Analiza wyników na Ojro w zależności od składu od czasu kiedy ja bywałem na ojro:

In [139]:
```
print("""Dane na których operuję są postaci:
Kapitan, Miejsce, Rok, Lokacja, gracz1, gracz2, gracz3, gracz4, gracz5, gracz6, gracz7, gracz8
Skark, 5, 2012, Gorzow, Vlad, Piszczu, Zly, Skark, Thurion, Marios, Raca, Romek
Vlad, 3, 2012, Serbia, Vlad, Michu, Karkowa, Typhus, Skark, Zly, Romek, Raca
Vlad, 1, 2014, Serbia, Typhus, Nathaniel, Raca, Ken, Vlad, Zly, Piszczu, Skark
Ken, 3, 2015, Praga, Typhus, Olek, Vlad, Ken, Skark, Zly, Raca, Piszczu
Vlad, 5, 2016, Ateny, Typhus, Romek, Skark, Vlad, Grzelich, Lesiu, Duda, Swistak
Typhus, 2, 2017, Hiszpania, Typhus, Skark, Vlad, Lesiu, Duda, Kacper, Zozo, Grzelich
Duda, 6, 2018, Chorwacja, Shizi, Kacper, Duda, Grzelich, Rudy, Dejw, Octos, Skark
Kacper, 3, 2019, Serbia, Lesiu, Michulec, Swistak, Shizi, Duda, Skark, Vlad, Karol""")
```

```
Dane na których operuję są postaci:
Kapitan, Miejsce, Rok, Lokacja, gracz1, gracz2, gracz3, gracz4, gracz5, gracz6, gracz7, gracz8
Skark, 5, 2012, Gorzow, Vlad, Piszczu, Zly, Skark, Thurion, Marios, Raca, Romek
Vlad, 3, 2012, Serbia, Vlad, Michu, Karkowa, Typhus, Skark, Zly, Romek, Raca
Vlad, 1, 2014, Serbia, Typhus, Nathaniel, Raca, Ken, Vlad, Zly, Piszczu, Skark
Ken, 3, 2015, Praga, Typhus, Olek, Vlad, Ken, Skark, Zly, Raca, Piszczu
Vlad, 5, 2016, Ateny, Typhus, Romek, Skark, Vlad, Grzelich, Lesiu, Duda, Swistak
Typhus, 2, 2017, Hiszpania, Typhus, Skark, Vlad, Lesiu, Duda, Kacper, Zozo, Grzelich
Duda, 6, 2018, Chorwacja, Shizi, Kacper, Duda, Grzelich, Rudy, Dejw, Octos, Skark
Kacper, 3, 2019, Serbia, Lesiu, Michulec, Swistak, Shizi, Duda, Skark, Vlad, Karol
```

In [123]:
```
import pandas as pd
import numpy as np
from sklearn.preprocessing import OneHotEncoder
from sklearn.ensemble import RandomForestClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
```

In [ ]:
```
wyniki = pd.read_csv(r"TAJEMNICA", sep = ", ")
wyniki.drop(["Rok", "Lokacja"], axis = 1, inplace = True)
```

Rozważmy problem repry w top3 albo nie w top3. Uznajmy za wydarzenie, które chcemy opisać, to że repka nie była w top3, więc wagi będą informować co na to wpłynęło.

In [113]:
```
logika = {5: 1, 6: 1, 1: 0, 2: 0, 3: 0}

wyniki["Miejsce"] = wyniki["Miejsce"].map(logika)
```

In [114]:
```
wyniki
```

Out[114]:

	Kapitan	Miejsce	gracz1	gracz2	gracz3	gracz4	gracz5	gracz6	gracz7	gracz8
0	Skark	1	Vlad	Piszczu	Zly	Skark	Thurion	Marios	Raca	Romek
1	Vlad	0	Vlad	Michu	Karkowa	Typhus	Skark	Zly	Romek	Raca
2	Vlad	0	Typhus	Nathaniel	Raca	Ken	Vlad	Zly	Piszczu	Skark
3	Ken	0	Typhus	Olek	Vlad	Ken	Skark	Zly	Raca	Piszczu
4	Vlad	1	Typhus	Romek	Skark	Vlad	Grzelich	Lesiu	Duda	Swistak
5	Typhus	0	Typhus	Skark	Vlad	Lesiu	Duda	Kacper	Zozo	Grzelich
6	Duda	1	Shizi	Kacper	Duda	Grzelich	Rudy	Dejw	Octos	Skark
7	Kacper	0	Lesiu	Michulec	Swistak	Shizi	Duda	Skark	Vlad	Karol

