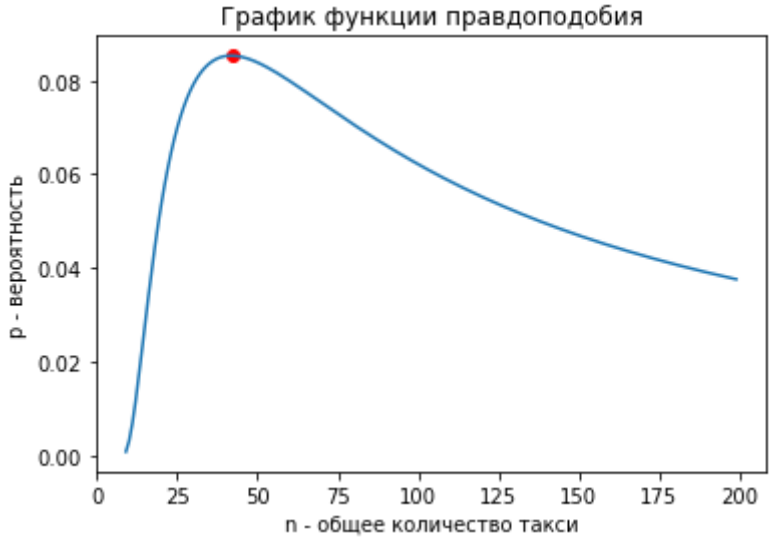
Домашняя работа по статистике

Выполнила Федгинкель Анастасия, БЭК 213

In [2]:
```
import numpy as np
from scipy.stats import expon
import math
from scipy.stats import bootstrap
from scipy.stats import norm
from scipy.stats import t
import pandas as pd
from scipy import stats
import random
import matplotlib.pyplot as plt
from scipy.stats import chi2_contingency
```

Задача 1

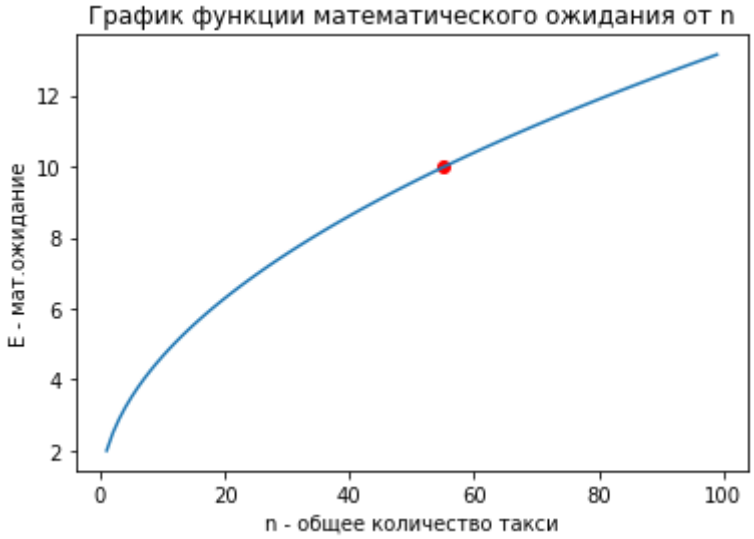
In [2]:
```
#a
def p(x):
    return 9*(x-1)*(x-2)*(x-3)*(x-4)*(x-5)*(x-6)*(x-7)*(x-8)/x**9 #функция правдопод
ps =[]
for i in range(9,200):
    ps.append(p(i)) #находим вероятность при разных значениях n
plt.plot([i for i in range(9,200)], ps)
p = max(ps) #находим максимальную вероятность из получившихся
ind = ps.index(p) #находим количество таксистов, соответствующее максимальному значе
i = [i for i in range(9,200)][ind]
plt.scatter(i,p, c='red')
plt.xlabel('n - общее количество такси')
plt.ylabel('p - вероятность')
plt.title('График функции правдоподобия')
plt.show()
print(f'p = {p}')
print(f'n = {i}')
```




```
p = 0.08525937285627631
n = 42
```

In [3]:
```
#б
def f(x,n): #функция, которая выводит вероятность "повторного" таксиста на x-ый раз
```

```
    m = 1
    for i in range (0, x-1):
        m*=(n-i)
    m*=(x-1)
    m = m/n**x
    return m
ns = []
for n in range(1,100):
    s = 0
    for x in range(2,100):
        s+= x * f(x,n)
    ns.append(s)#мат ожидания в зависимости от n
plt.plot( [n for n in range(1,100)], ns)
n_mm = [n for n in range(1,100)][ns.index(min(ns, key=lambda x: abs(x - 10)))] #нахо
e = min(ns, key=lambda x: abs(x - 10))#                                          ближ
plt.scatter(n_mm, e, c='red')
plt.xlabel('n - общее количество такси')
plt.ylabel('E - мат.ожидание')
plt.title('График функции математического ожидания от n')
plt.show()
print(f'E = {e}')
print(f'n = {n_mm}')
```




```
E = 9.97504888851411
n = 55
```

