Project description

The gym chain Model Fitness is developing a customer interaction strategy based on analytical data.

One of the most common problems gyms and other services face is customer churn. How do you know if a customer is no longer with you? You can calculate churn based on people who get rid of their accounts or don't renew their contracts. However, sometimes it's not obvious that a client has left: they may walk out on tiptoes.

For a gym, it makes sense to say a customer has left if they don't come for a month. Of course, it's possible they're in Cancun and will resume their visits when they return, but's that's not a typical case. Usually, if a customer joins, comes a few times, then disappears, they're unlikely to come back.

In order to fight churn, Model Fitness has digitized a number of its customer profiles. Your task is to analyze them and come up with a customer retention strategy.

You should:

- Learn to predict the probability of churn (for the upcoming month) for each customer
- Draw up typical user portraits: select the most outstanding groups and describe their main features
- Analyze the factors that impact churn most
- Draw basic conclusions and develop recommendations on how to improve customer service:
    - Identify target groups
    - Suggest measures to cut churn
    - Describe any other patterns you see with respect to interaction with customers

Data description

Model Fitness provided you with CSV files containing data on churn for a given month and information on the month preceding it. The dataset includes the following fields:

- **'Churn'** — the fact of churn for the month in question Current dataset fields:
- User data for the preceding month:
    - **'gender'**
    - **'Near_Location'** — whether the user lives or works in the neighborhood where the gym is located
    - **'Partner'** — whether the user is an employee of a partner company (the gym has partner companies whose employees get discounts; in those cases the gym stores information on customers' employers)
    - **'Promo_friends'** — whether the user originally signed up through a "bring a friend" offer (they used a friend's promo code when paying for their first membership)
    - **'Phone'** — whether the user provided their phone number
    - **'Age'**
    - **'Lifetime'** — the time (in months) since the customer first came to the gym
- Data from the log of visits and purchases and data on current membership status
    - **'Contract_period'** — 1 month, 3 months, 6 months, or 1 year
    - **'Month_to_end_contract'** — the months remaining until the contract expires
    - **'Group_visits'** — whether the user takes part in group sessions
    - **'Avg_class_frequency_total'** — average frequency of visits per week over the customer's lifetime
    - **'Avg_class_frequency_current_month'** — average frequency of visits per week over the preceding month
    - **'Avg_additional_charges_total'** — the total amount of money spent on other gym services: cafe, athletic goods, cosmetics, massages, etc.

Table of Contents

Step 1. Downloading the data

In [1]:
```python
import pandas as pd
import scipy.stats as stats
import datetime as dt
import numpy as np

import seaborn as sns
from matplotlib import pyplot as plt
import plotly.express as pxs
import plotly.graph_objects as go
from plotly.subplots import make_subplots

from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, roc_auc_score
from sklearn.cluster import KMeans
from scipy.cluster.hierarchy import dendrogram, linkage

import warnings
warnings.filterwarnings('ignore')
```

In [2]:
```python
try:
    data = pd.read_csv('gym_churn_us.csv')
except:
    data = pd.read_csv('/datasets/gym_churn_us.csv')
```

In [3]:
```python
data.head()
```

Out[3]:

	gender	Near_Location	Partner	Promo_friends	Phone	Contract_period	Group_visits	Age	Avg_additional_charges_total	Month_to_end_contract
0	1	1	1	1	0	6	1	29	14.227470	5.0
1	0	1	0	0	1	12	1	31	113.202938	12.0
2	0	1	1	0	1	1	0	28	129.448479	1.0
3	0	1	1	1	1	12	1	33	62.669863	12.0
4	1	1	1	1	1	1	0	26	198.362265	1.0

[back to table of contents](#)

Step 2. Exploratory data analysis (EDA)

Let's take a look at the dataset:

In [4]:
```python
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4000 entries, 0 to 3999
Data columns (total 14 columns):
 #   Column                             Non-Null Count  Dtype  
---  ------                             --------------  -----  
 0   gender                             4000 non-null   int64  
 1   Near_Location                      4000 non-null   int64  
 2   Partner                            4000 non-null   int64  
 3   Promo_friends                      4000 non-null   int64  
 4   Phone                              4000 non-null   int64  
 5   Contract_period                    4000 non-null   int64  
 6   Group_visits                       4000 non-null   int64  
 7   Age                                4000 non-null   int64  
 8   Avg_additional_charges_total       4000 non-null   float64
 9   Month_to_end_contract              4000 non-null   float64
 10  Lifetime                           4000 non-null   int64  
 11  Avg_class_frequency_total          4000 non-null   float64
 12  Avg_class_frequency_current_month  4000 non-null   float64
 13  Churn                              4000 non-null   int64  
dtypes: float64(4), int64(10)
memory usage: 437.6 KB
```

According to the information we got with .info() method above, we don't have any missing data here so we can directly check the data for duplicates:

In [5]:
```python
data.drop_duplicates()
print(data.info())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4000 entries, 0 to 3999
Data columns (total 14 columns):
 #   Column                             Non-Null Count  Dtype  
---  ------                             --------------  -----  
 0   gender                             4000 non-null   int64  
 1   Near_Location                      4000 non-null   int64  
 2   Partner                            4000 non-null   int64  
 3   Promo_friends                      4000 non-null   int64  
 4   Phone                              4000 non-null   int64  
 5   Contract_period                    4000 non-null   int64  
 6   Group_visits                       4000 non-null   int64  
 7   Age                                4000 non-null   int64  
 8   Avg_additional_charges_total       4000 non-null   float64
 9   Month_to_end_contract              4000 non-null   float64
 10  Lifetime                           4000 non-null   int64  
 11  Avg_class_frequency_total          4000 non-null   float64
 12  Avg_class_frequency_current_month  4000 non-null   float64
 13  Churn                              4000 non-null   int64  
dtypes: float64(4), int64(10)
memory usage: 437.6 KB
None
```

Nothing changed - it means we didn't have any duplicates here. Now we can study the mean values and standard deviation:

Study the mean values and standard deviation (use the describe() method).

In [6]:
```python
data.describe()
```

Out[6]:

	gender	Near_Location	Partner	Promo_friends	Phone	Contract_period	Group_visits	Age	Avg_additional_charges_tot
count	4000.000000	4000.000000	4000.000000	4000.000000	4000.000000	4000.000000	4000.000000	4000.000000	4000.0000
mean	0.510250	0.845250	0.486750	0.308500	0.903500	4.681250	0.412250	29.184250	146.9437
std	0.499957	0.361711	0.499887	0.461932	0.295313	4.549706	0.492301	3.258367	96.3556
min	0.000000	0.000000	0.000000	0.000000	0.000000	1.000000	0.000000	18.000000	0.1482
25%	0.000000	1.000000	0.000000	0.000000	1.000000	1.000000	0.000000	27.000000	68.8688
50%	1.000000	1.000000	0.000000	0.000000	1.000000	1.000000	0.000000	29.000000	136.2201
75%	1.000000	1.000000	1.000000	1.000000	1.000000	6.000000	1.000000	31.000000	210.9496
max	1.000000	1.000000	1.000000	1.000000	1.000000	12.000000	1.000000	41.000000	552.5907

Conclusions:

- **Gender** - the distribution is close to 50\50: the mean value is close to 0.5, median value is 1
- **Near_Location** - the distribution mostly contains "1" value - ve have "0" as min but after 25% it is all about "1". Standard deviation also looks like one of 1/3 distribution
- **Partner** and **Group_visits**- the same story as with "Gender" but here we have a little bit more "0"s
- **Promo_friends** - it is similar to "Near_Location" but we have just a little bit more 0 here - standard deviation looks like we have something like 60\40 distribution (since it's not far from 0,5)
- **Phone** - it is almost all "1" according to mean value
- **Contract_period** and **Month_to_end_contract**- seems like clients mostly buy 1 month trial
- **Age** - our clients are really young! 75% of them are not older then 31!
- **Avg_additional_charges_total** - the data are not homogeneous. Standard deviation is aproximately 2\3 of the mean value!
- **Lifetime** - In general our clients stay for 3-4 month as the mean value and the median value say
- **Avg_class_frequency_total** and **Avg_class_frequency_current_month** - twice a week is good enough
- **Churn** - we loose 1\4 of our clients monthly - the mean value is 0,26

Now let's look at the mean feature values in two groups: for those who left (churn) and for those who stayed and plot bar histograms and feature distributions.

