PD3

Artur Żółkowski

Wczytanie i zapoznanie się z danymi

In [22]:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import matplotlib
import seaborn as sns
```

In [3]:

```
df = pd.read_csv('australia.csv')
df
```

Out[3]:

	MinTemp	MaxTemp	Rainfall	Evaporation	Sunshine	WindGustSpeed	WindSpeed9am	\
0	17.9	35.2	0.0	12.0	12.3	48.0	6.0	
1	18.4	28.9	0.0	14.8	13.0	37.0	19.0	
2	19.4	37.6	0.0	10.8	10.6	46.0	30.0	
3	21.9	38.4	0.0	11.4	12.2	31.0	6.0	
4	24.2	41.0	0.0	11.2	8.4	35.0	17.0	
...	...	...	...	...	...	...	...	
56415	19.3	33.4	0.0	6.0	11.0	35.0	9.0	
56416	21.2	32.6	0.0	7.6	8.6	37.0	13.0	
56417	20.7	32.8	0.0	5.6	11.0	33.0	17.0	
56418	19.5	31.8	0.0	6.2	10.6	26.0	9.0	
56419	20.2	31.7	0.0	5.6	10.7	30.0	15.0	

56420 rows × 18 columns

In [5]:

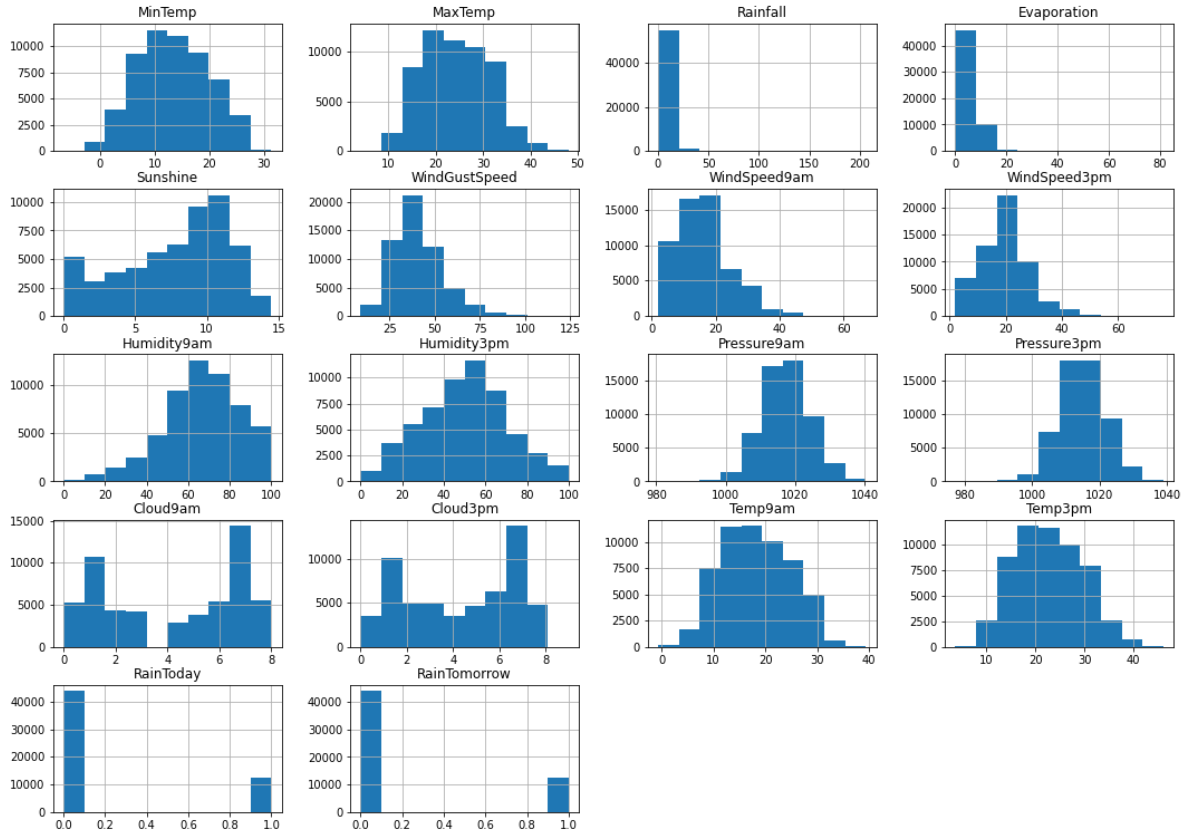
```
df.describe()
```

Out[5]:

	MinTemp	MaxTemp	Rainfall	Evaporation	Sunshine	WindGustSpeed
count	56420.000000	56420.000000	56420.000000	56420.000000	56420.000000	56420.000000
mean	13.464770	24.219206	2.130397	5.503135	7.735626	40.877366
std	6.416689	6.970676	7.014822	3.696282	3.758153	13.335232
min	-6.700000	4.100000	0.000000	0.000000	0.000000	9.000000
25%	8.600000	18.700000	0.000000	2.800000	5.000000	31.000000
50%	13.200000	23.900000	0.000000	5.000000	8.600000	39.000000
75%	18.400000	29.700000	0.600000	7.400000	10.700000	48.000000
max	31.400000	48.100000	206.200000	81.200000	14.500000	124.000000

In [4]:

```
df.hist(figsize=(18,13))
plt.show()
```



In [6]:

```
y = df['RainTomorrow']
X = df.drop(['RainTomorrow'],axis=1)
```

In [7]:

```
y.value_counts()
```

Out[7]:

```
0    43993
1    12427
Name: RainTomorrow, dtype: int64
```

Stworzenie i porównanie modeli

In [8]:

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42, stratify=y)
```

In [9]:

```
from sklearn.model_selection import KFold, cross_validate
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler

from sklearn.ensemble import RandomForestClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
```

In [10]:

```
kfolds = 10
split = KFold(n_splits=kfolds, shuffle=True, random_state=42)
```

Random Forest

In [11]:

```
rf = RandomForestClassifier(max_depth=30, n_estimators=300, max_features='sqrt', n_jobs = -

rf_model = Pipeline(steps=[('standardscaler', StandardScaler()), ('rf', rf)])

rf_cv_results = cross_validate(rf_model,
                X_train, y_train,
                cv=split,
                scoring=["accuracy", "f1", "roc_auc"], n_jobs=-1)
```

Logistic Regression

In [12]:

```
lr = LogisticRegression(penalty='elasticnet', solver='saga', l1_ratio=0.8)

lr_model = Pipeline(steps=[('standardscaler', StandardScaler()), ('lr', lr)])

lr_cv_results = cross_validate(lr_model,
                  X_train, y_train,
                  cv=split,
                  scoring=["accuracy", "f1", "roc_auc"], n_jobs=-1)
```

K neighbors

In [13]:

```
knn = KNeighborsClassifier(n_neighbors=10, weights='distance', p=1.5, n_jobs=-1)

knn_model = Pipeline(steps=[('standardscaler', StandardScaler()), ('knn', knn)])

knn_cv_results = cross_validate(knn_model,
                  X_train, y_train,
                  cv=split,
                  scoring=["accuracy", "f1", "roc_auc"], n_jobs=-1)
```

In [15]:

```
rf_df = pd.DataFrame({"model": "RF", "accuracy_score": rf_cv_results["test_accuracy"],
                      "f1_score": rf_cv_results["test_f1"], "roc_auc_score": rf_cv_results[
lr_df = pd.DataFrame({"model": "LR", "accuracy_score": lr_cv_results["test_accuracy"],
                      "f1_score": lr_cv_results["test_f1"], "roc_auc_score": lr_cv_results[
knn_df = pd.DataFrame({"model": "KNN", "accuracy_score": knn_cv_results["test_accuracy"],
                      "f1_score": knn_cv_results["test_f1"], "roc_auc_score": knn_cv_result
results = pd.concat([rf_df, lr_df, knn_df], axis=0)
```

In [17]:

```
results_mean = results.groupby('model').mean()
```

In [19]:

```
results
```

Out[19]:

	model	accuracy_score	f1_score	roc_auc_score
0	RF	0.853567	0.599637	0.881945
1	RF	0.854896	0.626355	0.896401
2	RF	0.856447	0.612903	0.897030
3	RF	0.859105	0.621879	0.891463
4	RF	0.865308	0.638526	0.896185
5	RF	0.861542	0.635993	0.894998
6	RF	0.873477	0.646440	0.904780
7	RF	0.857744	0.615569	0.886929
8	RF	0.863062	0.649660	0.899896
9	RF	0.865278	0.648148	0.898659
0	LR	0.846256	0.588375	0.877635
1	LR	0.847364	0.614869	0.887565
2	LR	0.851573	0.603081	0.885843
3	LR	0.850244	0.607433	0.877636
4	LR	0.855339	0.618352	0.879639
5	LR	0.848250	0.606999	0.883464
6	LR	0.866386	0.632093	0.895429
7	LR	0.851540	0.604019	0.878044
8	LR	0.856636	0.636721	0.889502
9	LR	0.858631	0.642777	0.888723
0	KNN	0.841825	0.576010	0.853694
1	KNN	0.844484	0.598857	0.868353
2	KNN	0.844041	0.579450	0.862539
3	KNN	0.850244	0.598575	0.864934
4	KNN	0.854453	0.610089	0.866576
5	KNN	0.852459	0.611435	0.873313
6	KNN	0.862619	0.617284	0.871259
7	KNN	0.849324	0.586877	0.851466
8	KNN	0.848881	0.612059	0.866928
9	KNN	0.846665	0.605473	0.866189

