PROJECT 01

FACEBOOK

1. Data Preprocessing
2. Exploratory Data Analysis (EDA)
3. Correlation Analysis
4. Analyze Ad-Hoc Business Questions

By:
Mehrdad Mansourdehghan

GitHUb

YouTube Channel

LinkedIn Profile

Send me E-mail

1. Project Goals

In this project, I'm going to use a dataset that is related to Facebook users. During this project, first I clean and preprocess the dataset. After that, I run exploratory data analysis (EDA) to know data much better and do some statistical techniques. In the next step, I analyze the correlation between variables. Finally, I answer some ad-hoc business questions and use Python coded, plots and visualizations to answer them.

2. Language, libraries, tools:

Language: Python
Libraries: Pandas, NumPy, Pickle, PandasProfilling, Seaborn, Matplotlib, StatsModels, scipy
IDE: Jupyter Notebook
Application: Microsoft Excel

3. Data

In this project I have one dataset that contains nearly 99K records. In the following, you can read the data documentation.

Userid: it's a label and code of each user (text)
Age: the age of user (numeric)
Gender: the gender of user (nominal)

- Male
- female

tenure: length of time (in days) that a user has been a member of the Facebook platform and they are loyal. (numeric)
friend_count: the number of social friends in Facebook that user has (numeric)
friendships_initiated: the number of times that user requested friendship to somebody else. (numeric)
Likes: the number of likes that user did. (numeric)
likes_received: the number of likes that user got from other. (numeric)
mobile_likes: the number of sending likes just in mobile app. (numeric)

mobile_likes_received: the number of receiving likes just in mobile app. (numeric)
www_likes: the number of sending likes just in website. (numeric)
www_likes_received: the number of receiving likes just in website. (numeric)

```
import numpy as np
import pandas as pd
```

```
df = pd.read_csv('dataset/data.csv')
```

Data Preprocessing

1. Look at Data:

Let's see the dimensions of the dataframe.

```
df.shape
(99003, 12)
```

2. Remove Unnecessary Columns:

We need all the columns for this project.

3. Change Column Name:

I change the name of the column based on the following:

friend_count: friends

friendships_initiated: request

likes: g_likes

likes_received: r_likes.

```
col_name = ['userid', 'age', 'gender', 'tenure', 'friends',
       'request', 'g_likes', 'r_likes', 'mobile_likes',
       'mobile_likes_received', 'www_likes', 'www_likes_received']

df.columns = col_name

print(df.columns)
Index(['userid', 'age', 'gender', 'tenure', 'friends', 'request', 'g_likes',
       'r_likes', 'mobile_likes', 'mobile_likes_received', 'www_likes',
       'www_likes_received'],
```

4. Handle Missing Values:

I create a for loop to iterate among all columns to realize whether they have null values or not. I'm looking for the number of null values in every single column as well as the percentage of null values.

```
for col in df.columns:
    number_null = df.loc[: , col].isnull().sum()
    perc_null = (number_null / df.shape[0]) * 100
    print('{} - {} - %{}'.format(col, number_null, round(perc_null,3)))
```

```
userid - 0 - %0.0
age - 0 - %0.0
gender - 175 - %0.177
tenure - 2 - %0.002
friends - 0 - %0.0
request - 0 - %0.0
g_likes - 0 - %0.0
r_likes - 0 - %0.0
mobile_likes - 0 - %0.0
mobile_likes_received - 0 - %0.0
www_likes - 0 - %0.0
www_likes_received - 0 - %0.0
```

We must have a different approach to handling null values. Since I have a large dataset:

Categorical:

less than 5%, I drop the rows.

between 5% and 30%, I impute with mode.

More than 30%, create a new label as “Other.”

Numerical:

between 0% and 30%, I impute with mean or median.

More than 30%, I drop the rows.

However, the best way is consulting with expert domain. Let’s begin with categorical variables. First, I deal with “gender” that has %0.177 null values. since it is less than 5%, I drop them.

```
df = df.dropna(subset = ['gender'])
```

And now, I work on numerical variables. I work on “tenure” and since it's less than 30%, I should impute it. But before doing this, I must make sure about distribution shape of these columns to see whether they are right-skewed or left-skewed. It can be helpful when I want to decide choosing mean or median for imputing. Also, I should check the data type to be sure about numerical type.

```
print(df['tenure'].dtypes)
```

```
float64
```

The result shows this variable has correct data type. Now we can see its distribution.

```
print('Skewness :' , round(df['tenure'].skew() ,3))

mean_tenure = df['tenure'].mean()
median_tenure = df['tenure'].median()

if mean_tenure > median_tenure:
    print('Mean is bigger than Median. Left Skewed. Median for imputing')
else:
    print('Mean is smaller than Median. Right Skewed. Mean for imputing')
```

```
Skewness : 1.531
Mean is bigger than Median. Left Skewed. Median for imputing
```

The result shows that I should choose median for imputing.

```
df['tenure'] = df['tenure'].fillna(median_tenure).round(0)
```

Finally, we check the null values for dataset again.

```
for col in df.columns:
    number_null = df.loc[: , col].isnull().sum()
    perc_null = (number_null / df.shape[0]) * 100
    print('{} - {} - %{}'.format(col, number_null, round(perc_null,5)))
```

```
userid - 0 - %0.0
age - 0 - %0.0
gender - 0 - %0.0
tenure - 0 - %0.0
friends - 0 - %0.0
request - 0 - %0.0
g_likes - 0 - %0.0
r_likes - 0 - %0.0
mobile_likes - 0 - %0.0
mobile_likes_received - 0 - %0.0
www_likes - 0 - %0.0
www_likes_received - 0 - %0.0
```

5. Handle Duplicate Values:

Now we should handle duplicate rows. Since all values might be same, we just we need to check whether there are two rows that all values in all columns are the same or not.

```
duplicate_rows = df.duplicated()

if duplicate_rows.any():
    print("The DataFrame has duplicate rows.")
else:
    print("The DataFrame does not have duplicate rows.")
```

```
The DataFrame does not have duplicate rows.
```