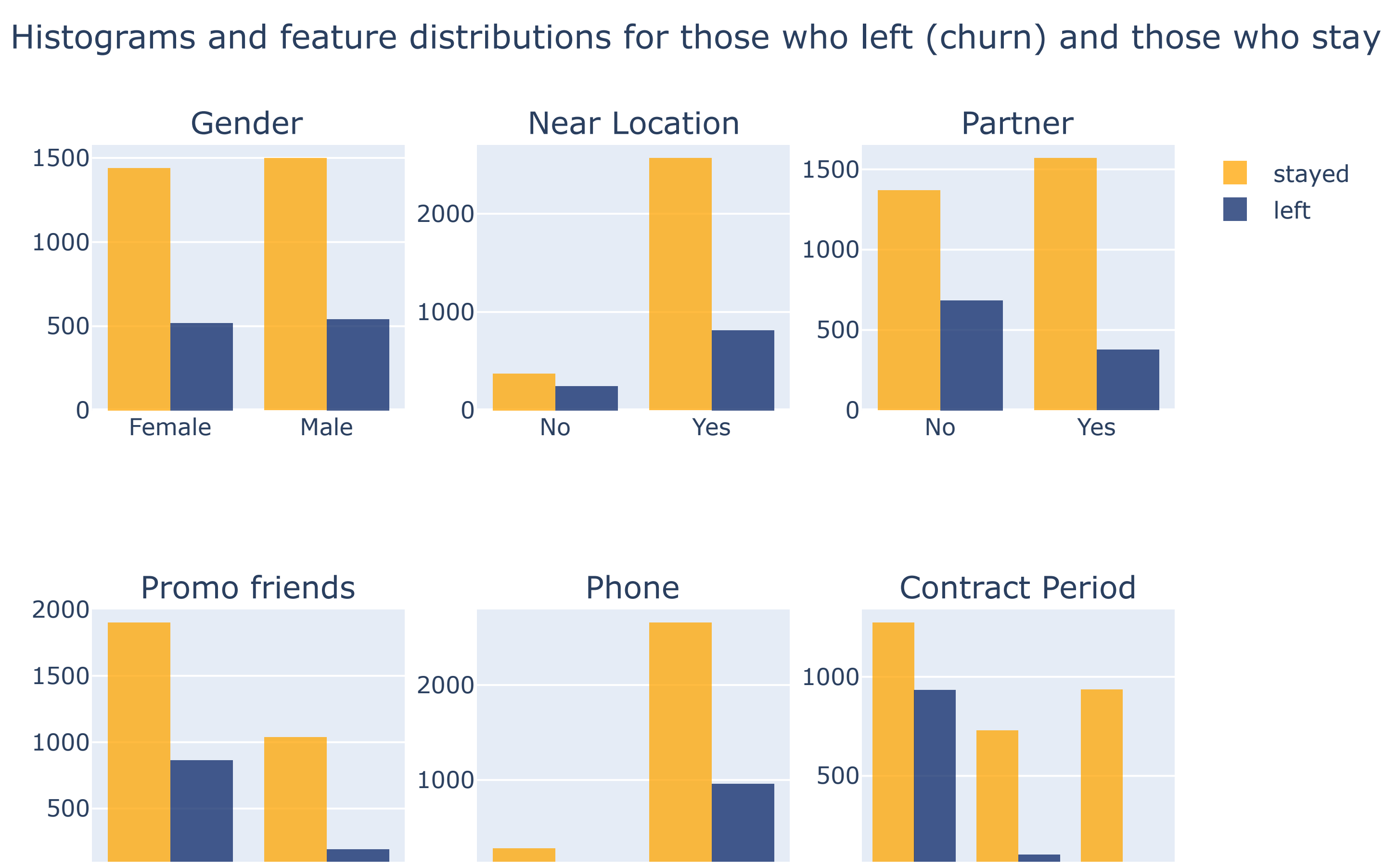
In [7]:
```python
data.groupby(by ='Churn').mean()
```

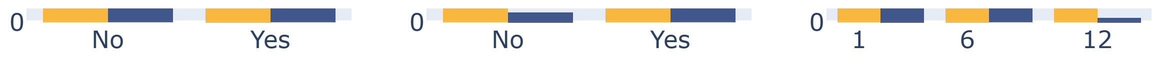
Out[7]:

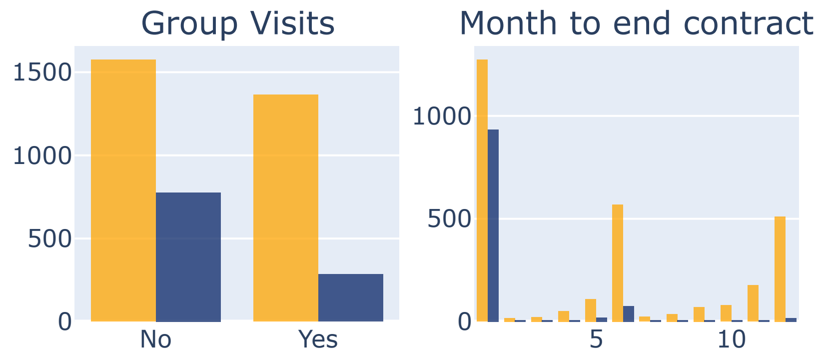
	gender	Near_Location	Partner	Promo_friends	Phone	Contract_period	Group_visits	Age	Avg_additional_charges_total	Month_t
Churn										
0	0.510037	0.873086	0.534195	0.353522	0.903709	5.747193	0.464103	29.976523	158.445715	
1	0.510839	0.768143	0.355325	0.183789	0.902922	1.728558	0.268615	26.989632	115.082899	

