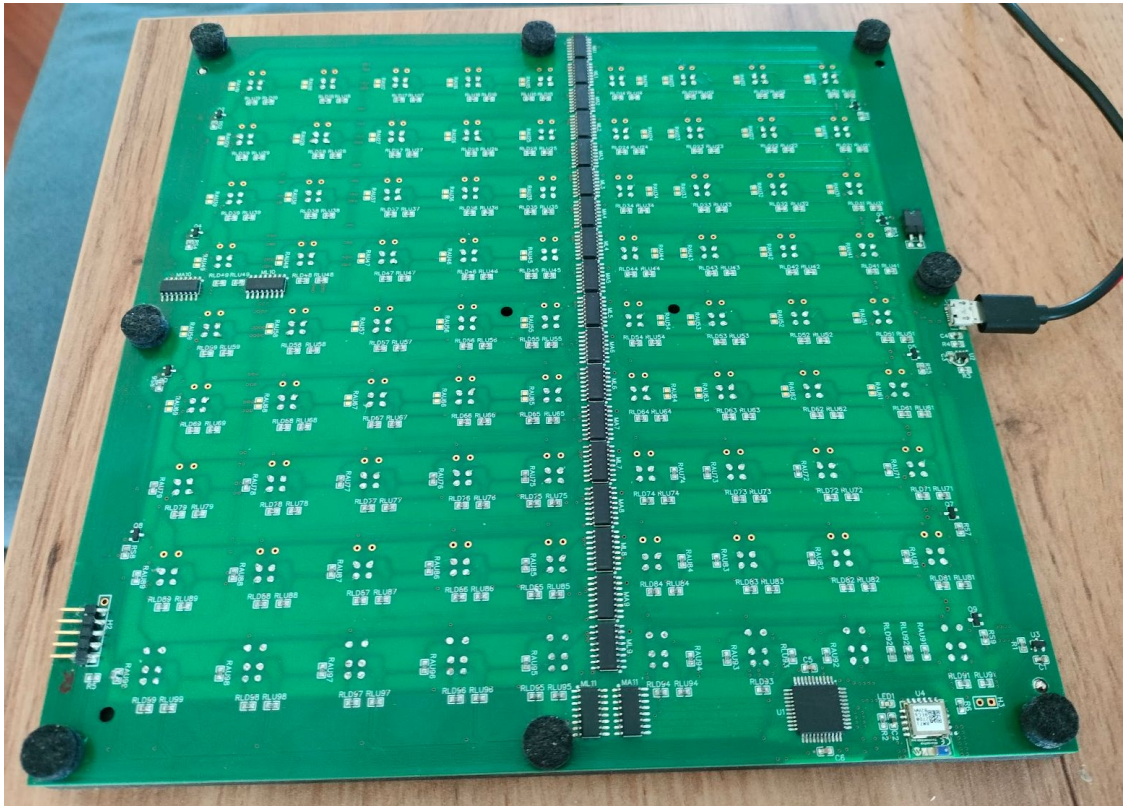
1) Elektroniczna plansza z sensorami na podczerwień oraz modułem Bluetooth:
2) Odbiornik Bluetooth przesyłający dane do portu COM
3) Aplikacja komputerowa wizualizująca dane z planszy
4) Analiza danych sensorycznych i trenowanie modeli ML w Google Colab