In [18]:

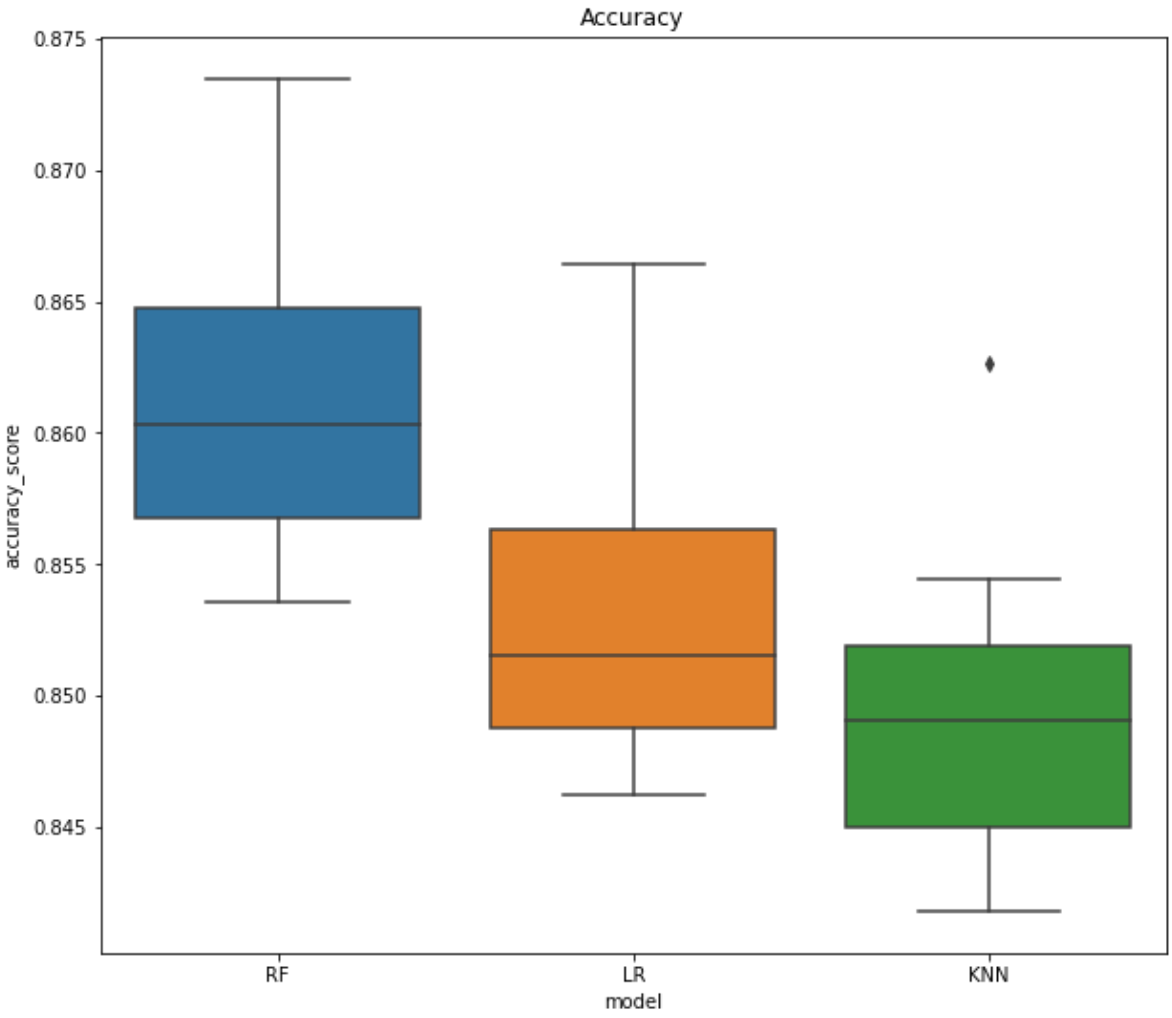
```
results_mean
```

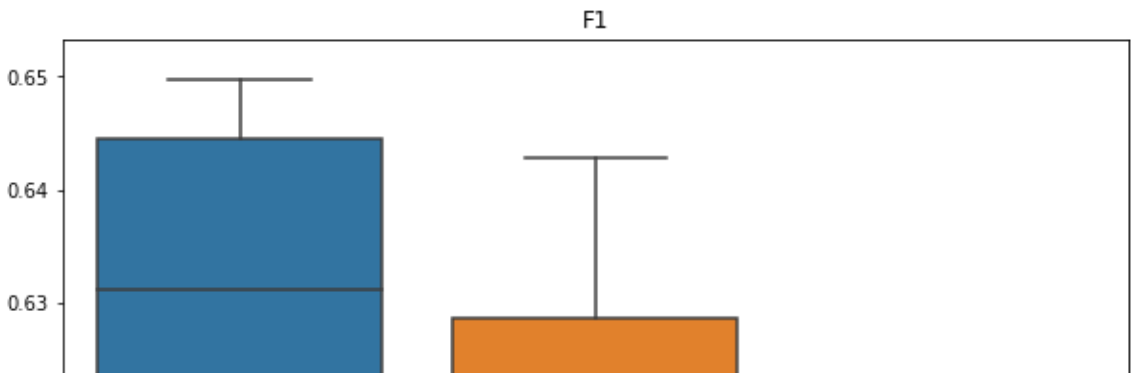
Out[18]:

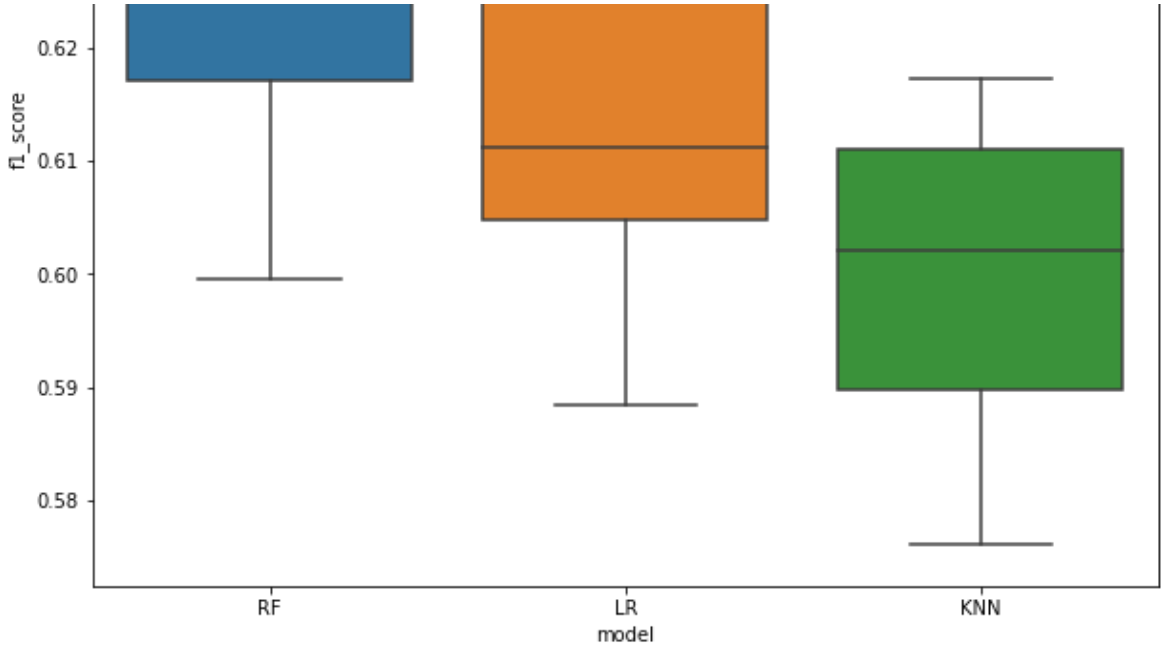
model	accuracy_score	f1_score	roc_auc_score
KNN	0.849500	0.599611	0.864525
LR	0.853222	0.615472	0.884348
RF	0.861043	0.629511	0.894829