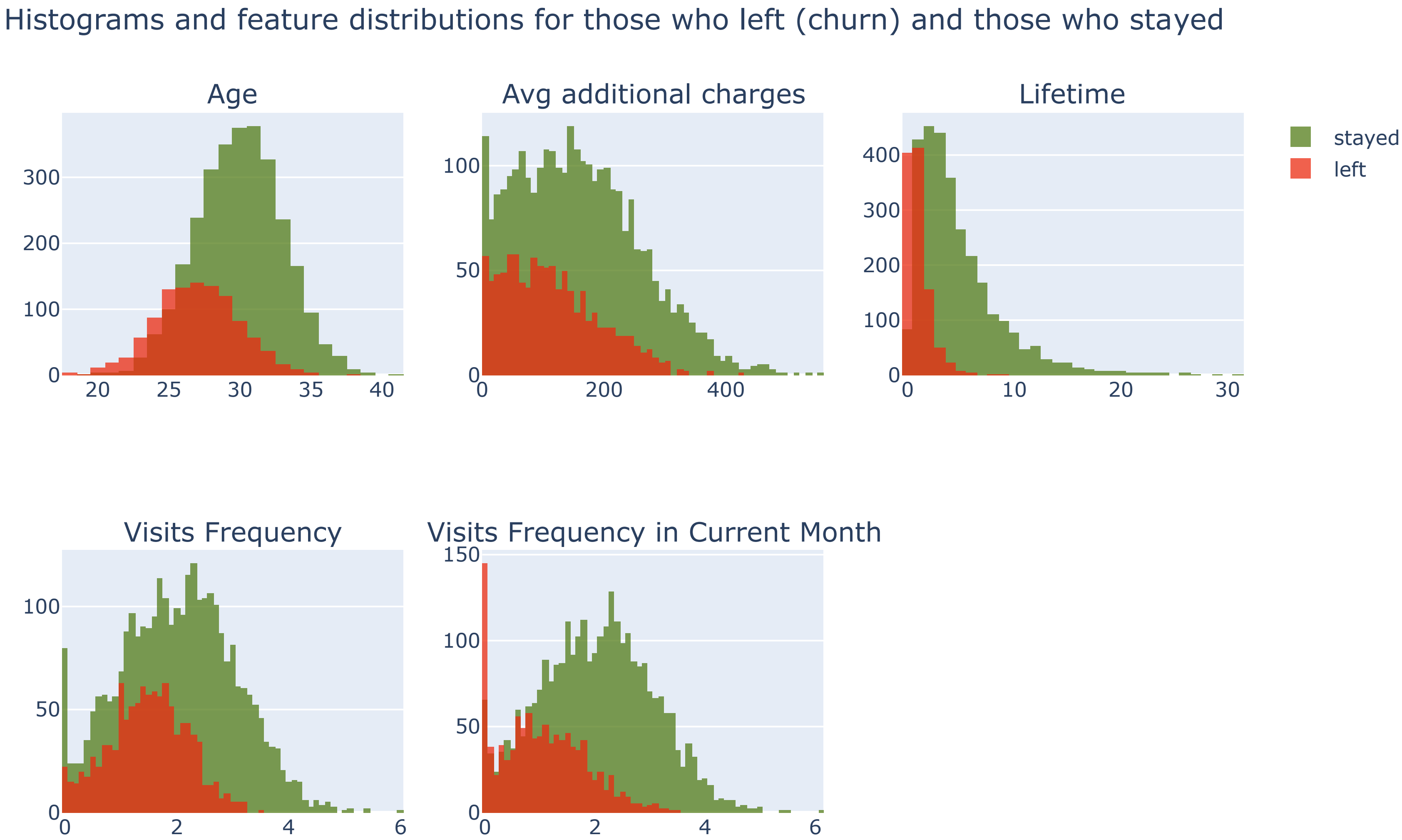
In [8]:
```python
#Plot bar histograms and feature distributions for those who left (churn) and those who stayed.
#1st part of visualization
fig = make_subplots(
    rows=3, cols=3, subplot_titles=("Gender", "Near Location", "Partner", "Promo friends", "Phone", "Contract Period",
                                    "Group Visits", "Month to end contract")
)
#gender
fig.add_trace(go.Histogram(x = data.query('Churn == 0')['gender'], name='stayed', marker_color='#FFA500',
    opacity=0.75, showlegend=False), row=1, col=1)
fig.add_trace(go.Histogram(x = data.query('Churn == 1')['gender'], name='left', marker_color='#082567',
    opacity=0.75, showlegend=False), row=1, col=1)

fig.update_xaxes(
    ticktext=["Female", "Male"],
    tickvals=["0", "1"], row=1, col=1
)

#near location
fig.add_trace(go.Histogram(x = data.query('Churn == 0')['Near_Location'], name='stayed', marker_color='#FFA500',
    opacity=0.75, showlegend=False), row=1, col=2)
fig.add_trace(go.Histogram(x = data.query('Churn == 1')['Near_Location'], name='left', marker_color='#082567',
    opacity=0.75, showlegend=False), row=1, col=2)
fig.update_xaxes(
   ticktext=["No", "Yes"],
    tickvals=["0", "1"], row=1, col=2
)

#partner
fig.add_trace(go.Histogram(x = data.query('Churn == 0')['Partner'], name='stayed', marker_color='#FFA500',
    opacity=0.75), row=1, col=3)
fig.add_trace(go.Histogram(x = data.query('Churn == 1')['Partner'], name='left', marker_color='#082567',
    opacity=0.75), row=1, col=3)
fig.update_xaxes(
   ticktext=["No", "Yes"],
    tickvals=["0", "1"], row=1, col=3
)

#promo friends
fig.add_trace(go.Histogram(x = data.query('Churn == 0')['Promo_friends'], name='stayed', marker_color='#FFA500',
    opacity=0.75, showlegend=False), row=2, col=1)
fig.add_trace(go.Histogram(x = data.query('Churn == 1')['Promo_friends'], name='left', marker_color='#082567',
    opacity=0.75, showlegend=False), row=2, col=1)
fig.update_xaxes(
   ticktext=["No", "Yes"],
    tickvals=["0", "1"], row=2, col=1
)

#phone
fig.add_trace(go.Histogram(x = data.query('Churn == 0')['Phone'], name='stayed', marker_color='#FFA500',
    opacity=0.75, showlegend=False), row=2, col=2)
fig.add_trace(go.Histogram(x = data.query('Churn == 1')['Phone'], name='left', marker_color='#082567',
    opacity=0.75, showlegend=False), row=2, col=2)
fig.update_xaxes(
   ticktext=["No", "Yes"],
    tickvals=["0", "1"], row=2, col=2
)

#contract period
fig.add_trace(go.Histogram(x = data.query('Churn == 0')['Contract_period'], name='stayed', marker_color='#FFA500',
    opacity=0.75, showlegend=False), row=2, col=3)
fig.add_trace(go.Histogram(x = data.query('Churn == 1')['Contract_period'], name='left', marker_color='#082567',
    opacity=0.75, showlegend=False), row=2, col=3)
fig.update_xaxes(
   ticktext=["1", "6", "12"],
    tickvals=["1", "6", "12"], row=2, col=3
)

#group visits
fig.add_trace(go.Histogram(x = data.query('Churn == 0')['Group_visits'], name='stayed', marker_color='#FFA500',
    opacity=0.75, showlegend=False), row=3, col=1)
fig.add_trace(go.Histogram(x = data.query('Churn == 1')['Group_visits'], name='left', marker_color='#082567',
    opacity=0.75, showlegend=False), row=3, col=1)
fig.update_xaxes(
   ticktext=["No", "Yes"],
    tickvals=["0", "1"], row=3, col=1
)

#Month to end contract
fig.add_trace(go.Histogram(x = data.query('Churn == 0')['Month_to_end_contract'], name='stayed', marker_color='#FFA500',
    opacity=0.75, showlegend=False), row=3, col=2)
fig.add_trace(go.Histogram(x = data.query('Churn == 1')['Month_to_end_contract'], name='left', marker_color='#082567',
    opacity=0.75, showlegend=False), row=3, col=2)
fig.update_xaxes(
   ticktext=["1", "6", "12"],
    tickvals=["1", "6", "12"], row=4, col=1
)

fig.update_layout(
```

```
    title_text="Histograms and feature distributions for those who left (churn) and those who stayed",
    height=800, width=750
)

fig.show()






#2nd part of visualization
fig = make_subplots(
    rows=2, cols=3, subplot_titles=("Age", "Avg additional charges", "Lifetime", "Visits Frequency",
                                    "Visits Frequency in Current Month")
)

#age
fig.add_trace(go.Histogram(x = data.query('Churn == 0')['Age'], name='stayed', marker_color='#527C18',
    opacity=0.75, showlegend=False), row=1, col=1)
fig.add_trace(go.Histogram(x = data.query('Churn == 1')['Age'], name='left', marker_color='#EC2B11',
    opacity=0.75, showlegend=False), row=1, col=1)


#Avg_additional_charges_total
fig.add_trace(go.Histogram(x = data.query('Churn == 0')['Avg_additional_charges_total'], name='stayed', marker_color='#5
    opacity=0.75, showlegend=False), row=1, col=2)
fig.add_trace(go.Histogram(x = data.query('Churn == 1')['Avg_additional_charges_total'], name='left', marker_color='#EC2
    opacity=0.75, showlegend=False), row=1, col=2)

#Lifetime
fig.add_trace(go.Histogram(x = data.query('Churn == 0')['Lifetime'], name='stayed', marker_color='#527C18',
    opacity=0.75, showlegend=True), row=1, col=3)
fig.add_trace(go.Histogram(x = data.query('Churn == 1')['Lifetime'], name='left', marker_color='#EC2B11',
    opacity=0.75, showlegend=True), row=1, col=3)


#Avg_class_frequency_total
fig.add_trace(go.Histogram(x = data.query('Churn == 0')['Avg_class_frequency_total'], name='stayed', marker_color='#527C
    opacity=0.75, showlegend=False), row=2, col=1)
fig.add_trace(go.Histogram(x = data.query('Churn == 1')['Avg_class_frequency_total'], name='left', marker_color='#EC2B11
    opacity=0.75, showlegend=False), row=2, col=1)

#Avg_class_frequency_current_month
fig.add_trace(go.Histogram(x = data.query('Churn == 0')['Avg_class_frequency_current_month'], name='stayed', marker_colo
    opacity=0.75, showlegend=False), row=2, col=2)
fig.add_trace(go.Histogram(x = data.query('Churn == 1')['Avg_class_frequency_current_month'], name='left', marker_color=
    opacity=0.75, showlegend=False), row=2, col=2)


fig.update_layout(
    title_text="Histograms and feature distributions for those who left (churn) and those who stayed",
    height=600, width=900, barmode='overlay'
)

fig.show()
```