Elektroniczna plansza

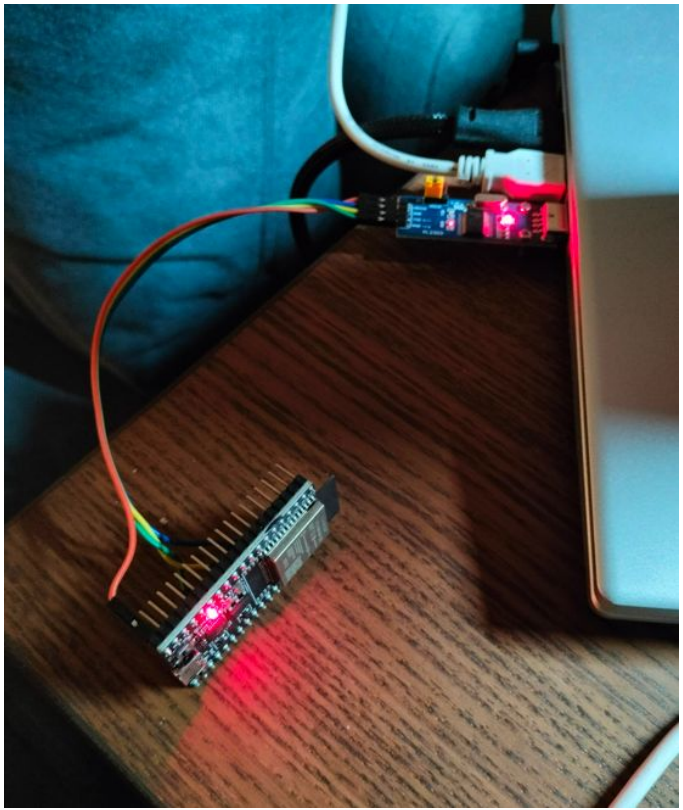




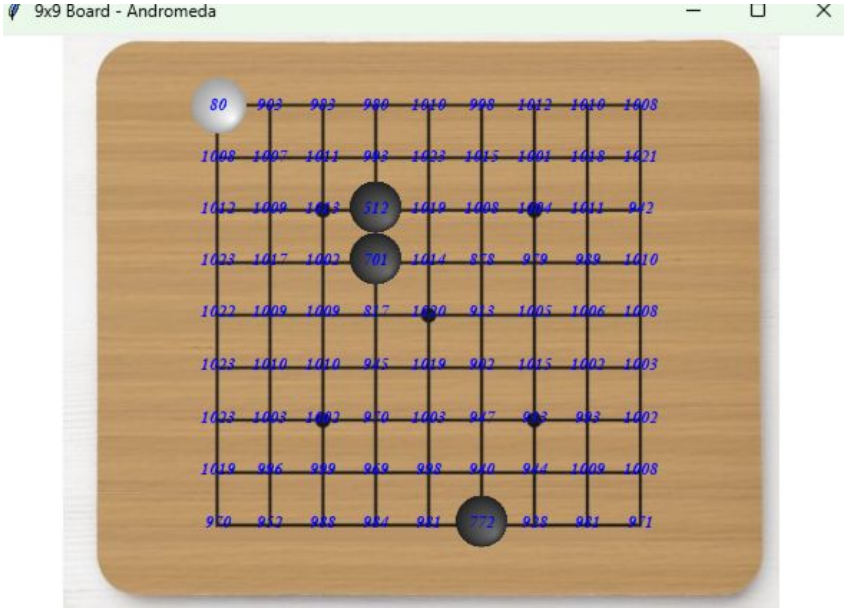
Elektroniczna plansza

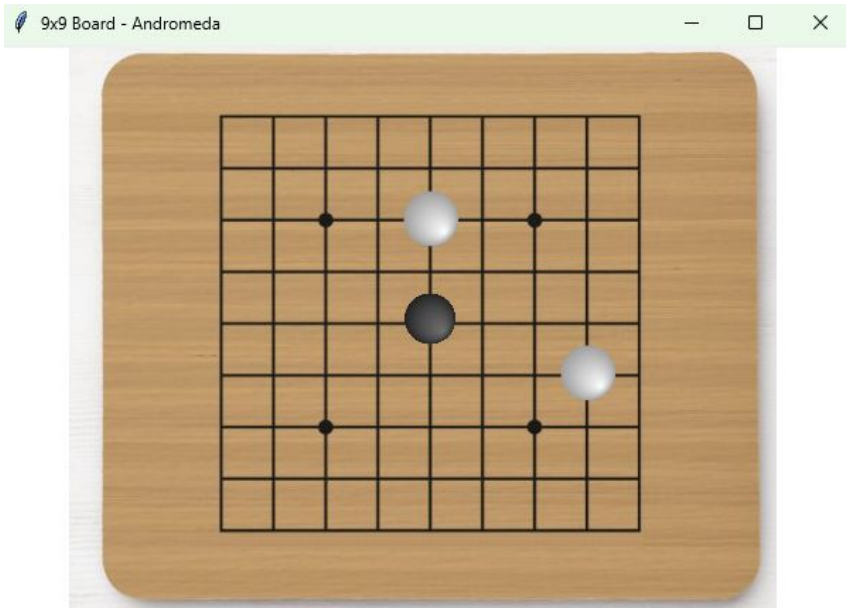


Odbiornik Bluetooth - ESP32



Aplikacja komputerowa





Dlaczego ML?

1) Model uczenia maszynowego służy głównie do automatyzacji procedury kalibracyjnej na podstawie pomiarów, można wyznaczyć metryki
2) Klasyczne modele są bardzo czasochłonne gdyż wymagają ręcznej kalibracji

Analiza danych i trenowanie modeli

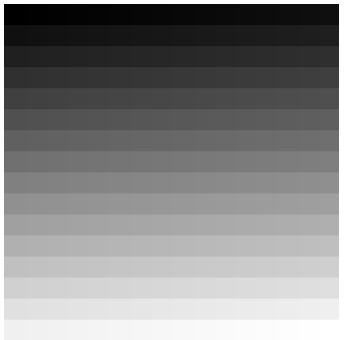
1) Model I - model regresji liniowej przetwarzający dane z sensorów na kolor w skali szarości
2) Model II - model klasyfikujący kolor w skali szarości do jednej z 3 grup: kamień biały, kamień czarny i brak kamienia