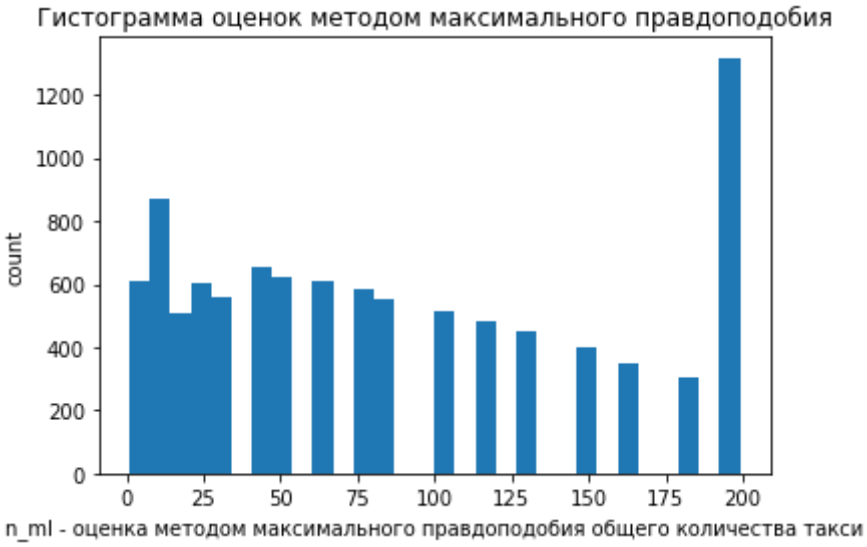
In [4]:

```
#б.1
np.random.seed(42)
ts = []
for i in range(10000): #генерируем 10000 вызовов такси до первого повторяющегося
    t = []
    while True:
            taxi = np.random.choice(ns, 1)
            if taxi not in t:
                t.append(taxi)
            else:
                ts.append(len(t)+1) #сохраняем номер повторного
                break
```

In [5]:

```
def p(x, n): #функция, которая выводит вероятность повторного таксиста на x-ый раз n
    return ((x-1)*math.factorial(n-1)/math.factorial(n-x+1))/n**(x-1)
```

In [6]:
```
ps = []
ml = []
for x in ts:
    prob = []
    for n in range(x-1, 200):
        prob.append(p(x,n))
    ps.append(max(prob)) #находим 10000 вероятностей
    ml.append([i for i in range(x-1,200)][prob.index(max(prob))])
plt.hist(ml, bins = 30)
plt.xlabel('n_ml - оценка методом максимального правдоподобия общего количества такс
plt.ylabel('count')
plt.title('Гистограмма оценок методом максимального правдоподобия');
#здесь все оценки 199 и выше отнесены в один столбец
```



In [7]:
```
print('Смещение =', np.mean(ml)-100)
print('Дисперсия =', np.sum((np.array(ml)-100)**2)/10000)
print('Среднеквадратическая ошибка =', math.sqrt(np.sum((np.array(ml)-100)**2)/10000
```

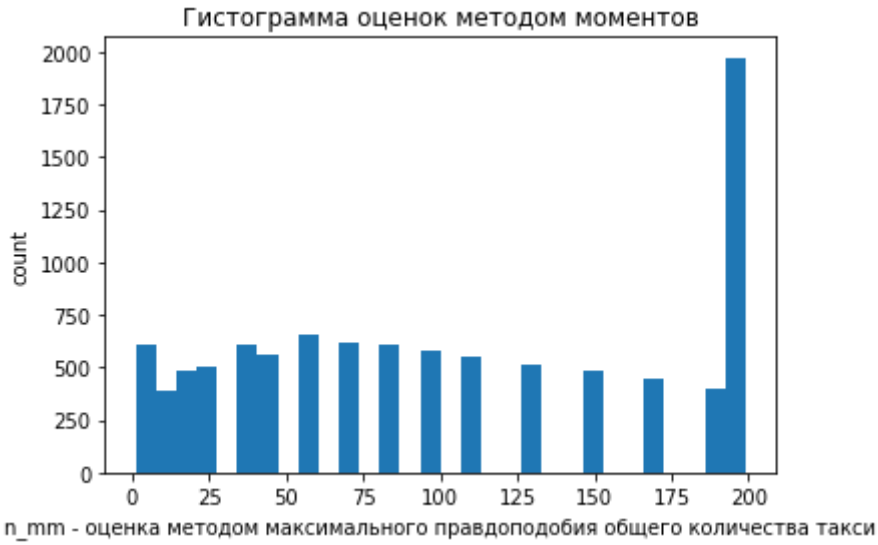
```
Смещение = -15.730500000000006
Дисперсия = 4485.8501
Среднеквадратическая ошибка = 66.97648915850993
```