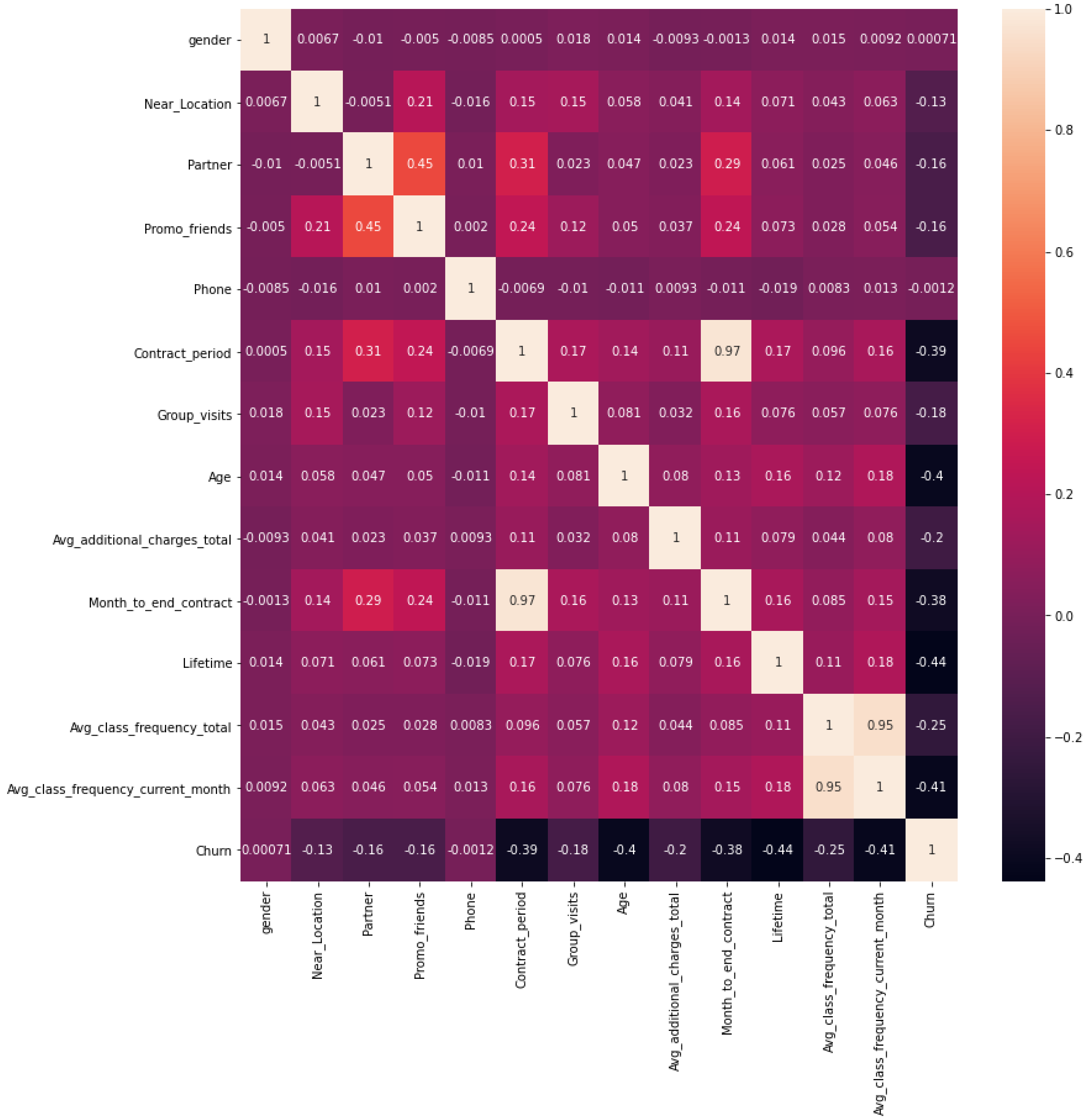


Conclusions:

- **Gender** - the distribution is close to equal - we have both "boys and girls" in both groops
- **Near_Location** - people that finish to train at our gym live a little bit further
- **Partner** - people that finish to train at our gym in 65% of cases don't work at partner companies
- **Promo_friends** - people that finish to train at our gym are twice more "lonely" in their healthy lifestyle
- **Phone** - 90% of both staying and leaving people have phone number in their profiles
- **Contract_period** and **Lifetime** - seems like leaving people were less sure in their decision to go to gym and so their contract period is more then 3 times shorter
- **Group_visits** - 74% of leaving people didn't take part in group sessions while among staying people it's only every second
- **Age** - people that leave are 3 years younger. Not a critical difference
- **Avg_additional_charges_total** - the difference is 33 USD (or 28%). Maybe eavers are simply not satisfied with additional options?
- **Month_to_end_contract** - It is expected that people will leave close to the end of the contract
- **Avg_class_frequency_total** and **Avg_class_frequency_current_month** - leavers attended to the gym less often. Maybe because of the "Near_Location" reason? Let's build a correlation matrix and check it

Now we will build a correlation matrix to show which parameters are correlated the most

In [9]:
```python
corrMatrix = data.corr()
size = (13,13)
plt.subplots(figsize = size)
sns.heatmap(corrMatrix, annot=True)
plt.show()
```



Conclusion: The strongest correlations are between the parameters that are (pretty obvious) correlated - **Contract_period** and **Month_to_end_contract**, **Avg_class_frequency_total** and **Avg_class_frequency_current_month** (0.97 and 0.95). One variable from a pair of highly correlated features should be removed to avoid its domination over other variables during training stage.

We also have an interesting 0,45 correlation between **Partner** and **Promo_friends** - people encourage each other at work. All other correlations are or less strong or have no logic in the posibility of correlation.

In [10]:
```python
data.drop(columns = ['Avg_class_frequency_current_month', 'Month_to_end_contract'], inplace = True)
```

[back to table of contents]()

Step 3. Building a model to predict user churn

Build a binary classification model for customers where the target feature is the user's leaving next month.

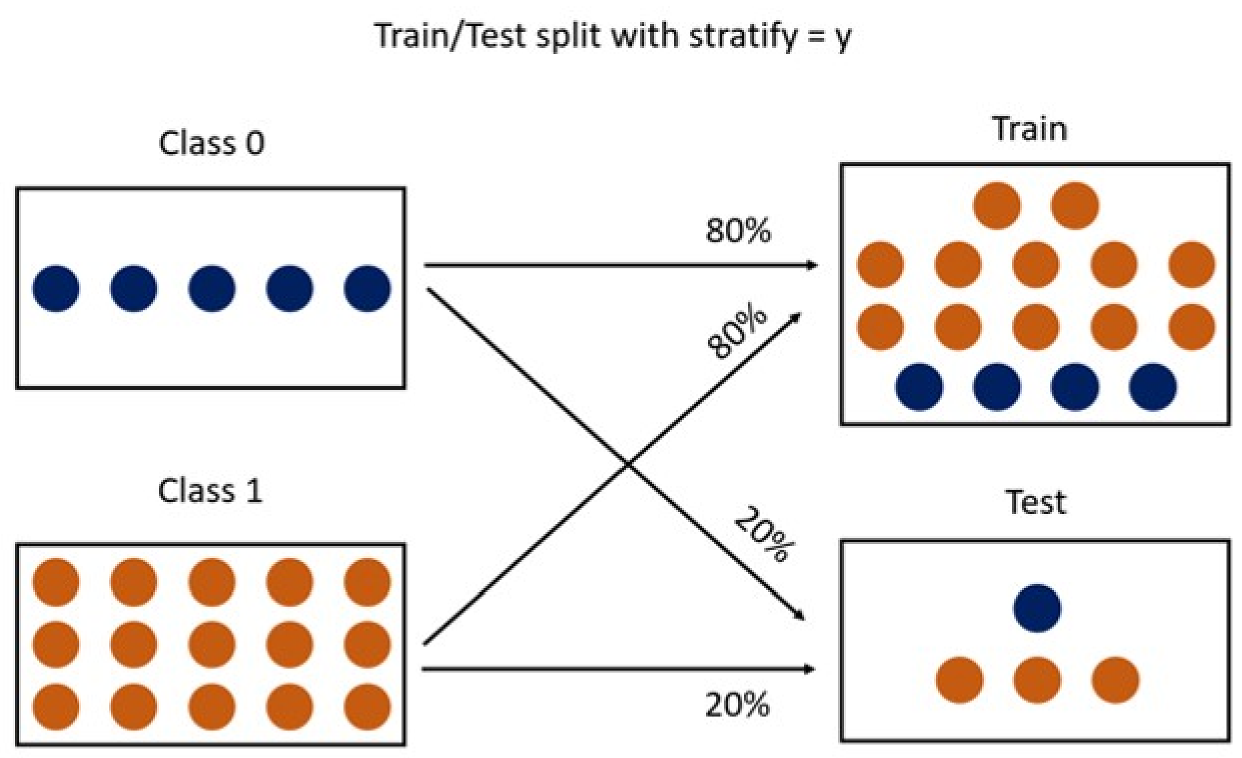
Since the target feature is the user's leaving next month we will define "Churn" column as **y** and all other features as **X** parameteres