Model I - regresja liniowa

1. W celu określenia koloru kamienia potrzebny jest model, który określi jego kolor w skali szarości

   Dane wejściowe:

- odczyty z sensora nr 0 czytającego wydrukowany kolor od 0 do 130 - powyżej tej wartości sensor nie rozpoznaje



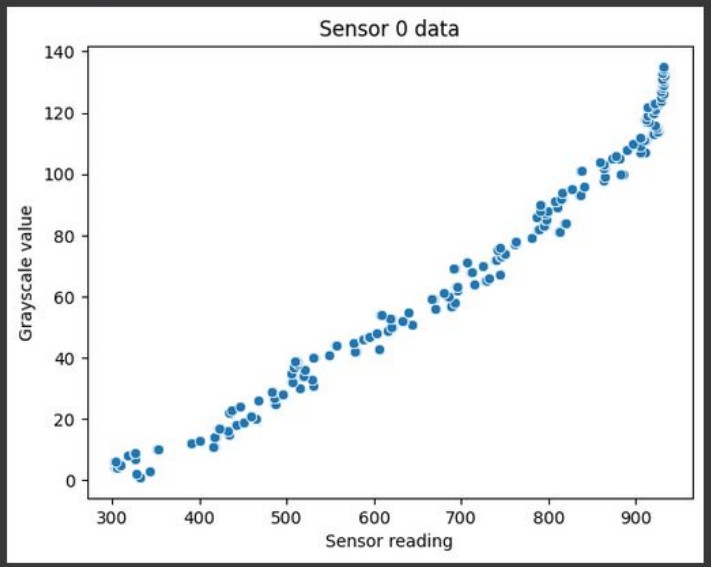
Data cleaning

- Data cleaning ograniczył się tylko do usunięcia pomiarów które były nieprawidłowo wykonane

```
[14] # Drop rows where the measurement was done incorrectly
     df_stones = df_stones[df_stones['greyscale_value_0'] != 80]
     df_stones = df_stones[df_stones['greyscale_value_0'] != 97]
     df_stones = df_stones[df_stones['greyscale_value_0'] != 0]
```