6. Handle Number Variables:

First of all, I declare all number variables.

```
num_list = ['age', 'tenure', 'friends','request', 'g_likes', 'r_likes', 'mobile_likes', 'mobile_likes_received', 'www_likes',
       'www_likes_received']
```

I must make sure about the data type of the number variable. Just because the column shows numbers, it doesn't mean that they are numbers. Thus, with regular expression I should clean them.

```
def non_numeric(x):
    non_numeric_df = pd.DataFrame(df[df[x].astype(str).str.contains('[^\d\.]+')])
    return non_numeric_df
```

Now, I apply this function to those variables.

```
non_numeric('age')
```

userid	age	gender	tenure	friends	request	g_likes	r_likes	mobile_likes	mobile_likes_received	www_likes	www_likes_received

```
non_numeric('tenure')
```

userid	age	gender	tenure	friends	request	g_likes	r_likes	mobile_likes	mobile_likes_received	www_likes	www_likes_received

```
non_numeric('friends')
```

userid	age	gender	tenure	friends	request	g_likes	r_likes	mobile_likes	mobile_likes_received	www_likes	www_likes_received

```
non_numeric('request')
```

userid	age	gender	tenure	friends	request	g_likes	r_likes	mobile_likes	mobile_likes_received	www_likes	www_likes_received

The results show fortunately in those variables, we just have number nothing else. However, we can see the sum of those variables for double check.

```
for i in range(len(num_list)):
    var_sum = df.loc[: , num_list[i]].sum()
    print(num_list[i] , var_sum)
```

```
age 3677768
tenure 52936947.0
friends 19407045
request 10621917
g_likes 15428029
r_likes 14099133
mobile_likes 10490184
mobile_likes_received 8313152
www_likes 4937840
www_likes_received 5785977
```

In the next, we should run sanity check. In this dataset, “g_likes” must be aggregation of “mobile_likes” and “ww_likes”. Also, for receiving likes we have the same approach. We check this matter and if is there any mismatched data, we consider new columns as getting like and receiving likes.

```
df['sanity_g_like'] = df['mobile_likes'] + df['www_likes']
df['diff_g_like'] = df['g_likes'] - df['sanity_g_like']
```

Now we should check whether there is any difference or not.

```
df['diff_g_like'].sum()

5
```

It means there are 5 records that sum of “mobile_likes” and “ww_likes” does not equal to “g_likes”. In this case, we should ignore the old one, and consider the new column’s value.

```
df.sort_values(by = 'diff_g_like', ascending = False)
```

userid	age	gender	tenure	friends	request	g_likes	r_likes	mobile_likes	mobile_likes_received	www_likes	www_likes_received	sanity_g_like	diff_g_like
1930804	41	female	782.0	35	14	176	159	174	142	0	17	174	2
1535515	23	female	510.0	629	440	1298	1522	1216	1172	81	350	1297	1
1030735	21	male	373.0	473	409	896	136	895	113	0	23	895	1
1182272	17	female	1082.0	4464	1716	2049	17159	681	9657	1367	7502	2048	1

```
df.loc[df['diff_g_like'] > 0, 'g_likes'] = df['sanity_g_like']
```

Now, we run the same process for receiving like.

```
df['sanity_r_like'] = df['mobile_likes_received'] + df['www_likes_received']
df['diff_r_like'] = df['r_likes'] - df['sanity_r_like']
df['diff_r_like'].sum()

4
```

```
df.sort_values(by = 'diff_r_like', ascending = False)
```

tenure	friends	request	g_likes	r_likes	mobile_likes	mobile_likes_received	www_likes	www_likes_received	sanity_g_like	diff_g_like	sanity_r_like	diff_r_like
32.0	293	174	3423	2222	2581	1847	842	374	3423	0	2221	1
1005.0	4351	2210	5045	4498	4742	3909	303	588	5045	0	4497	1
600.0	1028	754	1916	1292	1672	722	244	569	1916	0	1291	1
1704.0	501	221	138	65	99	55	39	9	138	0	64	1

Then, I change them to the correct value.

```
df.loc[df['diff_r_like'] > 0, 'r_likes'] = df['sanity_r_like']
```

Finally, I remove the extra columns.

7. Handle Categorical Variables:

Now, I declare all categorical variables.

```
cat_list = ['gender']
```

Then, we must make sure about the possible range for each of them. They must be the same with data documentation.

"Gender"

```
print(df['gender'].unique())
```

```
['male' 'female']
```

```
print(df['gender'].value_counts())
```

```
male      58574
female    40254
```

All of them are correct and based on data documentation.

Exploratory Data Analysis (EDA)

In this section we do descriptive statistical analysis to know data much more. By doing this, we can realize their distribution, central tendency measurement, the dispersion measurement and shape of the data. First, we import a package that is very helpful in descriptive statistical analysis. Moreover, we write some functions for Matplotlib and Seaborn to add more information about the statistical analysis by drawing distribution plots (quantitative variables) and bar charts (qualitative variables).

1. Import Package and Dataset:

First, I should import necessary packages and also the cleaned dataset.

```
import pandas as pd
import numpy as np
import sweetviz as sv
import pickle

import seaborn as sns

import matplotlib.pyplot as plt
import matplotlib.mlab as mlab
import matplotlib
plt.style.use('ggplot')
from matplotlib.pyplot import figure

%matplotlib inline
```

2. Run PandasProfiling Package:

Then I activated the PandasProfilling library and prepared the environment for statistical analysis. This library gives us all important information about descriptive statistics analysis.