In [11]:
```
X = data.drop(columns = ['Churn'])
y = data['Churn']
```

Next step is to divide the data into train and validation sets using the train_test_split() function.

It is necessary to specify random_state inside train_test_split function. Otherwise, sets will be formed differently every time we re-run the code.

We also will add stratify=y inside train_test_split function, where y is our target variable. As a result, the dataset will be devided into two clusters. The first one will contain all the observations of class 0, while the second one – all the observations of class 1. Then, 20% of observations from each cluster will be combined into test set:



This approach helps to split the data into train and test sets in a way that preserves the proportions of observations in each class as observed in the original dataset. It is important since our classes are not balanced.

In [12]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, stratify=y, random_state = 0)
# here we split the data 80/20
```

The model is built. Now we will train it on the train set with two methods: logistic regression and random forest

In [13]:
```
#creating and training logistic regression model:

#we specify solver to silence the warning. "liblinear" fits here since we have only 4000 rows
lr_model = LogisticRegression(solver = "liblinear")
lr_model.fit(X_train, y_train)

# use the trained model to make forecasts
lr_predictions = lr_model.predict(X_test)
lr_probabilities = lr_model.predict_proba(X_test)[:,1]
```

In [14]:
```
#creating scaler
scaler = StandardScaler()
scaler.fit(X_train)

# transform train and test sets
X_train_st = scaler.transform(X_train)
X_test_st = scaler.transform(X_test)

#creating and training random forest model:
rf_model = RandomForestClassifier(n_estimators = 100, random_state = 0)
rf_model.fit(X_train_st, y_train)

# use the trained model to make predictions
rf_predictions = rf_model.predict(X_test_st)
rf_probabilities = rf_model.predict_proba(X_test_st)[:,1]
```

Now we will evaluate accuracy, precision, and recall for both models using the validation data and compare the models.

```
In [15]: def print_all_metrics(y_true, y_pred, y_proba, title = 'Classification metrics'):
             print(title)
             print('\tAccuracy: {:.2f}'.format(accuracy_score(y_true, y_pred)))
             print('\tPrecision: {:.2f}'.format(precision_score(y_true, y_pred)))
             print('\tRecall: {:.2f}'.format(recall_score(y_true, y_pred)))
             #print('\tF1: {:.2f}'.format(f1_score(y_true, y_pred)))
             #print('\tROC_AUC: {:.2f}'.format(roc_auc_score(y_true, y_proba)))
```

```
In [16]: print_all_metrics(y_test, lr_predictions, lr_probabilities, title = 'Metrics for logistic regression:')
```

```
Metrics for logistic regression:
        Accuracy: 0.90
        Precision: 0.83
        Recall: 0.80
```

```
In [17]: print_all_metrics(y_test, rf_predictions, rf_probabilities, title = 'Metrics for random forest:')
```

```
Metrics for random forest:
        Accuracy: 0.89
        Precision: 0.81
        Recall: 0.79
```

Conclusion:

Accuracy is the share of accurate predictions among all predictions. The closer we are to 100% accuracy, the better. **Precision** tells us what share of predictions in class 1 are true. In other words, we look at the share of correct answers only in the target class. The third metric - **Recall** aims at minimizing the opposite risks. Recall demonstrates the number of real class 1 objects you were able to discover with the model. Both Precision and Recall take values from 0 to 1. The closer to 1, the better. Since then **logistic regression model is a little bit better**.

back to table of contents

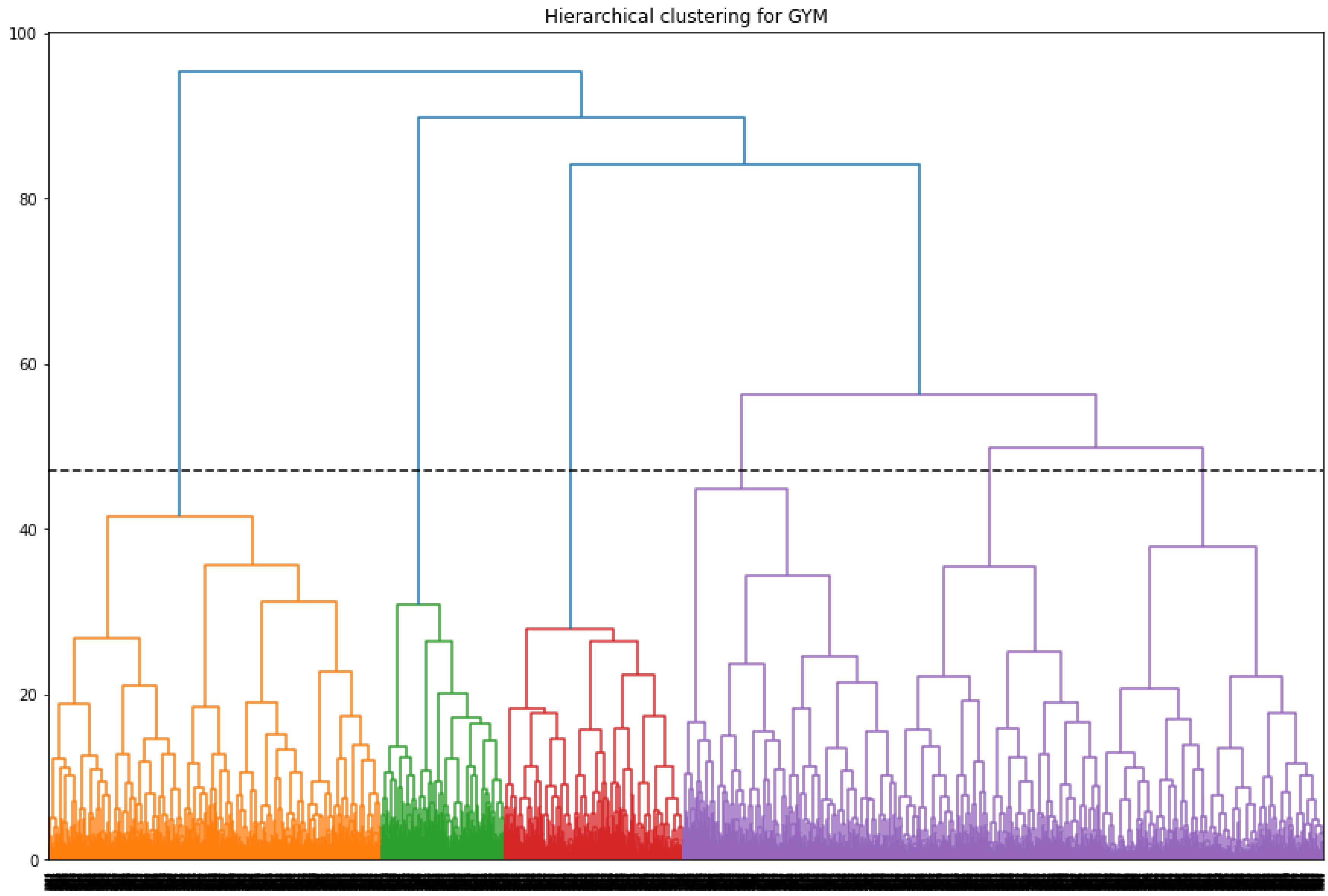
Step 4. Creating user clusters

First of all we need to standardize the data:

```
In [18]: sc = StandardScaler()
         X_sc = sc.fit_transform(X)
```

Now we will use the linkage() function to build a matrix of distances based on the standardized feature matrix and plot a dendrogram. We will use the resulting graph to estimate the number of clusters we can single out.

In [19]:
```
linked = linkage(X_sc, method = 'ward')
plt.figure(figsize=(15, 10))
dendrogram(linked, orientation='top')
plt.title('Hierarchical clustering for GYM')
plt.axhline(y=47, color = 'black', linestyle = "--")
plt.show()
```



Conclusion: In my personal oppinion the best number of clusters will be 6 (as you can see dow to the "---" line) because the "distance" between them and their size are close to equal.

Since the froject was created as part of Yandex100 bootcamp, it was asked to train the clustering model with 5 clusters (so that it'll be easier to compare the results with those of other students)

Next step is to train the clustering model with the K-means algorithm and predict customer clusters. We have to specify random_state for KMeans by the same reason as for train_test_split function:

In [20]:
```
km = KMeans(n_clusters = 5, random_state = 0) # setting the number of clusters as 5
labels = km.fit_predict(X_sc)
```

Now we will look at the mean feature values for clusters, plot distributions of features for the clusters and calculate the churn rate for each cluster

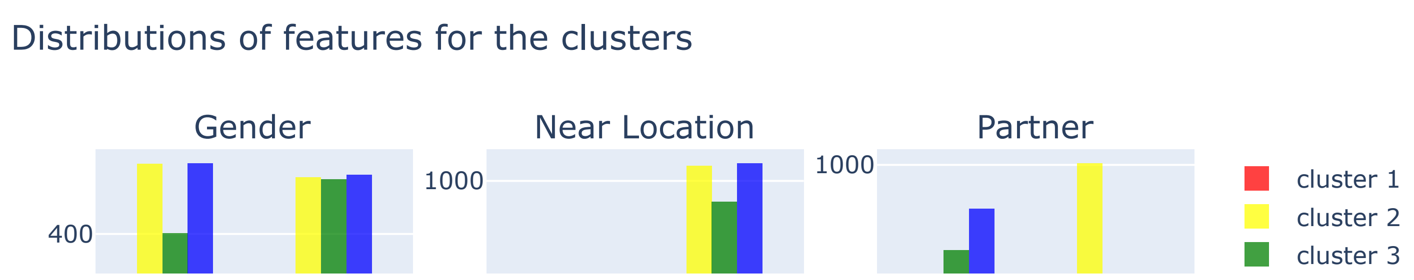
In [21]:
```
data['cluster_km'] = labels

# print the statistics of the mean feature values per cluster
data.groupby(['cluster_km']).mean()
```

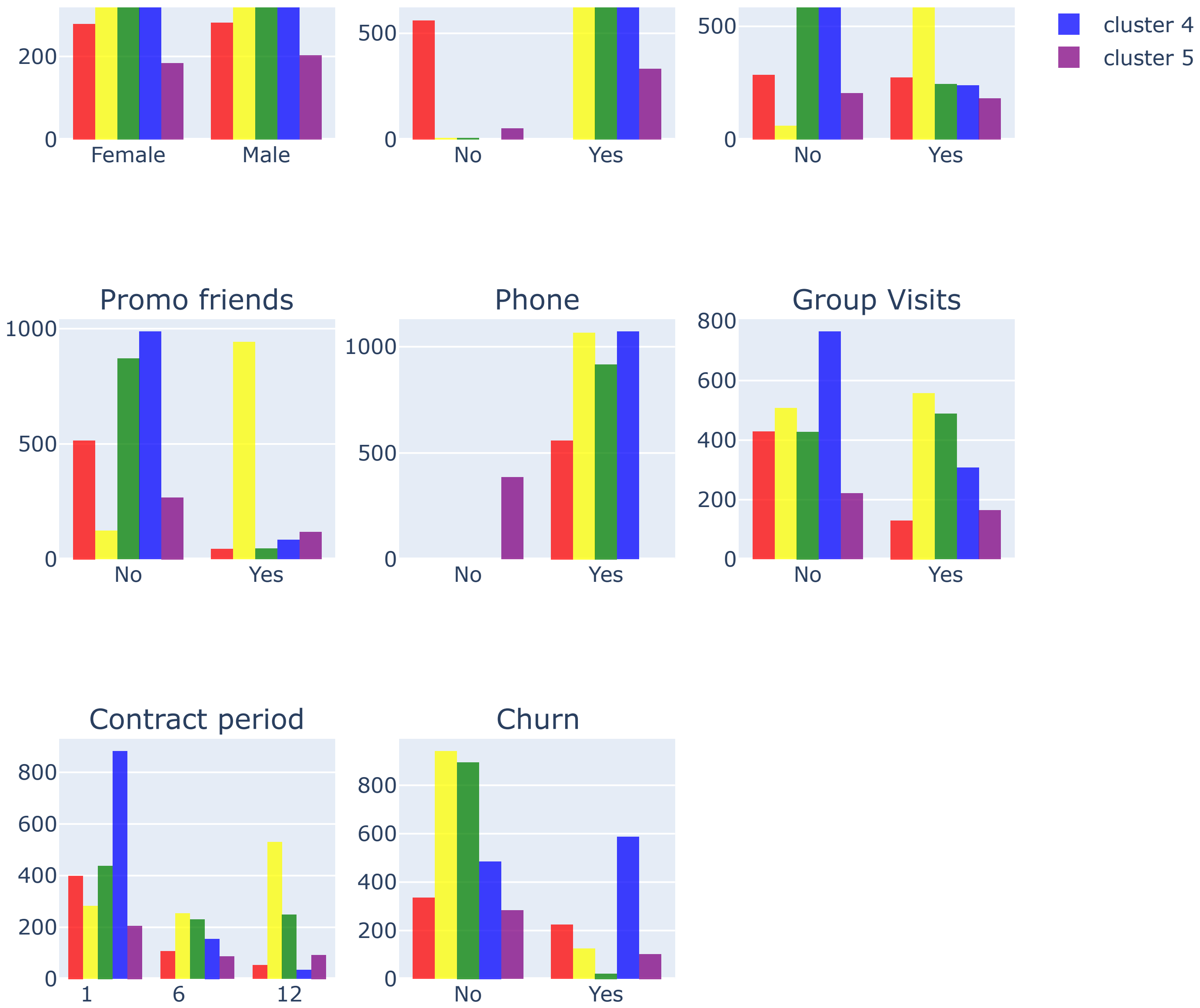
Out[21]:

cluster_km	gender	Near_Location	Partner	Promo_friends	Phone	Contract_period	Group_visits	Age	Avg_additional_charges_total	Lifeti
0	0.502683	0.000000	0.490161	0.078712	1.0	3.000000	0.232558	28.708408	137.385192	3.007
1	0.486867	0.996248	0.943715	0.883677	1.0	7.660413	0.523452	29.474672	149.409166	4.057
2	0.561614	0.996728	0.267176	0.050164	1.0	5.241003	0.533261	30.958561	186.697490	5.778
3	0.488806	1.000000	0.223881	0.078358	1.0	2.082090	0.286381	27.584888	116.455656	1.933
4	0.523316	0.862694	0.471503	0.305699	0.0	4.777202	0.427461	29.297927	144.208179	3.940

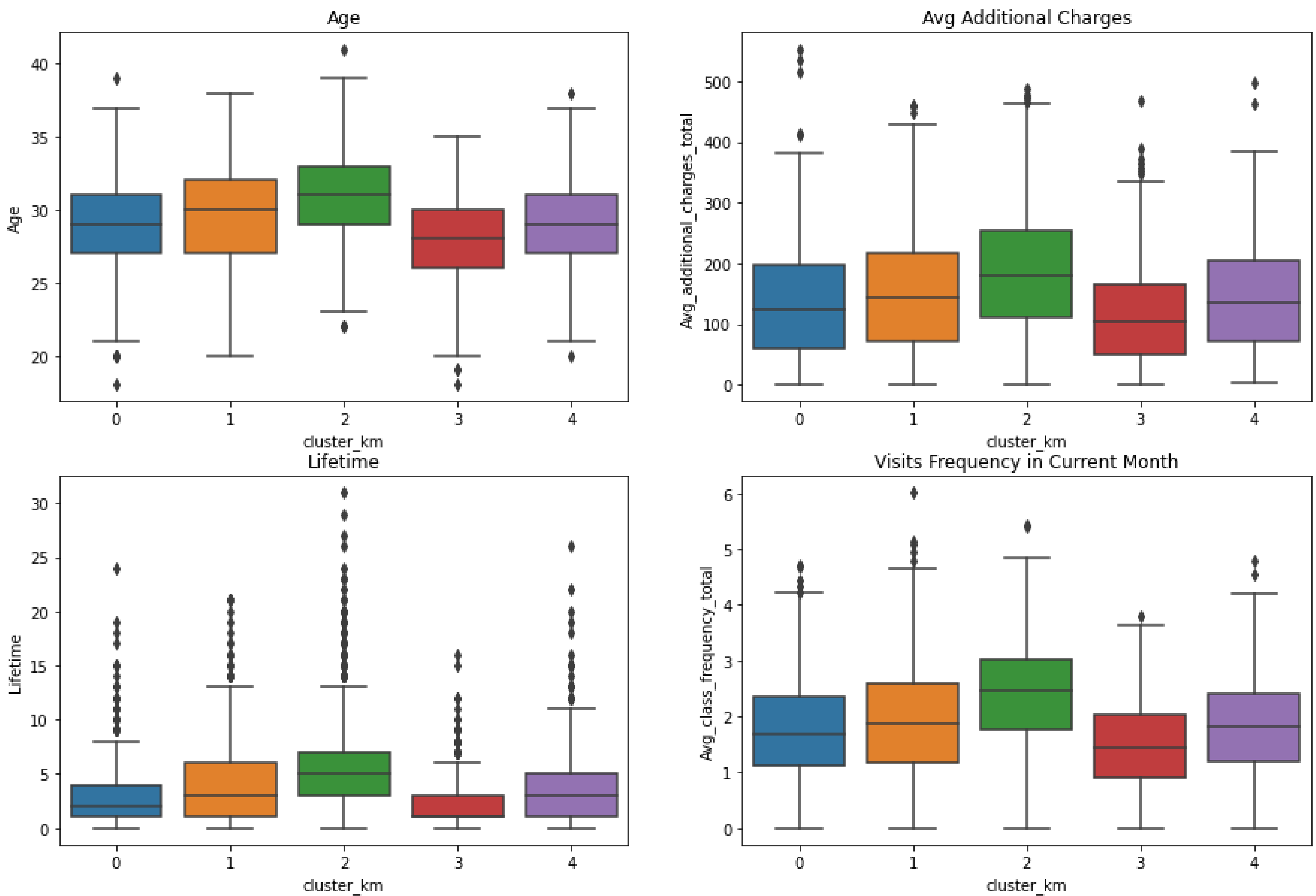
In [22]:
```python
#1st part of visualization
fig = make_subplots(
    rows=3, cols=3, subplot_titles=("Gender", "Near Location", "Partner", "Promo friends", "Phone",
                                    "Group Visits", "Contract period", "Churn")
)
#gender
fig.add_trace(go.Histogram(x = data.query('cluster_km == 0')['gender'], name='cluster 1', marker_color='red',
    opacity=0.75, showlegend=False), row=1, col=1)
fig.add_trace(go.Histogram(x = data.query('cluster_km == 1')['gender'], name='cluster 2', marker_color='yellow',
    opacity=0.75, showlegend=False), row=1, col=1)
fig.add_trace(go.Histogram(x = data.query('cluster_km == 2')['gender'], name='cluster 3', marker_color='green',
    opacity=0.75, showlegend=False), row=1, col=1)
fig.add_trace(go.Histogram(x = data.query('cluster_km == 3')['gender'], name='cluster 4', marker_color='blue',
    opacity=0.75, showlegend=False), row=1, col=1)
fig.add_trace(go.Histogram(x = data.query('cluster_km == 4')['gender'], name='cluster 5', marker_color='purple',
    opacity=0.75, showlegend=False), row=1, col=1)
fig.update_xaxes(
    ticktext=["Female", "Male"],
    tickvals=["0", "1"], row=1, col=1
)