In [115]:
```
zmienna_celu = wyniki["Miejsce"]
zmienne_opisujace = wyniki.drop("Miejsce", axis =1)
```

In [117]:
```
zenkodowane_dane = pd.get_dummies(zmienne_opisujace, prefix = ["Kap", "", "", "", "", "", "", "", ""])
wyfiltrowane = zenkodowane_dane.groupby(lambda x:x, axis = 1).sum()
wyfiltrowane
```

Out[117]:

	Kap_Duda	Kap_Kacper	Kap_Ken	Kap_Skark	Kap_Typhus	Kap_Vlad	_Dejw	_Duda	_Grzelich	_Kacper	...	_Romek	_Rudy	_Shizi	_Skark	_Swistak
0	0	0	0	1	0	0	0	0	0	0	...	1	0	0	1	0
1	0	0	0	0	0	1	0	0	0	0	...	1	0	0	1	0
2	0	0	0	0	0	1	0	0	0	0	...	0	0	0	1	0
3	0	0	1	0	0	0	0	0	0	0	...	0	0	0	1	0
4	0	0	0	0	0	1	0	1	1	0	...	1	0	0	1	1
5	0	0	0	0	1	0	0	0	1	1	...	0	0	0	1	0
6	1	0	0	0	0	0	1	1	1	1	...	0	1	1	1	0
7	0	1	0	0	0	0	0	0	1	0	...	0	0	1	1	1

8 rows × 32 columns

In [118]:
```
nasz_las = RandomForestClassifier(n_estimators = 10, min_samples_leaf = 1, max_depth = 3)
nasz_las.fit(wyfiltrowane, zmienna_celu)
```

Out[118]:
```
RandomForestClassifier(bootstrap=True, ccp_alpha=0.0, class_weight=None,
                       criterion='gini', max_depth=3, max_features='auto',
                       max_leaf_nodes=None, max_samples=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, n_estimators=10,
                       n_jobs=None, oob_score=False, random_state=None,
                       verbose=0, warm_start=False)
```

Policzmy sobie co nam powie RandomForest na temat tego co najbardziej wpływało na bycie repki w top3 albo nie. Im większa wartość przypisana tym większy wpływ na decyzję algorytmu miała konkretna informacja

In [126]:
```
nazwy_kolumn = wyfiltrowane.columns
waznosc_cech = nasz_las.feature_importances_
slownik_waznosci = dict(zip(nazwy_kolumn, waznosc_cech))
{k: v for k, v in sorted(slownik_waznosci.items(), key=lambda item: item[1])}
```

Out[126]:
```
{'Kap_Duda': 0.0,
 'Kap_Ken': 0.0,
 'Kap_Typhus': 0.0,
 '_Dejw': 0.0,
 '_Karkowa': 0.0,
 '_Karol': 0.0,
 '_Lesiu': 0.0,
 '_Marios': 0.0,
 '_Michu': 0.0,
 '_Michulec': 0.0,
 '_Octos': 0.0,
 '_Olek': 0.0,
 '_Skark': 0.0,
 '_Swistak': 0.0,
 '_Thurion': 0.0,
 '_Zozo': 0.0,
 '_Rudy': 0.006896551724137935,
 '_Piszczu': 0.006896551724137935,
 '_Shizi': 0.03333333333333333,
 '_Nathaniel': 0.035522714833059674,
 'Kap_Kacper': 0.037241379310344824,
 '_Vlad': 0.044444444444444446,
 '_Duda': 0.048888888888888885,
 'Kap_Skark': 0.05555555555555546,
 '_Typhus': 0.06,
 'Kap_Vlad': 0.061111111111111116,
 '_Zly': 0.06586206896551723,
 '_Grzelich': 0.07333333333333333,
 '_Kacper': 0.07555555555555556,
 '_Raca': 0.08555555555555555,
 '_Ken': 0.15,
 '_Romek': 0.15980295566502462}
```

In [120]:
```
nasza_reg_log = LogisticRegression(penalty = "l2")
nasza_reg_log.fit(wyfiltrowane, zmienna_celu)
```

Out[120]:
```
LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
                   intercept_scaling=1, l1_ratio=None, max_iter=100,
                   multi_class='auto', n_jobs=None, penalty='l2',
                   random_state=None, solver='lbfgs', tol=0.0001, verbose=0,
                   warm_start=False)
```