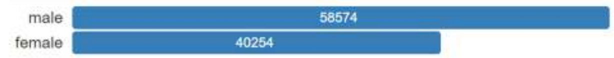
```
import pandas_profiling

profile = pandas_profiling.ProfileReport(df, minimal=True)
profile.to_file('Statistical Analysis.html')
```

3. Qualitative Variables:

Here we analyze qualitative variables, and we see each label in every single variable account for the highest frequency. Then, we can see the statistical interpretation for categorical variables:

gender

Value	Count	Frequency (%)
male	58574	59.3%
female	40254	40.7%



Male users account for the majority gender

4. Quantitative Variables:

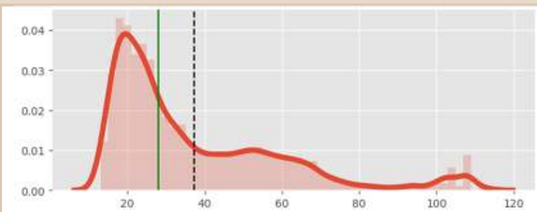
Now, let's analyze descriptive statics for quantitative variables. In this section we can find central tendency, dispersion, and shape measurements. Then we draw the distribution plot to compare with normal distribution.

```python
def kde_plot(x):
    import seaborn as sns
    import matplotlib.pyplot as plt

    plt.figure(figsize = (8,3))
    sns.distplot(df[x], kde_kws={"lw": 5}, hist_kws = {'alpha': 0.25})
    sns.despine(left = True)

    mean = df[x].mean()
    median = df[x].median()

    plt.axvline(mean, color ='black', linestyle ='dashed')
    plt.axvline(median, color ='green', linestyle ='solid')
    plt.xlabel('')
    plt.ylabel('')

    return plt.show()
```

Now we can see the statistical interpretation for quantitative variables:

age

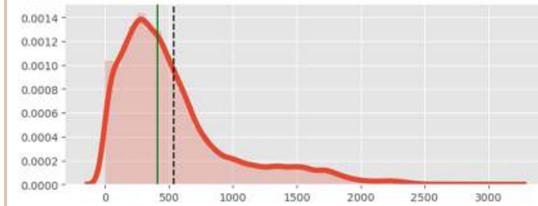
Minimum	13	Standard deviation	22.52554322
5-th percentile	15	Coefficient of variation (CV)	0.6053003848
Q1	20	Kurtosis	1.58140572
median	28	Mean	37.21382604
Q3	50	Median Absolute Deviation (MAD)	10
95-th percentile	89	Skewness	1.418839279
Maximum	113	Sum	3677768
Range	100	Variance	507.4000974
Interquartile range (IQR)	30	Monotonicity	Not monotonic



The average age of users is 37.2 years old.
Half of the users are equal to or less than 28 years old.
The difference between the oldest and youngest users is 100 years.
25% of the users are aged less than or equal to 20 years.
75% of the users are aged less than or equal to 50 years.
50% of the users are aged between 20 years and 50 years.
The distribution is skewed to right (big outlier and we have some users who are older than majority of others)

tenure

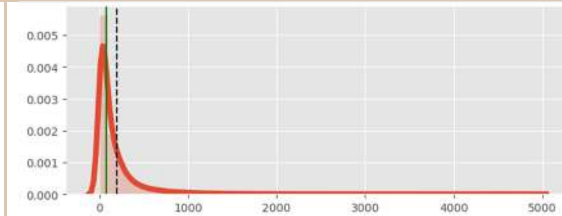
Minimum	0	Standard deviation	454.2541671
5-th percentile	47	Coefficient of variation (CV)	0.8480472217
Q1	226	Kurtosis	2.19550629
median	412	Mean	535.6472558
Q3	673	Median Absolute Deviation (MAD)	212
95-th percentile	1567	Skewness	1.530860759
Maximum	3139	Sum	52936947
Range	3139	Variance	206346.8483
Interquartile range (IQR)	447	Monotonicity	Not monotonic



The average days of user's loyalty is 535 days.
Half of the users have had an account on Facebook for less than 412 days.
The difference between the maximum and minimum tenure is 3139 days.
25% of the users are loyal to Facebook less than or equal to 226 days.
75% of the users are loyal to Facebook less than or equal to 673 days.
25% of the users are loyal to Facebook between 226 and 673 days.
The distribution is right skewed (big outlier and we have some users that are loyal much more than majority of them).

friends

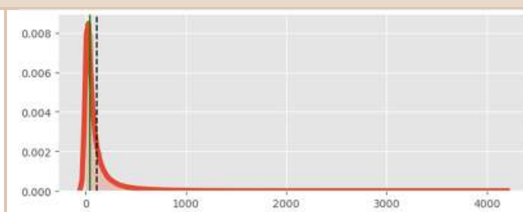
Minimum	0	Standard deviation	387.4598878
5-th percentile	3	Coefficient of variation (CV)	1.973092029
Q1	31	Kurtosis	50.08544462
median	82	Mean	196.371929
Q3	206	Median Absolute Deviation (MAD)	64
95-th percentile	720	Skewness	6.059193109
Maximum	4923	Sum	19407045
Range	4923	Variance	150125.1646
Interquartile range (IQR)	175	Monotonicity	Not monotonic



The average having friends is 196 users.
Half of the users have friends with less than 82 people.
The difference between the maximum and minimum time to have friends is 4923 users.
25% of the users have friends fewer than or equal to 31 people.
75% of the users have friends fewer than or equal to 206 people.
50% of the users have friends between 31 and 206 people.
The distribution is extremely right skewed (big outlier and we have some users that have much more friends compared to majority of other users).

request

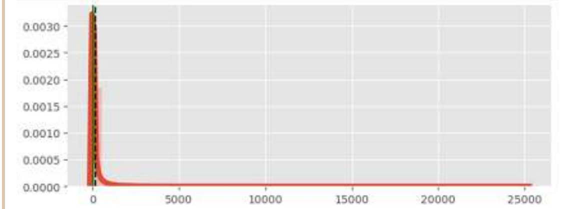
Minimum	0	Standard deviation	188.8598704
5-th percentile	1	Coefficient of variation (CV)	1.757182181
Q1	17	Kurtosis	42.53287331
median	46	Mean	107.4788218
Q3	117	Median Absolute Deviation (MAD)	38
95-th percentile	418	Skewness	5.151263571
Maximum	4144	Sum	10621917
Range	4144	Variance	35668.05066
Interquartile range (IQR)	100	Monotonicity	Not monotonic



The average number of sending requests is 107 times.
Half of the users sent requests less than 46 times.
The difference between the maximum and minimum sending request is 4144 times.
25% of the users sent requests less than 17 times.
75% of the users sent requests less than 117 times.
50% of the users sent requests between 17 and 17 times.
The distribution is extremely right skewed (big outlier and we have some users that sent requests much more that other users).