Data visualization

```
sns.scatterplot(x = '0', y = 'greyscale_value_0', data = df_stones);

# Set title and axis labels
plt.title('Sensor 0 data')
plt.xlabel('Sensor reading')
plt.ylabel('Grayscale value')

# Show the plot
plt.show()
```



Preparing data for training

Regression

```
# Sklearn
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import PolynomialFeatures
from sklearn.model_selection import GridSearchCV
from sklearn.pipeline import Pipeline

X = df_stones[['0']]
y = df_stones[['greyscale_value_0']]
```

Train, test, validation split

```
[ ] X_train, X_test, y_train, y_test = train_test_split(X,
                                                        y,
                                                        test_size=0.2,
                                                        random_state=42)
```

Model training

```
pipe = Pipeline([('sc', StandardScaler()),
                 ('poly', PolynomialFeatures()),
                 ('lr', LinearRegression())
                 ])
params = {'poly__degree': np.arange(10)}
gs = GridSearchCV(pipe,
                  param_grid = params,
                  cv = 10,
                  verbose = 3,
                  return_train_score=True,
                  refit=True
)
gs.fit(X_train, y_train)
```

```
[ ] gs.score(X_train, y_train)
```

```
0.9930577963889095
```

Model validation

```
[ ] gs.score(X_test, y_test)
```

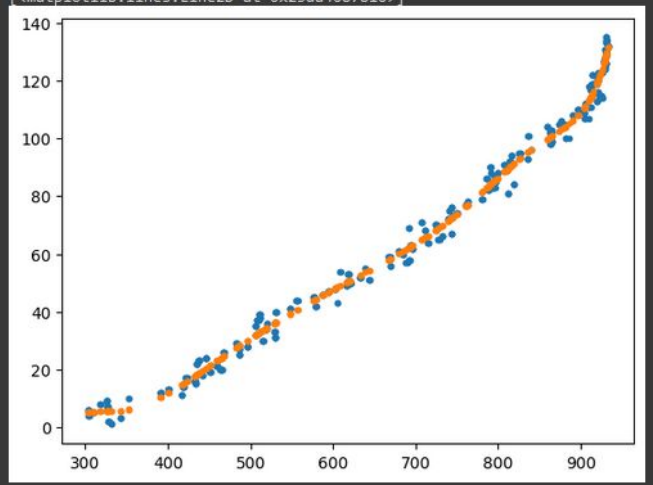
```
0.9931734082087877
```

```
[ ] gs.best_params_
```

```
{'poly__degree': 9}
```

```
[ ] plt.plot(X, y, '.')
    plt.plot(X_train, gs.predict(X_train), '.')
```

```
[<matplotlib.lines.Line2D at 0x23da4087810>]
```



Model export

```
import joblib
joblib.dump(gs.best_estimator_, 'ADC_to_color.pkl')
```