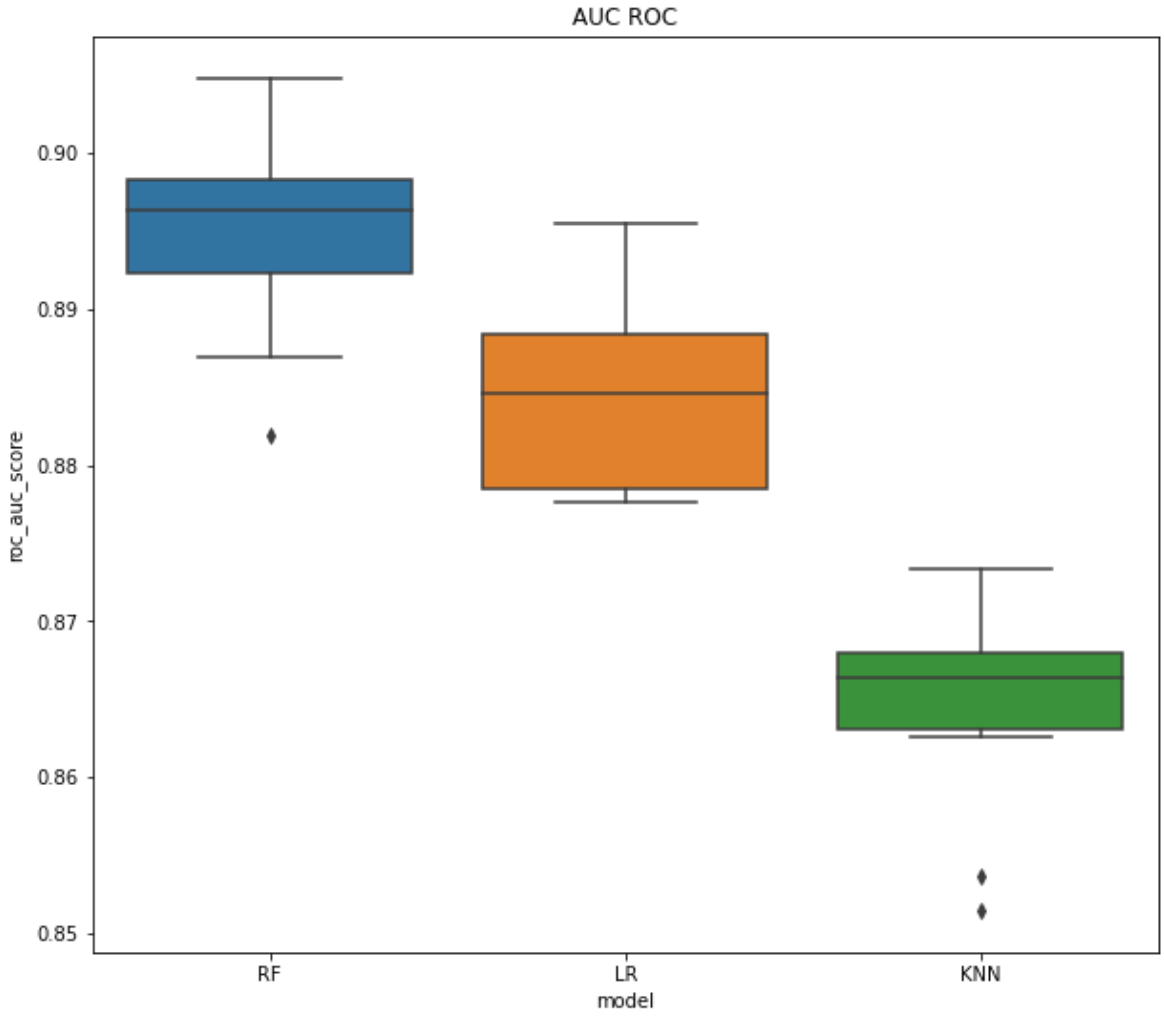
In [26]:

```
plt.subplots(figsize=(10,30))

plt.subplot(3, 1, 1)
sns.boxplot(data = results, x = "model", y = "accuracy_score")
plt.title('Accuracy')

plt.subplot(3, 1, 2)
sns.boxplot(data = results, x = "model", y = "f1_score")
plt.title('F1')

plt.subplot(3, 1, 3)
sns.boxplot(data = results, x = "model", y = "roc_auc_score")
plt.title('AUC ROC')

plt.show()
```









Wnioski

Najlepszym z przeanalizowanych modeli okazał się las losowy, u którego każda z miar jest zauważalnie lepsza. Analizując miary należy jednak pamiętać jak one działają i np. w przypadku accuracy model przewidujący, że nigdy nie będzie padać miałby 80% acc. Pamiętajmy również, że wyniki modelu można zmieniać dopasowując odpowiednio hiperparametry, tak by maksymalizować miarę oceny, na której nam najbardziej zależy.