g_likes

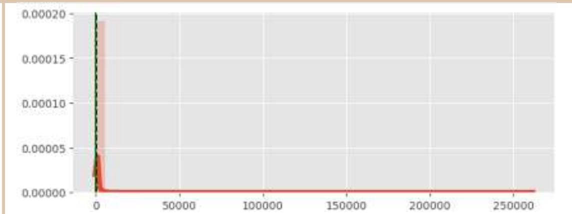
Minimum	0	Standard deviation	572.5478307
5-th percentile	0	Coefficient of variation (CV)	3.667595864
Q1	1	Kurtosis	200.4069858
median	11	Mean	156.1098474
Q3	81	Median Absolute Deviation (MAD)	11
95-th percentile	726	Skewness	11.02429313
Maximum	25111	Sum	15428024
Range	25111	Variance	327811.0184
Interquartile range (IQR)	80	Monotonicity	Not monotonic



The average giving like to other users is 156 times.
Half of the users gave like to others less than 11 times.
The difference between the maximum and minimum number of giving like to other users is 25111 times.
25% of the users liked other users less or equal to 1 time.
75% of the users liked other users less or equal to 81 times.
50% of the users liked other users between 1 and 81 times.
The distribution is extremely right skewed (big outlier and we have some users that like others much more that other users).

r_likes

Minimum	0	Standard deviation	1388.976039
5-th percentile	0	Coefficient of variation (CV)	9.736042845
Q1	1	Kurtosis	17362.80562
median	8	Mean	142.6633039
Q3	59	Median Absolute Deviation (MAD)	8
95-th percentile	560.65	Skewness	112.0165049
Maximum	261197	Sum	14099129
Range	261197	Variance	1929254.438
Interquartile range (IQR)	58	Monotonicity	Not monotonic



The average receive like from other users is 142 times.
Half of the users receive like from others less than 8 times.
The difference between the maximum and minimum number of receiving like other users is 261197 times.
25% of the users received like from other users less or equal to 1 time.
75% of the users received like from other users less or equal to 59 times.
50% of the users received like from other users between 1 and 59 times.
The distribution is extremely right skewed (big outlier and we have some users that received like from others much more that other users).

5. Maximum & Minimum Values:

We can also, see the minimum and maximum values for each quantitative variable:

```
#minimum age
df[df['age'] == df['age'].min()].head(1)
```

	userid	age	gender	tenure	friends	request	g_likes	r_likes
6	1136133	13	male	12.0	0	0	0	0

```
#maximum age
df[df['age'] == df['age'].max()].head(1)
```

	userid	age	gender	tenure	friends	request	g_likes	r_likes
11097	1536088	113	female	1133.0	11	10	0	0

```
#minimum tenure
df[df['tenure'] == df['tenure'].min()].head(1)
```

	userid	age	gender	tenure	friends	request	g_likes	r_likes
7	1680361	13	female	0.0	0	0	0	0

```
#maximum tenure
df[df['tenure'] == df['tenure'].max()].head(1)
```

	userid	age	gender	tenure	friends	request	g_likes	r_likes
86234	1419799	111	male	3139.0	372	40	11	21

```
#minimum friends
df[df['friends'] == df['friends'].min()].head(1)
```

	userid	age	gender	tenure	friends	request	g_likes	r_likes
0	2094382	14	male	266.0	0	0	0	0

```
#maximum friends
df[df['friends'] == df['friends'].max()].head(1)
```

	userid	age	gender	tenure	friends	request	g_likes	r_likes
97985	2090699	103	female	783.0	4923	96	26	80

```
#minimum request
df[df['request'] == df['request'].min()].head(1)
```

	userid	age	gender	tenure	friends	request	g_likes	r_likes
0	2094382	14	male	266.0	0	0	0	0

```
#maximum request
df[df['request'] == df['request'].max()].head(1)
```

	userid	age	gender	tenure	friends	request	g_likes	r_likes
98818	1654565	19	male	394.0	4538	4144	4501	15088

```
#minimum g_likes
df[df['g_likes'] == df['g_likes'].min()].head(1)
```

	userid	age	gender	tenure	friends	request	g_likes	r_likes
0	2094382	14	male	266.0	0	0	0	0

```
#maximum g_likes
df[df['g_likes'] == df['g_likes'].max()].head(1)
```

	userid	age	gender	tenure	friends	request	g_likes	r_likes
96834	1684195	23	male	529.0	1056	665	25111	3447

```
#minimum r_likes
df[df['r_likes'] == df['r_likes'].min()].head(1)
```

	userid	age	gender	tenure	friends	request	g_likes	r_likes
0	2094382	14	male	266.0	0	0	0	0

```
#maximum r_likes
df[df['r_likes'] == df['r_likes'].max()].head(1)
```

	userid	age	gender	tenure	friends	request	g_likes	r_likes
94735	1674584	17	female	401.0	818	395	1016	261197

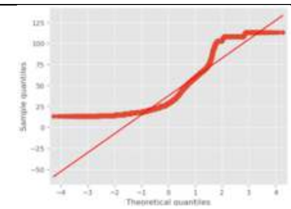
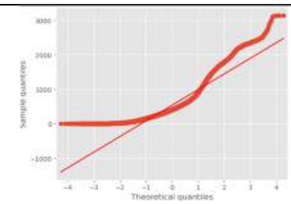
6. Check Distribution Function:

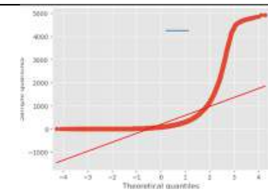
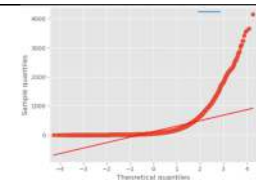
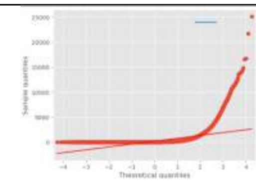
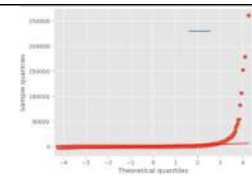
In this section, we want to determine whether a distribution of a variable follows a normal distribution. For doing this, I check with two ways, Quantile-Quantile that compares the distribution of the data to a theoretical normal distribution. If the points on the plot form a straight line, then the data is likely normally distributed. The second way is Kolmogorov-Smirnov test that compares the empirical distribution function of the data to the theoretical normal distribution. If the test statistic is small and the p-value is large, then we conclude that the data is normally distributed.