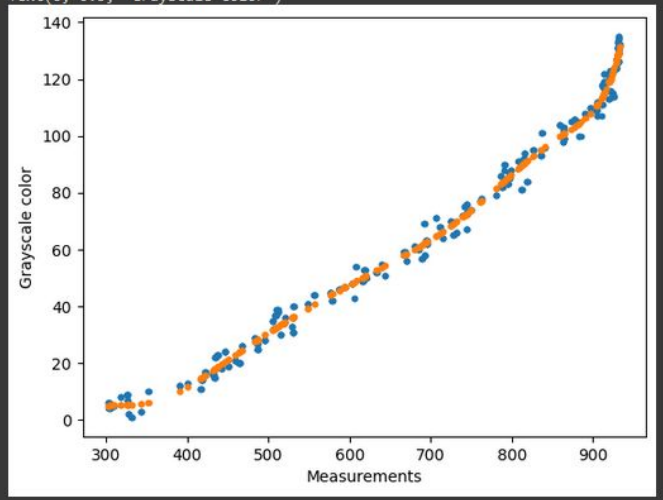
```
['ADC_to_color.pkl']
```

Model import on local machine

```
[ ] gs = joblib.load("ADC_to_color.pkl")
```

```
plt.plot(X, y, '.')
plt.plot(X_train, gs.predict(X_train), '.')
plt.xlabel("Measurements")
plt.ylabel("Grayscale color")
```

```
Text(0, 0.5, 'Grayscale color')
```




```
[ ] print(gs.predict(pd.DataFrame([600])).astype(int)[0][0])
```

```
47
```

Model II - klasyfikacja na podstawie koloru

1. Zebrano dane z 4 sensorów przykładając kamienie o różnej jakości, różnym kształcie i w różnym położeniu nad sensorem
2. Zbudowano model na podstawie danych z jednego sensora

Import data

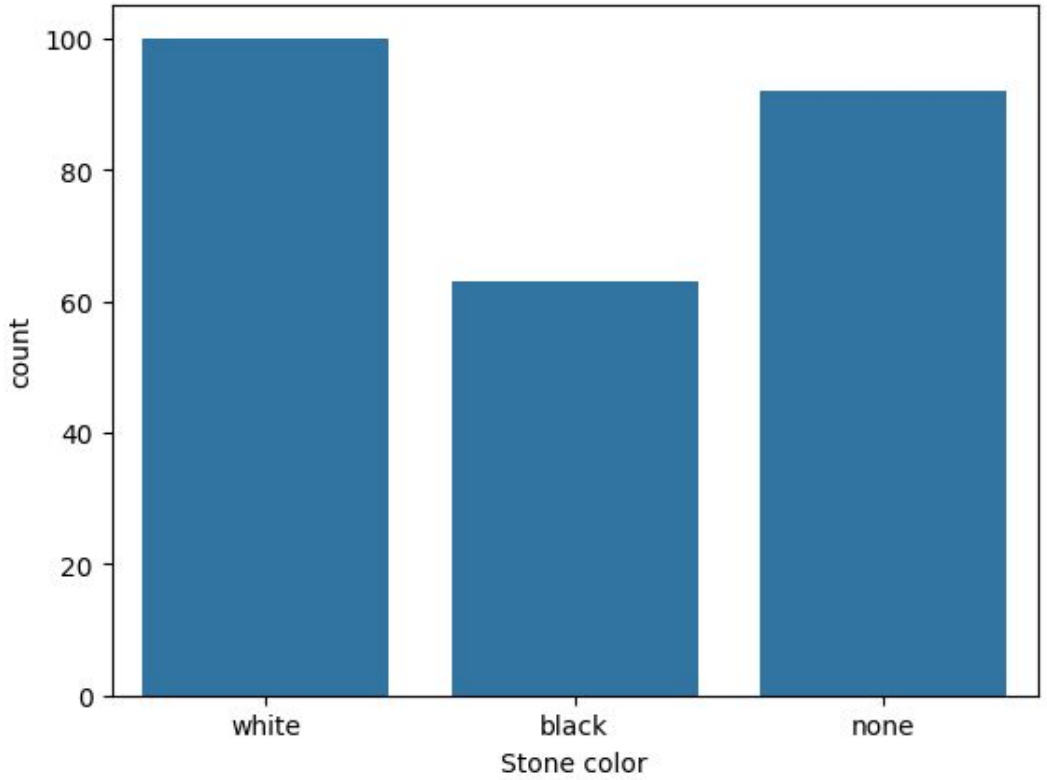
```
[ ]  df_stones = pd.DataFrame()
     nr_of_files = 4
     sensors_saved = ['0', '2', '18', '20']
     sensors_saved_2 = ['0']
     colors_saved = [str(f"color_{i}") for i in sensors_saved]

     for file_nr in range(1, nr_of_files + 1):
         df = pd.read_csv(f"stone_classification_4_sensors_{file_nr}.csv", index_col='Unnamed: 0')
         df_stones = pd.concat([df_stones, df], ignore_index=True)

     df_stones = df_stones[sensors_saved+colors_saved]
     df_stones= df_stones.replace({"w": "white", "b": "black", "n": "none"})
```