#Near Location
fig.add_trace(go.Histogram(x = data.query('cluster_km == 0')['Near_Location'], name='cluster 1', marker_color='red',
    opacity=0.75, showlegend=False), row=1, col=2)
fig.add_trace(go.Histogram(x = data.query('cluster_km == 1')['Near_Location'], name='cluster 2', marker_color='yellow',
    opacity=0.75, showlegend=False), row=1, col=2)
fig.add_trace(go.Histogram(x = data.query('cluster_km == 2')['Near_Location'], name='cluster 3', marker_color='green',
    opacity=0.75, showlegend=False), row=1, col=2)
fig.add_trace(go.Histogram(x = data.query('cluster_km == 3')['Near_Location'], name='cluster 4', marker_color='blue',
    opacity=0.75, showlegend=False), row=1, col=2)
fig.add_trace(go.Histogram(x = data.query('cluster_km == 4')['Near_Location'], name='cluster 5', marker_color='purple',
    opacity=0.75, showlegend=False), row=1, col=2)
fig.update_xaxes(
    ticktext=["No", "Yes"],
    tickvals=["0", "1"], row=1, col=2
)

#Partner
fig.add_trace(go.Histogram(x = data.query('cluster_km == 0')['Partner'], name='cluster 1', marker_color='red',
    opacity=0.75, showlegend=True), row=1, col=3)
fig.add_trace(go.Histogram(x = data.query('cluster_km == 1')['Partner'], name='cluster 2', marker_color='yellow',
    opacity=0.75, showlegend=True), row=1, col=3)
fig.add_trace(go.Histogram(x = data.query('cluster_km == 2')['Partner'], name='cluster 3', marker_color='green',
    opacity=0.75, showlegend=True), row=1, col=3)
fig.add_trace(go.Histogram(x = data.query('cluster_km == 3')['Partner'], name='cluster 4', marker_color='blue',
    opacity=0.75, showlegend=True), row=1, col=3)
fig.add_trace(go.Histogram(x = data.query('cluster_km == 4')['Partner'], name='cluster 5', marker_color='purple',
    opacity=0.75, showlegend=True), row=1, col=3)
fig.update_xaxes(
    ticktext=["No", "Yes"],
    tickvals=["0", "1"], row=1, col=3
)

#Promo friends
fig.add_trace(go.Histogram(x = data.query('cluster_km == 0')['Promo_friends'], name='cluster 1', marker_color='red',
    opacity=0.75, showlegend=False), row=2, col=1)
fig.add_trace(go.Histogram(x = data.query('cluster_km == 1')['Promo_friends'], name='cluster 2', marker_color='yellow',
    opacity=0.75, showlegend=False), row=2, col=1)
fig.add_trace(go.Histogram(x = data.query('cluster_km == 2')['Promo_friends'], name='cluster 3', marker_color='green',
    opacity=0.75, showlegend=False), row=2, col=1)
fig.add_trace(go.Histogram(x = data.query('cluster_km == 3')['Promo_friends'], name='cluster 4', marker_color='blue',
    opacity=0.75, showlegend=False), row=2, col=1)
fig.add_trace(go.Histogram(x = data.query('cluster_km == 4')['Promo_friends'], name='cluster 5', marker_color='purple',
    opacity=0.75, showlegend=False), row=2, col=1)
fig.update_xaxes(
    ticktext=["No", "Yes"],
    tickvals=["0", "1"], row=2, col=1
)

#Phone
fig.add_trace(go.Histogram(x = data.query('cluster_km == 0')['Phone'], name='cluster 1', marker_color='red',
    opacity=0.75, showlegend=False), row=2, col=2)
fig.add_trace(go.Histogram(x = data.query('cluster_km == 1')['Phone'], name='cluster 2', marker_color='yellow',
    opacity=0.75, showlegend=False), row=2, col=2)
fig.add_trace(go.Histogram(x = data.query('cluster_km == 2')['Phone'], name='cluster 3', marker_color='green',
    opacity=0.75, showlegend=False), row=2, col=2)
fig.add_trace(go.Histogram(x = data.query('cluster_km == 3')['Phone'], name='cluster 4', marker_color='blue',
    opacity=0.75, showlegend=False), row=2, col=2)
fig.add_trace(go.Histogram(x = data.query('cluster_km == 4')['Phone'], name='cluster 5', marker_color='purple',
    opacity=0.75, showlegend=False), row=2, col=2)
fig.update_xaxes(
    ticktext=["No", "Yes"],
    tickvals=["0", "1"], row=2, col=2
)