Q-Q plot

```
import statsmodels.api as sm

def QQ_plot (var):
    sm.qqplot(df[var], line='s')
    plt.xlabel('Theoretical quantiles')
    plt.ylabel('Sample quantiles')
    plt.show()
```

I run the function for all numerical variables.

age	*the points deviate significantly from the line, then the age is not normally distributed.*	
tenure	*the points deviate significantly from the line, then the tenure is not normally distributed.*	

friends	*the points deviate significantly from the line, then the friends is not normally distributed.*	
request	*the points deviate significantly from the line, then the request is not normally distributed.*	
G_likes	*the points deviate significantly from the line, then the request is not normally distributed.*	
R_likes	*the points deviate significantly from the line, then the request may not be normally distributed.*	

Kolmogorov-Smirnov test

This test measures the maximum vertical distance between the empirical cumulative distribution function of the variable and the cumulative distribution function. The p-value measures the probability of observing a test statistic as extreme as or more extreme than the one observed, assuming the null hypothesis that the data follows the reference distribution. If the p-value is greater than the significance level (e.g. 0.05), then we fail to reject the null hypothesis and conclude that the data follows a normal distribution.

```
from scipy.stats import kstest, norm

def KS_test(var):
    kstest_result = kstest(df[var], norm.cdf)

    print(f"test statistic: {kstest_result.statistic:.4f}")
    print(f"test p-value: {kstest_result.pvalue:.4f}")
```

age	*Since the p-value is less than 0.05, we reject the null hypothesis, and it means it is not normal distribution.*	`KS_test('age')`
		`test statistic: 1.0000`
		`test p-value: 0.0000`

tenure	*Since the p-value is less than 0.05, we reject the null hypothesis, and it means it is not normal distribution.*	`KS_test('tenure')`
		`test statistic: 0.9971`
		`test p-value: 0.0000`

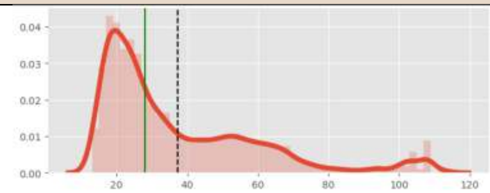
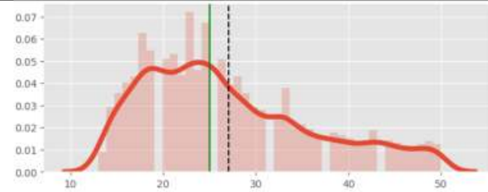
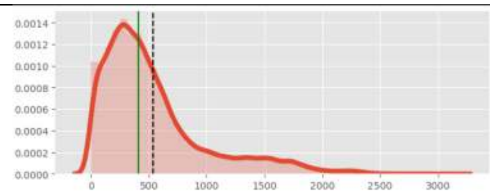
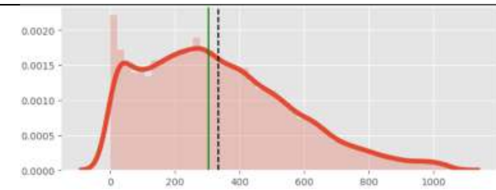
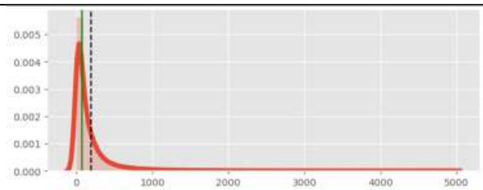
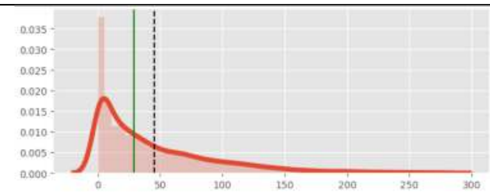
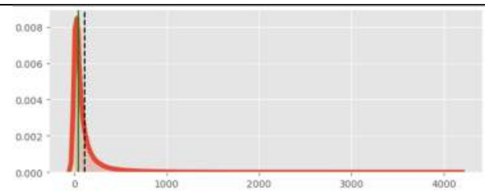
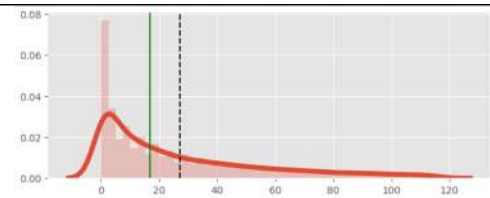
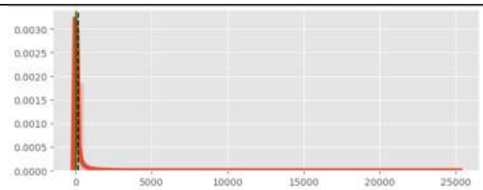
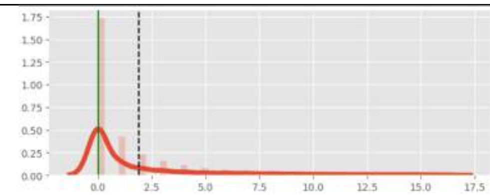
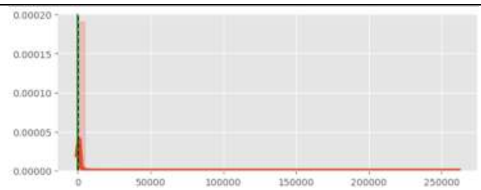
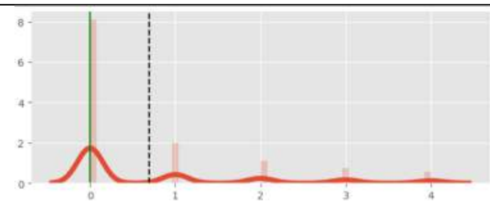
friends	*Since the p-value is less than 0.05, we reject the null hypothesis, and it means it is not normal distribution.*	`KS_test('friends')` test statistic: 0.9491 test p-value: 0.0000
request	*Since the p-value is less than 0.05, we reject the null hypothesis, and it means it is not normal distribution.*	`KS_test('request')` test statistic: 0.9303 test p-value: 0.0000
G_likes	*Since the p-value is less than 0.05, we reject the null hypothesis, and it means it is not normal distribution.*	`KS_test('g_likes')` test statistic: 0.6818 test p-value: 0.0000
R_likes	*Since the p-value is less than 0.05, we reject the null hypothesis, and it means it is not normal distribution.*	`KS_test('r_likes')` test statistic: 0.6566 test p-value: 0.0000