Data cleaning

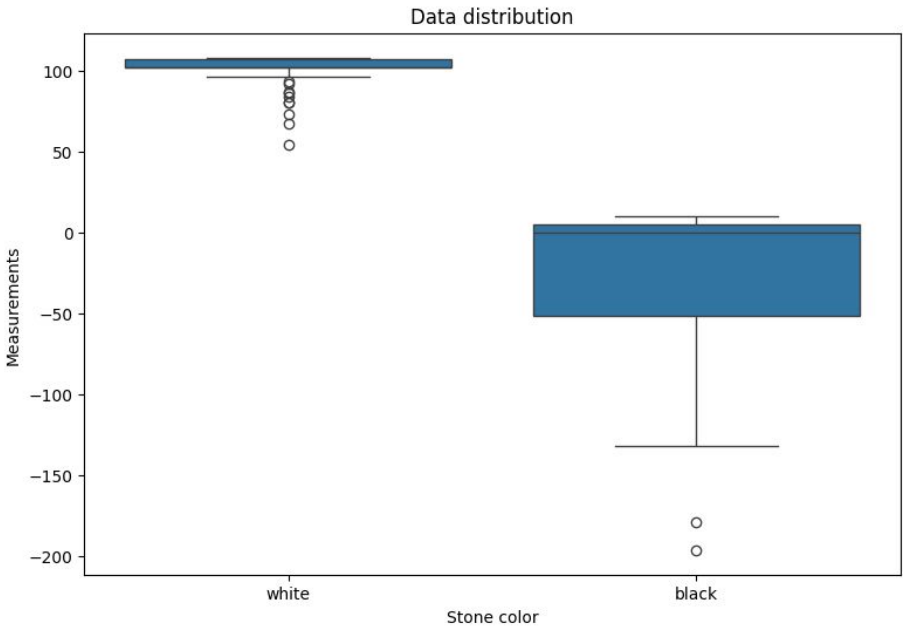
- Data cleaning polegał na odrzuceniu pomiarów, które zostały wykonane dla kamieni znajdujących się w dużej odległości od środka sensora

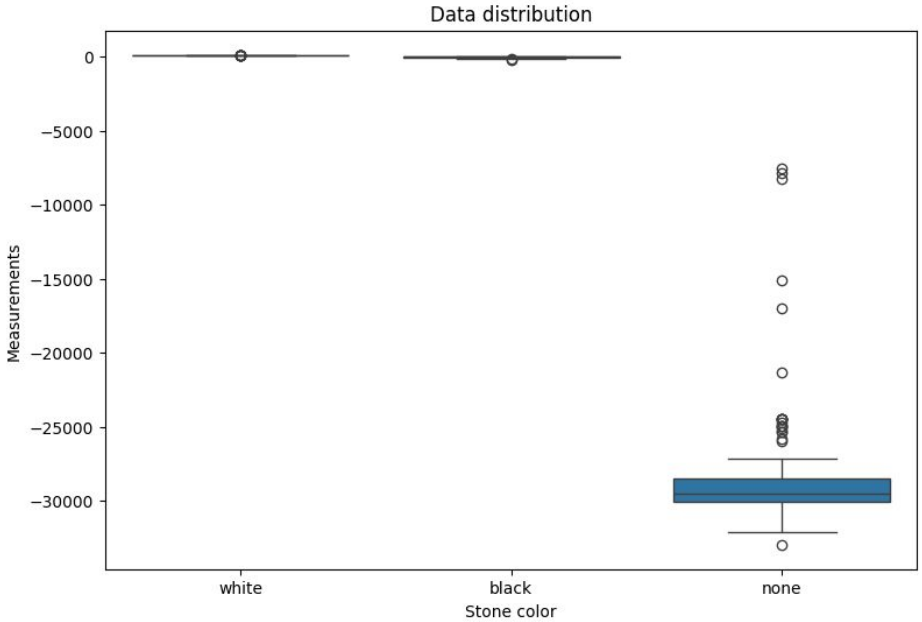
```
df_2_stones = df_stones[['0', 'color_0']]
df_2_stones = df_2_stones.loc[(df_2_stones['color_0'] != 'black') | (df_2_stones['0'] > -200)]
df_2_stones = df_2_stones.loc[(df_2_stones['color_0'] != 'none') | (df_2_stones['0'] < -5000)]
```