Policzmy sobie co nam powie Regresja Logistyczna na temat tego co najbardziej wpływało na bycie repki w top3 albo nie. Im większa wartość przypisana tym większy wpływ na nie ugranie top3 według algorytmu miała konkretna informacja

In [131]:
```
wagi = nasza_reg_log.coef_
slownik_wag = dict(zip(nazwy_kolumn, wagi[0,:]))
{k: v for k, v in sorted(slownik_wag.items(), key=lambda item: item[1])}
```

Out[131]:
```
{'_Typhus': -0.3467567441511153,
 '_Ken': -0.2892469137187629,
 'Kap_Typhus': -0.26050608424994154,
 '_Zozo': -0.26050608424994154,
 '_Vlad': -0.2451183778765039,
 'Kap_Kacper': -0.24506034288321027,
 '_Karol': -0.24506034288321027,
 '_Michulec': -0.24506034288321027,
 '_Karkowa': -0.22816898362667237,
 '_Michu': -0.22816898362667237,
 '_Raca': -0.17071718907614870,
 '_Zly': -0.17071718907614870,
 '_Nathaniel': -0.15466721865886787,
 'Kap_Ken': -0.13457969505895,
 '_Olek': -0.13457969505895,
 '_Lesiu': -0.07440118968889031,
 '_Kacper': -0.015392644892258544,
 '_Skark': -4.939407356006018e-06,
 '_Shizi': 5.309647447273366e-05,
 'Kap_Vlad': 0.04832903551587213,
 '_Piszczu': 0.05745179455052655,
 '_Duda': 0.17071224966879727,
 '_Swistak': 0.18610489456105123,
 'Kap_Duda': 0.2451134939357683,
 '_Dejw': 0.2451134939357683,
 '_Octos': 0.2451134939357683,
 '_Rudy': 0.2451134939357683,
 'Kap_Skark': 0.34669870826928656,
 '_Marios': 0.34669870826928656,
 '_Thurion': 0.34669870826928656,
 '_Grzelich': 0.41577592552003,
 '_Romek': 0.5496949620868757}
```

In [127]:
```
nasz_svc = SVC(kernel = "linear")
nasz_svc.fit(wyfiltrowane, zmienna_celu)
```

Out[127]:
```
SVC(C=1.0, break_ties=False, cache_size=200, class_weight=None, coef0=0.0,
    decision_function_shape='ovr', degree=3, gamma='scale', kernel='linear',
    max_iter=-1, probability=False, random_state=None, shrinking=True,
    tol=0.001, verbose=False)
```

Policzmy sobie co nam powie Support Vector Machine, na temat tego co najbardziej wpływało na bycie repki w top3 albo nie. Im większa wartość przypisana tym większy wpływ na nie ugranie top3 według algorytmu miała konkretna informacja.

In [133]:
```
wagi_svc = nasz_svc.coef_
slownik_wag_svc = dict(zip(nazwy_kolumn, wagi_svc[0,:]))
{k: v for k, v in sorted(slownik_wag_svc.items(), key=lambda item: item[1])}
```

Out[133]:
```
{'Kap_Typhus': -0.3752728739270458,
 '_Zozo': -0.3752728739270458,
 '_Karkowa': -0.3579901808216483,
 '_Michu': -0.3579901808216483,
 'Kap_Kacper': -0.30555555555555555,
 '_Karol': -0.30555555555555555,
 '_Michulec': -0.30555555555555555,
 '_Typhus': -0.23893076276022307,
 '_Vlad': -0.21238503889944527,
 '_Raca': -0.17135652746985525,
 '_Zly': -0.17135652746985525,
 '_Kacper': -0.1628878350276008,
 '_Ken': -0.14546762605902383,
 '_Nathaniel': -0.14316987234448628,
 '_Shizi': -0.09317051665611054,
 '_Lesiu': -0.04102851142958985,
 'Kap_Ken': -0.0022977537145375446,
 '_Olek': -0.0022977537145375446,
 '_Skark': -2.7755575615628914e-16,
 'Kap_Vlad': 0.13863986488136026,
 '_Duda': 0.17135652746985514,
 '_Piszczu': 0.18663365333573096,
 'Kap_Duda': 0.21238503889944445,
 '_Dejw': 0.21238503889944445,
 '_Octos': 0.21238503889944445,
 '_Rudy': 0.21238503889944445,
 'Kap_Skark': 0.3321012794163334,
 '_Marios': 0.3321012794163334,
 '_Thurion': 0.3321012794163334,
 '_Swistak': 0.3342443624974557,
 '_Grzelich': 0.4769120830254106,
 '_Romek': 0.6139110166421 8}
```