7. Handle Outliers:

In the previous sections, we saw some variables have outliers, and before going further we should handle them. The histogram, distribution and test show they are not normal distribution. So, for handling outlier, we cannot use z-score. Instead, we use MAD (median absolute deviation) technique that is a very robust method for this condition.

```
#find the median of variable
Median = df[var].median()

#create empty column
df['Median_Diff'] = 0

#calculate differenrce
for i in range(len(df)):
    median_diff = abs(df.loc[i , var] - Median)
    df.at[i, 'Median_Diff'] = median_diff

#calculate the median of new column
MAD = df['Median_Diff'].median()

#determine treshold
threshold = MAD * 3

#detect and filter rows based on outlier
df = df[~(df['Median_Diff'] > threshold)]

#remove the differenece column
df = df.drop(['Median_Diff'], axis=1)
```

For the last time, let's look at the histograms.

	Before	After
age		
tenure		
friends		
request		
G_likes		
R_likes		

However, the other technique that we should use for dealing with outliers was quartile method.

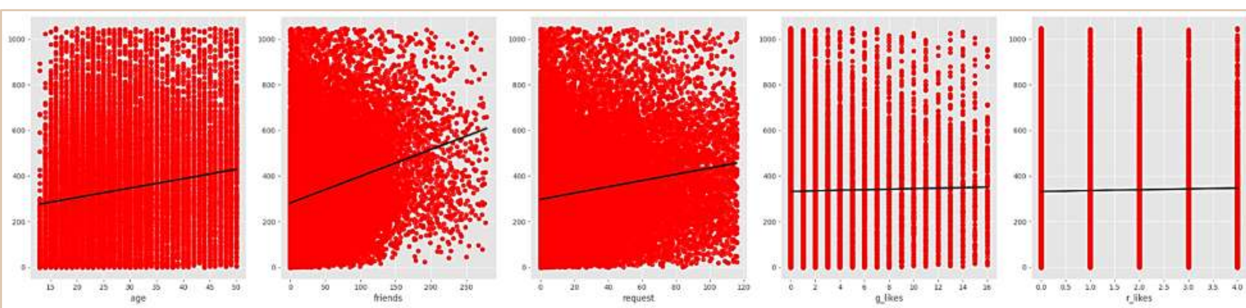
Correlation Analysis

In this section, I'm going to see whether there is relationship between variables with tenure. Since tenure is very crucial for each social media and the companies want to have more loyal users, it makes sense that we see how many other factors are correlated to tenure. For doing this, we should consider two different ways. One way is correlation between numeric variables with tenure, and the second way is correlation between categorical variables with tenure. For the first one, we use Pearson correlation and for the second one, we use ANOVA test.

1. Pearson Correlation

First, based on OLS method, we analyze the correlation and regression between two variables on plot.

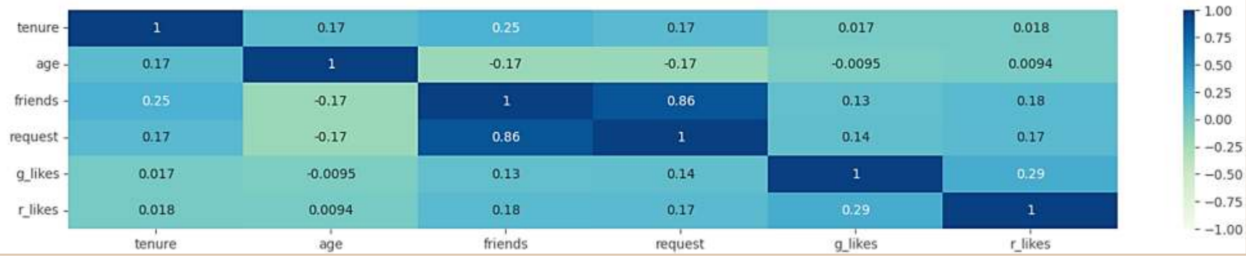
```
#declare numeric variable
numeric = ['age','friends','request','g_likes','r_likes']

# Create a grid of subplots
fig, axes = plt.subplots(1, 4, figsize=(25, 6))

# Flatten the axes array to make it 1D
axes = axes.ravel()

# Loop through each subplot and plot sns.regplot
for i, col in enumerate(numeric):
    sns.regplot(x=col, y=df['tenure'], data=df, ax=axes[i], scatter_kws={"color": "red"}, line_kws={"color":"black"})
    axes[i].set_xlabel(col)
    axes[i].set_ylabel('')

# Adjust spacing between subplots
plt.tight_layout()

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



The scatter plots show:

- When users age, you have more tenure.
- When users have more friends, they prefer to stay on the Facebook.
- When users send more friendship requests, they have more tenure.
- However, there is no relationship between giving more like or receiving like to other people and more or less tenure.

```python
pearson_cor = df[['tenure' , 'age','friends','request','g_likes', 'r_likes']].corr(method = 'pearson')

plt.figure(figsize = (18,3))

sns.heatmap(pearson_cor,
            vmin = -1,
            vmax = 1,
            cmap = "GnBu",
            annot = True)
plt.show()
```



2. ANOVA Test

For categorical variables, first we should see whether a particular categorical variable impacts tenure or not (it is significant or not). For doing this, we run ANOVA test to compare the meaningful difference between means. After that, for the variables that are significant, we run pairwise descriptive analysis for all labels in the particular categorical variable and then compare their impacts on the tenure.