Data exploration



Data exploration





Preparing data for training

```
from sklearn.model_selection import train_test_split
X = df_2_stones[['0']]
y = df_2_stones['color_0']
X_train, X_test, y_train, y_test = train_test_split(X,
                                                    y,
                                                    test_size=0.20,
                                                    random_state=42,
                                                    stratify = y)
```

```
from sklearn.preprocessing import PolynomialFeatures
from sklearn.pipeline import Pipeline
from sklearn.linear_model import LogisticRegression
from sklearn.preprocessing import StandardScaler
```

Model training - logistic regression

```
params = {'poly__degree': np.arange(3),
          'lr__C': np.logspace(-3, 0, 10),
          'lr__penalty': ['l1', 'l2']
          }

gs = GridSearchCV(pipe,
                  param_grid = params,
                  cv = 10,
                  verbose = 3,
                  return_train_score=True,
                  n_jobs=2,
                  scoring='accuracy'
)
gs.fit(X_train, y_train)
```

	precision	recall	f1-score	support
black	0.00	0.00	0.00	50.0
misplaced	0.00	0.00	0.00	0.0
none	0.00	0.00	0.00	74.0
white	0.00	0.00	0.00	80.0
accuracy			0.00	204.0
macro avg	0.00	0.00	0.00	204.0
weighted avg	0.00	0.00	0.00	204.0

```
y_train_pred_class = pd.DataFrame(y_train_pred).value_counts()
y_train_pred_class
```

```
misplaced    204
dtype: int64
```

Model training - random forest

```
rfc = RandomForestClassifier()
param_grid = {
    "max_depth": np.arange(2,5),
    "min_samples_split": np.arange(2,10),
    "min_samples_leaf": np.arange(2,10),
    "n_estimators": [10, 20, 30]
}

gs = GridSearchCV(rfc,
                  param_grid,
                  cv = 10,
                  verbose = 4,
                  n_jobs=2,
                  scoring='accuracy'
)
gs.fit(X_train, y_train)
```

	precision	recall	f1-score	support
black	0.98	1.00	0.99	50
none	1.00	0.99	0.99	74
white	1.00	1.00	1.00	80
accuracy			1.00	204
macro avg	0.99	1.00	0.99	204
weighted avg	1.00	1.00	1.00	204

```
white        80
none         73
black        50
misplaced     1
dtype: int64
```

Model test - random forest

```
y_test_pred = gs.predict_proba(X_test)
y_test_pred = predict_stone(y_test_pred, 0.8, X_test)
print(classification_report(y_test, y_test_pred))
```

	precision	recall	f1-score	support
black	1.00	1.00	1.00	13
misplaced	0.00	0.00	0.00	0
none	1.00	0.89	0.94	18
white	1.00	1.00	1.00	20
accuracy			0.96	51
macro avg	0.75	0.72	0.74	51
weighted avg	1.00	0.96	0.98	51

```
white        20
none         16
black        13
misplaced     2
```