#Group visits
fig.add_trace(go.Histogram(x = data.query('cluster_km == 0')['Group_visits'], name='cluster 1', marker_color='red',
    opacity=0.75, showlegend=False), row=2, col=3)
```

```
fig.add_trace(go.Histogram(x = data.query('cluster_km == 1')['Group_visits'], name='cluster 2', marker_color='yellow',
    opacity=0.75, showlegend=False), row=2, col=3)
fig.add_trace(go.Histogram(x = data.query('cluster_km == 2')['Group_visits'], name='cluster 3', marker_color='green',
    opacity=0.75, showlegend=False), row=2, col=3)
fig.add_trace(go.Histogram(x = data.query('cluster_km == 3')['Group_visits'], name='cluster 4', marker_color='blue',
    opacity=0.75, showlegend=False), row=2, col=3)
fig.add_trace(go.Histogram(x = data.query('cluster_km == 4')['Group_visits'], name='cluster 5', marker_color='purple',
    opacity=0.75, showlegend=False), row=2, col=3)
fig.update_xaxes(
    ticktext=["No", "Yes"],
    tickvals=["0", "1"], row=2, col=3
)

#Month_to_end_contract
fig.add_trace(go.Histogram(x = data.query('cluster_km == 0')['Contract_period'], name='cluster 1', marker_color='red',
    opacity=0.75, showlegend=False), row=3, col=1)
fig.add_trace(go.Histogram(x = data.query('cluster_km == 1')['Contract_period'], name='cluster 2', marker_color='yellow'
    opacity=0.75, showlegend=False), row=3, col=1)
fig.add_trace(go.Histogram(x = data.query('cluster_km == 2')['Contract_period'], name='cluster 3', marker_color='green',
    opacity=0.75, showlegend=False), row=3, col=1)
fig.add_trace(go.Histogram(x = data.query('cluster_km == 3')['Contract_period'], name='cluster 4', marker_color='blue',
    opacity=0.75, showlegend=False), row=3, col=1)
fig.add_trace(go.Histogram(x = data.query('cluster_km == 4')['Contract_period'], name='cluster 5', marker_color='purple'
    opacity=0.75, showlegend=False), row=3, col=1)
fig.update_xaxes(
   ticktext=["1", "6", "12"],
    tickvals=["1", "6", "12"], row=3, col=1
)

#Churn
fig.add_trace(go.Histogram(x = data.query('cluster_km == 0')['Churn'], name='cluster 1', marker_color='red',
    opacity=0.75, showlegend=False), row=3, col=2)
fig.add_trace(go.Histogram(x = data.query('cluster_km == 1')['Churn'], name='cluster 2', marker_color='yellow',
    opacity=0.75, showlegend=False), row=3, col=2)
fig.add_trace(go.Histogram(x = data.query('cluster_km == 2')['Churn'], name='cluster 3', marker_color='green',
    opacity=0.75, showlegend=False), row=3, col=2)
fig.add_trace(go.Histogram(x = data.query('cluster_km == 3')['Churn'], name='cluster 4', marker_color='blue',
    opacity=0.75, showlegend=False), row=3, col=2)
fig.add_trace(go.Histogram(x = data.query('cluster_km == 4')['Churn'], name='cluster 5', marker_color='purple',
    opacity=0.75, showlegend=False), row=3, col=2)
fig.update_xaxes(
    ticktext=["No", "Yes"],
    tickvals=["0", "1"], row=3, col=2
)

fig.update_layout(
    title_text="Distributions of features for the clusters",
    height=800, width=750
)

fig.show()




#2nd paty of visualization (continious variables)
fig, axes = plt.subplots(2, 2, figsize=(15, 10))
fig.suptitle('Distributions of features for the clusters - continious variables')

#age
sns.boxplot(ax=axes[0, 0], x='cluster_km', y='Age',data=data)
axes[0, 0].set_title('Age')

#Avg_additional_charges_total
sns.boxplot(ax=axes[0, 1], x='cluster_km', y='Avg_additional_charges_total',data=data)
axes[0, 1].set_title('Avg Additional Charges')

#Lifetime
sns.boxplot(ax=axes[1, 0], x='cluster_km', y='Lifetime',data=data)
axes[1, 0].set_title('Lifetime')

#Avg_class_frequency_current_month
sns.boxplot(ax=axes[1, 1], x='cluster_km', y='Avg_class_frequency_total',data=data)
axes[1, 1].set_title('Visits Frequency in Current Month')
plt.show()
```





Distributions of features for the clusters - continious variables



Conclusions:

- **Gender** - the distribution is close to equal
- **Near_Location** - the distribution of 4 last clusters is close to equal but not with the first one - all people of this cluster live far from the gym
- **Partner** - 2nd cluster here has the greatesr ammount of workers from partner companies - aproximatelly twice more then in other clusters
- **Promo_friends** - the same situation as with the previous parameter - we saw already that they have correlation of 0,45
- **Phone** - almost everybody left their phone number but only not the 5th cluster!
- **Contract_period**- 2nd cluster has the greatest ammount here. Looks like partner program gives enormous discount for the 1 year trial
- **Group_visits** - 3rd cluster has noone of group lovers but 4th is the greatest
- **Age** - the age is pretty the same in all groups
- **Avg_additional_charges_total** - the maximum difference is 18 USD (or 11%). Maybe eavers are simply not satisfied with additional options?
- **Lifetime** - 1st cluster leaves us the first but only 1 month earlier then 2nd cluster which has the longest retention
- **Avg_class_frequency_total** - All clusters in general visit gym 1-2 timeeees sa week

- **Churn** - it is the check for all our clusters. We see that the **1st** cluster has the lowest churn rate - it can be explained with ammount of partner program benefits. **4th** cluster is also out of danger - these people live in the same neighbourhood, love group visits. **5th** cluster already has 0,26 charn - maybe because 18% of them don't live in the same neighbourhood. A third of them have promo friends and a halph - partnership but noone (NOONE!) left a phone number. We are sorry to see that 38% of people in **3rd** cluster leave us because they seem lonely. They have the lowest ammount of promo friends and they don't visit group sessions. The highest churn rate is for **1st** cluster. All people from this cluster live far from the gym so maybe that is the reason why they visit us least often and spend the least amount of money on other gym services (they also need money for bus tickets or gas). Only 20% of them have promo-friends.

back to table of contents

Step 5. Conclusions and recommendations on working with customers

One of the most common problems gyms and other services face is customer churn. How do you know if a customer is no longer with you? You can calculate churn based on people who get rid of their accounts or don't renew their contracts. However, sometimes it's not obvious that a client has left: they may walk out on tiptoes.

In this project we:

- trained 2 models to predict the probability of churn (for the upcoming month) for each customer and decided that logistic regression model is a little bit better.
- created user's clusters and drew up typical user portraits and describe their main features.
- analyzed the factors that impact churn most:
  - people that finish to train at our gym live a little bit further
  - people that finish to train at our gym in 65% of cases don't work at partner companies
  - leaving people were less sure in their decision to go to gym and so their contract period is more then 3 times shorter
  - 74% of leaving people didn't take part in group sessions while among staying people it's only every second

The strongest retention values show people that are ready to interaction - they work in partner companies, come to the gym by friend-brings-friend system and take part in group sessions while lonely people that don't have these features leave us. The distance is also one of the problems. Maybe it's time to create special offers like "Had a run to gym? - GET 10% DISCOUNT for a protein shake!" or "Each 5th group session is for FREE!". We also need any offers for people that visit us less then twice a week or calculate more profitable price for 3- and 6-months trials.

People leave their phone numbers in 90% of cases - maybe they are waiting for our discount offers? :)

back to table of contents

Requirements

In [23]:
```
pip freeze > requirements.txt
```

```
Note: you may need to restart the kernel to use updated packages.
```