```python
cat_list = ['gender']

import statsmodels.api as sm
from statsmodels.formula.api import ols

for i in cat_list:
    formula = 'tenure ~ {}'.format(i)
    model = ols(formula, data=df).fit()
    anova = sm.stats.anova_lm(model, typ=2)
    p_value = anova.iloc[0,3]

    print('P-value ~ {}: {}'.format(i , p_value))
```

```
P-value ~ gender: 0.000868011156315235
```

According to results, we can come up that gender is significant to explain the tenure, because the p-value is less than 0.05 and we reject null hypothesis. So, in the next step, we want to see for each label in the above categorical variables, which of them has the most impact on tenure.

```
def mean_pairwise(cat_var):
    mean_by = df.groupby(cat_var)['tenure'].mean()
    mean_by = pd.DataFrame(mean_by)
    mean_by = mean_by.sort_values(by=['tenure'], inplace=False, ascending=False)

    return mean_by
```

```
mean_pairwise('gender')
```

gender	tenure
female	343.317820
male	332.401851

Being female would cause to have more tenure.

Ad-Hoc Business Questions

In this section we want to analyze data in Python to answer some ad-hoc questions that might be asked in daily-basis business.

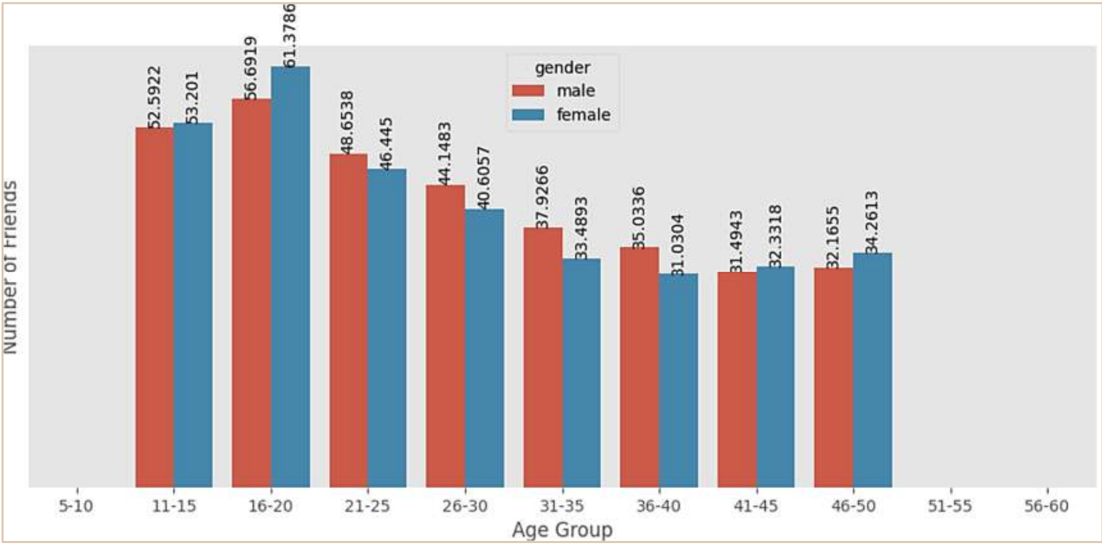
1. Based on age group, which gender does have more friends?

First, we should bin age into some groups.

```
labels = ['5-10','11-15','16-20','21-25','26-30', '31-35', '36-40', '41-45', '46-50', '51-55', '56-60']

df['age_group'] = pd.cut(df['age'], bins = np.arange(5,61,5), labels = labels, right = True)
```

now let's answer the question:

```
fig, ax = plt.subplots(figsize=(12, 5))

sns.barplot(
    x=df['age_group'],
    y=df['friends'],
    hue=df['gender'],
    ci=None,
    ax=ax)

#setting of axis
plt.ylabel("Number of Friends")
plt.xlabel("Age Group")
plt.yticks([])

#add data lable
for container in ax.containers:
    ax.bar_label(container, label_type='edge', fontsize = 10, rotation=90)

#remove the border
for spine in ax.spines.values():
    spine.set_visible(False)

plt.show()
```



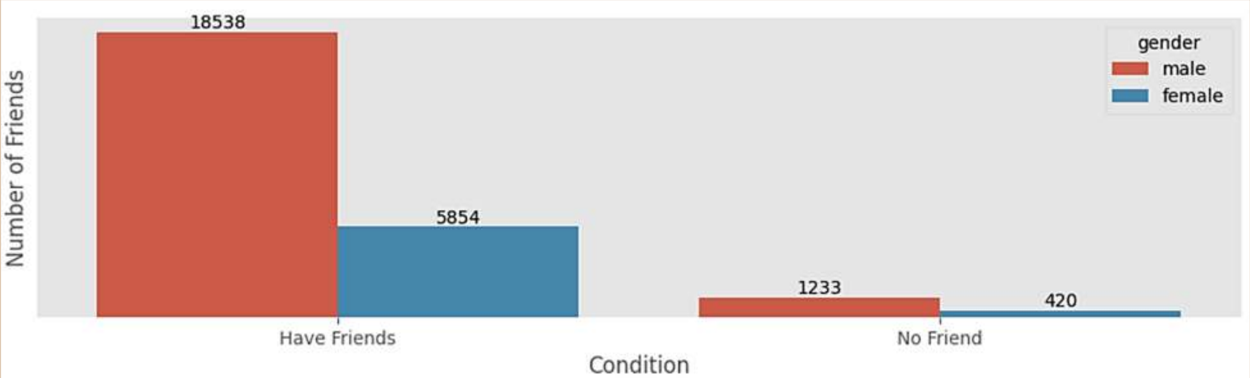
2. How many people do not have any friends? (Based on gender)

To begin with, we calculate the number of people who have friends (at least one) and have no friend.

