In [115]:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import datetime
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import roc_auc_score
```

In [2]:

```
def date_parser_func(date):
    return datetime.datetime.strptime(date, "%Y-%m-%d %H:%M:%S")
```

In [3]:

```
df_game_result = pd.read_csv("game_results.csv",parse_dates=["timestamp"],date_p
arser=date_parser_func)
df_users = pd.read_csv("users.csv",parse_dates=["create_time"],date_parser=date_
parser_func)
df_predictions = pd.read_csv("predictions.csv")
```

In [4]:

```
df_game_result.shape
```

Out[4]:

```
(812733, 9)
```

In [5]:

```
df_game_result.head()
```

Out[5]:

	id	user_id	timestamp	winner	length	magic_used	player_cards	round	type
0	218714	218490	2019-11-10 01:14:52	1	375	7	52	1	1
1	219061	218490	2019-11-10 01:23:06	1	475	3	41	2	1
2	219430	218490	2019-11-10 01:31:24	1	476	0	13	3	1
3	219689	218490	2019-11-10 01:38:43	1	329	2	52	1	1
4	219859	218492	2019-11-10 01:42:48	1	297	2	52	1	1

In [6]:

```
df_users.head()
```

Out[6]:

	id	create_time
0	218490	2019-11-10 00:57:10
1	218492	2019-11-10 01:32:13
2	218493	2019-11-10 01:43:16
3	218499	2019-11-10 03:26:18
4	218507	2019-11-10 07:02:37

In [7]:

```
df_game_result.dtypes
```

Out[7]:

```
id                         int64
user_id                    int64
timestamp         datetime64[ns]
winner                     int64
length                     int64
magic_used                 int64
player_cards               int64
round                      int64
type                       int64
dtype: object
```

In [8]:

```
df_users.dtypes
```

Out[8]:

```
id                          int64
create_time    datetime64[ns]
dtype: object
```

Дата считалась правильно

Проверим данные на наличие пропущенных значений

In [9]:

```
df_game_result.isna().sum()
```

Out[9]:

```
id              0
user_id         0
timestamp       0
winner          0
length          0
magic_used      0
player_cards    0
round           0
type            0
dtype: int64
```

In [10]:

```
df_users.isna().sum()
```

Out[10]:

```
id             0
create_time    0
dtype: int64
```

Пропущенных значений нет

Соединим 2 набора данных

In [11]:

```
df_users.rename(columns={"id":"user_id"},inplace=True)
```

In [12]:

```
result_df = df_game_result.join(df_users.set_index("user_id"),on="user_id")
```

Как мне кажется, наиболее оптимальной будет модель использующая деревья, так как она обладает наибольшей интерпретируемостью, поэтому все сгенеренные данные не нужно будет масштабировать. Данный пункт было бы разумнее описать в части про обучение модели, но мне кажется здесь тоже уместно.

Создадим новый столбец данных: разницу между временем регистрации и временем игры в секундах

In [13]:

```
result_df['diff'] = (result_df.timestamp-result_df.create_time)
```

In [14]:

```
result_df['diff'] = result_df['diff'].apply(lambda x: x.seconds)
```

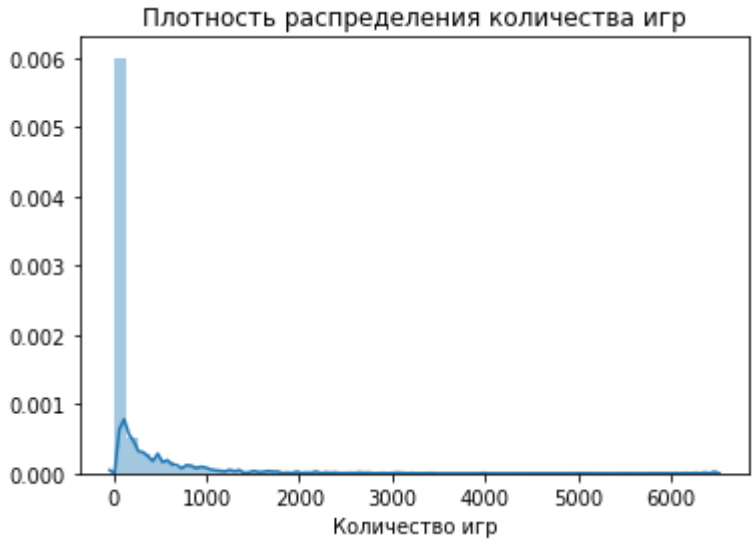
Посмотрим сколько игр сыграл каждый игрок

In [15]:

```
ax = sns.distplot(result_df.user_id.value_counts())
plt.xlabel("Количество игр")
plt.title("Плотность распределения количества игр")
```

Out[15]:

```
Text(0.5, 1.0, 'Плотность распределения количества игр')
```



Как видно из графика распределения большинство игроков играют меньше 1000 игр, найдем это число. Найдем моду распределения.

In [16]:

```
result_df.user_id.value_counts().mode()
```

Out[16]:

```
0    3
dtype: int64
```

Большинство игроков играют 3 игры.

Рассмотрим эти данные детальнее

In [17]:

```
result_df.user_id.value_counts().describe()
```

Out[17]:

```
count    5289.000000
mean      153.664776
std       388.296857
min         1.000000
25%         3.000000
50%        12.000000
75%        95.000000
max      6479.000000
Name: user_id, dtype: float64
```

75 процентов игроков сыграли 95 или менее игр

In [ ]:

```
```

In [19]:

```
result_df['count_of_games']=0
```

Out[19]:

	id	user_id	timestamp	winner	length	magic_used	player_cards	round	t
0	218714	218490	2019-11-10 01:14:52	1	375	7	52	1	
1	219061	218490	2019-11-10 01:23:06	1	475	3	41	2	
2	219430	218490	2019-11-10 01:31:24	1	476	0	13	3	
3	219689	218490	2019-11-10 01:38:43	1	329	2	52	1	
4	219859	218492	2019-11-10 01:42:48	1	297	2	52	1	
...	...	...	...	...	...	...	...	...	
812728	19207349	234692	2020-05-09 11:35:10	0	254	0	9	1	
812729	19208623	234692	2020-05-09 11:50:55	0	45	0	0	1	
812730	19212265	234692	2020-05-09 12:32:59	1	701	0	16	1	
812731	19212509	234692	2020-05-09 12:36:07	1	156	0	52	2	
812732	19212820	234692	2020-05-09 12:39:57	1	199	0	52	3	

812733 rows × 12 columns

Функция для подсчета сыграных игр

In [22]:

```
def count_of_games(pd_arr):
    prev_index = -1
    for index in range(len(pd_arr)):
        if pd_arr.iloc[index]['user_id'] == pd_arr.iloc[prev_index]['user_id']:
            pd_arr.iloc[index,pd_arr.columns.get_loc('count_of_games')] = pd_arr
.iloc[prev_index]['count_of_games']+1
        prev_index = index
        if index%100000==0:
            print(index)
```

Отсортируем всех игроков по user_id и времени игры, для того, чтобы правильно посчитать количетсво сыграных игр на текущий момент

In [23]:

```
result_df.sort_values(['user_id','timestamp'],inplace=True)
```

СЛЕДУЮЩАЯ ЧАСТЬ ООЧЕНЬ ДОЛГО СЧИТАЕТСЯ

In [ ]:

```
count_of_games(result_df)
```

In [27]:

```
result_df.drop(['timestamp'],axis=1,inplace=True)
```

Вычислим день недели, возможно по выходным люди чаще становятся космонавтами

In [32]:

```
result_df['weekend'] = 0
```

In [48]:

```
def find_weekend(line):
    if (line['create_time'].weekday()>4):
        return 1
    else:
        return 0
```

In [51]:

```
result_df.weekend = result_df.apply(find_weekend,axis=1)
```

In [52]:

```
result_df
```

Out[52]:

	id	user_id	winner	length	magic_used	player_cards	round	type	create_
0	218714	218490	1	375	7	52	1	1	2019-1 00:5
1	219061	218490	1	475	3	41	2	1	2019-1 00:5
2	219430	218490	1	476	0	13	3	1	2019-1 00:5
3	219689	218490	1	329	2	52	1	1	2019-1 00:5
5	219943	218490	1	389	2	20	2	1	2019-1 00:5
...	...	...	...	...	...	...	...	...	
731592	15802664	234754	1	199	0	52	2	1	2020-0 23:1
731606	15802858	234754	1	151	0	52	3	1	2020-0 23:1
739724	15984613	234754	1	264	2	52	1	1	2020-0 23:1
739738	15984892	234754	1	129	0	52	2	1	2020-0 23:1
739755	15985250	234754	1	168	4	52	3	1	2020-0 23:1

812733 rows × 12 columns

In [53]:

```
result_df.drop(['create_time'],axis=1,inplace=True)
```

Добавим переменную которую нужно предсказать, предварительно сохранив данные

In [58]:

```
df_pred = pd.read_csv('predictions.csv')
```

In [68]:

```
result_df = result_df.join(df_pred.set_index(['id']),on='user_id')
```

In [75]:

```
result_df.corr()
```

Out[75]:

	id	user_id	winner	length	magic_used	player_cards	
id	1.000000	0.965359	-0.012337	0.004728	-0.027301	-0.013411	-0.0
user_id	0.965359	1.000000	-0.003846	0.043626	0.000362	-0.010824	-0.0
winner	-0.012337	-0.003846	1.000000	-0.001035	0.064093	0.384497	-0.0
length	0.004728	0.043626	-0.001035	1.000000	0.152333	0.010219	-0.0
magic_used	-0.027301	0.000362	0.064093	0.152333	1.000000	0.110698	0.0
player_cards	-0.013411	-0.010824	0.384497	0.010219	0.110698	1.000000	0.0
round	-0.004061	-0.000727	-0.066190	-0.022726	0.027292	0.004712	1.0
type	-0.018008	-0.033916	-0.078429	-0.109665	-0.030518	-0.003590	-0.0
diff	0.052022	0.024627	-0.007051	-0.048775	-0.078303	-0.017428	-0.0
count_of_games	0.093812	-0.014470	-0.024228	-0.251443	-0.070643	0.026032	-0.0
weekend	-0.066613	-0.050847	0.002132	-0.013834	-0.002494	0.001688	-0.0
prediction	-0.019640	0.013601	0.010700	0.065179	0.162494	0.033252	0.0

Наибольшая корреляция предскаываемой переменной наблюдается с используемой магией и разницей во времени между играми

Пропущенные значения заполним как -1, так будет удобнее

In [77]:

```
result_df.fillna(-1,inplace=True)
```

In [ ]:

```

```

In [ ]:

```

```

Построение модели

Так как для предсказания будем использовать деревья, то можно не заниматься масшатибруемостью данных

In [106]:

```
answer = result_df[result_df.prediction==-1]
data = result_df[result_df.prediction!=-1]
```

In [107]:

```
answer.shape
```

Out[107]:

(169617, 12)

In [108]:

```
data.shape
```

Out[108]:

(643116, 12)

In [109]:

```
answer.drop(['prediction'],axis=1,inplace=True)
```

In [110]:

```
answer
```

Out[110]:

	id	user_id	winner	length	magic_used	player_cards	round	type	diff
4	219859	218492	1	297	2	52	1	1	635.0
6	220071	218492	1	368	0	4	2	1	1024.0
8	220283	218492	1	284	0	52	3	1	1334.0
58	245848	218535	1	278	1	52	1	1	497.0
61	246207	218535	1	218	0	52	2	1	739.0
...	...	...	...	...	...	...	...	...	...
731028	15790626	234734	1	186	2	52	2	1	5112.0
731058	15791153	234734	1	236	0	8	3	1	5376.0
731073	15791361	234747	1	178	2	16	1	1	395.0
731093	15791789	234747	1	175	0	52	2	1	604.0
731107	15792068	234747	1	124	0	52	3	1	748.0

169617 rows × 11 columns

In [111]:

```
y = data.prediction.to_numpy()
X = data.drop(['prediction'],axis=1).to_numpy()
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33, random
_state=42)
```

In [114]:

```
rfc = RandomForestClassifier()
rfc.fit(X_train,y_train)
```

Out[114]:

```
RandomForestClassifier(bootstrap=True, ccp_alpha=0.0, class_weight=
None,
                       criterion='gini', max_depth=None, max_featur
es='auto',
                       max_leaf_nodes=None, max_samples=None,
                       min_impurity_decrease=0.0, min_impurity_spli
t=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, n_estimators=1
00,
                       n_jobs=None, oob_score=False, random_state=N
one,
                       verbose=0, warm_start=False)
```

В качестве метрики используем ROC_AUC. Наиболее демократичная метрика, которая достаточно хорошо отражает работу классификатора.

In [117]:

```
result = rfc.predict_proba(X_test)
```

In [129]:

```
print("ROC_AUC_SCORE: ",roc_auc_score(y_test,result[:,1]))
```

```
ROC_AUC_SCORE:  0.9916704794931576
```

Можно было бы построить побольше графиков отражающих распределения каждой переменной, но, по моему мнению это немного лишнее в данной задаче. При желании это можно сделать

In [ ]:

```

```

In [ ]:

```

```