```
df[df['friends'] == 0].value_counts()

userid   age  gender  tenure  friends  request  g_likes  r_likes  age_group
1000183  41   male    362.0   0        0        0        0        41-45        1
1780518  36   male    149.0   0        0        0        0        36-40        1
1786737  34   male    320.0   0        0        0        0        31-35        1
1785638  24   male    686.0   0        0        0        0        21-25        1
1784614  23   female  176.0   0        0        0        0        21-25        1
                                                                              ..
1376108  14   male    24.0    0        0        3        0        11-15        1
1371905  24   male    63.0    0        0        0        0        21-25        1
1371735  17   male    14.0    0        0        0        0        16-20        1
1371065  28   female  892.0   0        0        0        0        26-30        1
2193232  30   male    94.0    0        0        3        2        26-30        1
```

then we can draw chart:

```python
fig, ax = plt.subplots(figsize=(12, 3))

sns.countplot(
    x = fc,
    hue = df['gender'],
    ax = ax)

#setting of axis
plt.ylabel("Number of Friends")
plt.xlabel("Condition")
ax.set_xticklabels(['Have Friends', 'No Friend'])
plt.yticks([])

#add data lable
for container in ax.containers:
    ax.bar_label(container, label_type='edge', fontsize = 10)

#remove the border
for spine in ax.spines.values():
    spine.set_visible(False)

plt.show()
```



3. Who received the most likes from other users?

to answer this question, we just need to sort the dataframe based on received like:

```
df.sort_values(by = 'r_likes', ascending = False)[:5]
```

	userid	age	gender	tenure	friends	request	g_likes	r_likes	age_group
17904	1709858	15	female	454.0	57	42	0	4	11-15
15078	1565886	36	male	293.0	37	29	4	4	36-40
23688	2168112	29	male	403.0	118	110	3	4	26-30
11234	1389640	27	male	67.0	23	17	0	4	26-30
20446	1031310	14	male	112.0	74	63	4	4	11-15

4. For every single user, calculate how many like did they get per day?

first, we calculate like per day:

```
df['likes_per_day'] = df['r_likes'] / df['tenure'].where(df['tenure'] > 0)
```

and then we can sort the data based on like received and then like per day:

```
famous = df.sort_values(by =['r_likes', 'likes_per_day'], ascending = False)[:5]
famous
```

	userid	age	gender	tenure	friends	request	g_likes	r_likes	age_group	likes_per_day
6109	1952466	36	male	1.0	7	7	1	4	36-40	4.000000
4409	1919733	37	female	2.0	3	3	3	4	36-40	2.000000
11101	1811812	16	male	2.0	22	21	0	4	16-20	2.000000
4562	1059026	21	female	3.0	4	4	0	4	21-25	1.333333
1643	1653309	28	male	5.0	0	0	10	4	26-30	0.800000

```
fig, ax = plt.subplots(figsize=(12, 3))

famous.plot(
    x='userid',
    y='likes_per_day',
    kind='bar',
    color='red',
    ax=ax)

#setting of axis
plt.ylabel("")
plt.xlabel("User ID")
plt.yticks([])

#add data lable
for container in ax.containers:
    ax.bar_label(container, label_type='edge', fontsize = 10)

#remove the border
for spine in ax.spines.values():
    spine.set_visible(False)

#remove the legend
ax.get_legend().remove()

plt.show()
```

5. Show the users who are interested in sending friendship request.

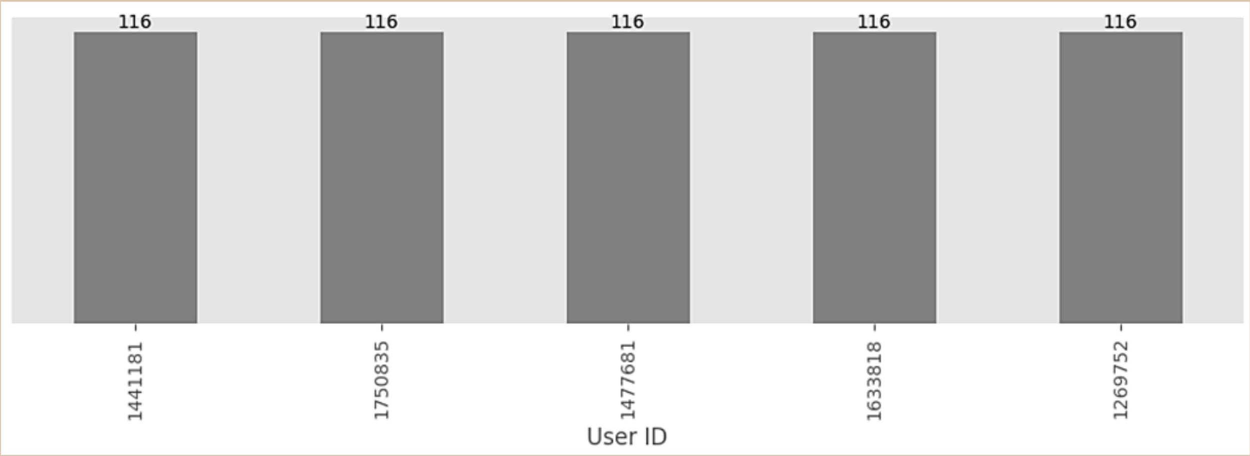
first, we filter dataset based on this condition:

```
followers = df.sort_values(by = 'request', ascending = False)[:5]
followers
```

	userid	age	gender	tenure	friends	request	g_likes	r_likes	age_group	likes_per_day
24243	1441181	21	male	110.0	130	116	0	0	21-25	0.000000
24513	1750835	30	male	222.0	136	116	1	2	26-30	0.009009
24324	1477681	20	female	707.0	131	116	2	0	16-20	0.000000
24697	1633818	28	female	353.0	143	116	1	1	26-30	0.002833
25127	1269752	26	female	345.0	161	116	1	0	26-30	0.000000

Then we can draw the plot.

```
fig, ax = plt.subplots(figsize=(12, 3))

followers.plot(
    x='userid',
    y='request',
    kind='bar',
    color='gray',
    ax=ax)

#setting of axis
plt.ylabel("")
plt.xlabel("User ID")
plt.yticks([])

#add data lable
for container in ax.containers:
    ax.bar_label(container, label_type='edge', fontsize = 10)

#remove the border
for spine in ax.spines.values():
    spine.set_visible(False)

#remove the legend
ax.get_legend().remove()

plt.show()
```


116
116
116
116
116
1441181
1750835
1477681
1633818
1269752
User ID

- END -