In [8]:
```
#в.2 - использую данные из в.1
def f(x,n): #функция, которая выводит вероятность "повторного" таксиста на x-ый раз
    m = 1
    for i in range (0, x-1):
        m*=(n-i)
    m*=(x-1)
    m = m/n**x
    return m

ns = [n for n in range(1,200)]
es = []
for n in range(1,200):
    s = 0
    for x in range(2,200):
        s+= x * f(x,n)
    es.append(s)#мат ожидания в зависимости от n

mm =  []
for t in ts:
    mm.append(ns[es.index(min(es, key=lambda x: abs(x - t)))]) #для всех мат.ожидани
```

```
plt.hist(mm, bins = 30)
plt.xlabel('n_mm - оценка методом максимального правдоподобия общего количества такс
plt.ylabel('count')
plt.title('Гистограмма оценок методом моментов');
#все оценки 199 и выше в одном столбце
```



In [9]:

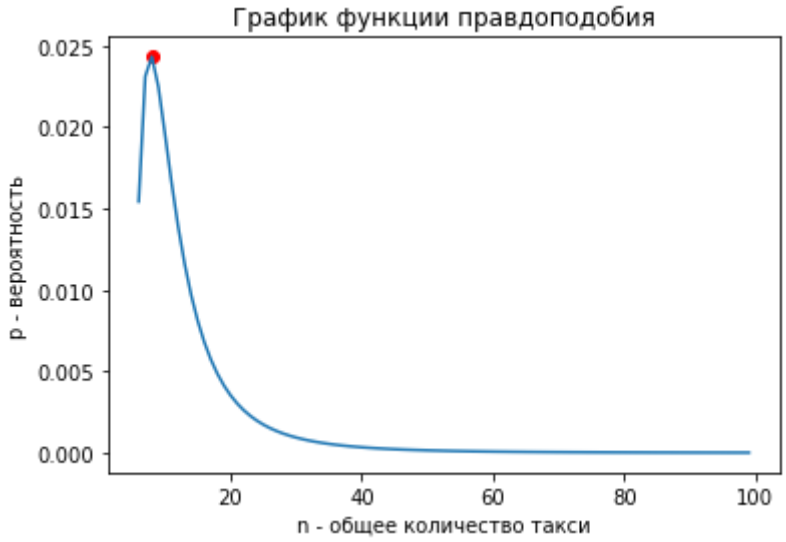
```
print('Смещение =', np.mean(mm)-100)
print('Дисперсия =', np.sum((np.array(mm)-100)**2)/10000)
print('Среднеквадратическая ошибка =', math.sqrt(np.sum((np.array(mm)-100)**2)/10000
```

```
Смещение = 0.712299999999999
Дисперсия = 4786.3757
Среднеквадратическая ошибка = 69.18363751639545
```

Задача 2

In [10]:

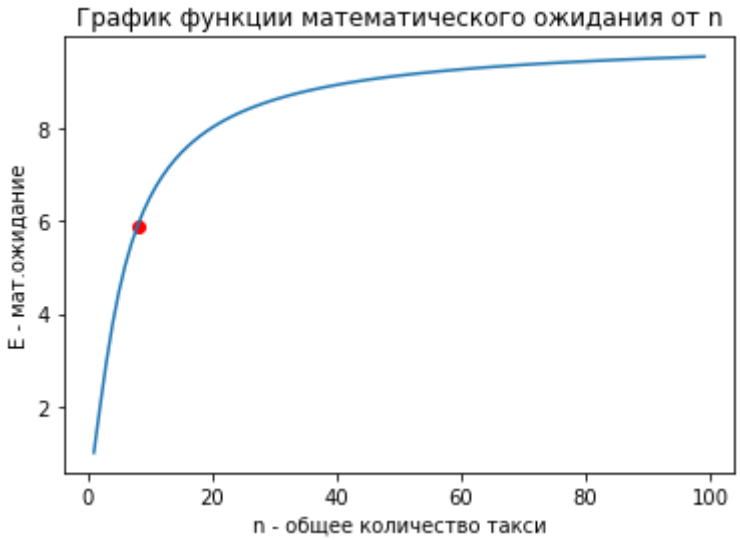
```
#a
def p(n): #функция, которая находит вероятность
    return n/n * (n-1)/n * (n-2)/n * (n-3)/n * (n-4)/n * (n-5)/n * 6**4/n**4
ps =[]
for i in range(6,100):
    ps.append(p(i)) #генерируем вероятности для n от 6 до 99 (мненьше 6 быть не може
plt.plot([i for i in range(6,100)], ps) #график функции правдоподобия
p = max(ps) #ищем максимальную вероятность
ind = ps.index(p)
i = [i for i in range(6,100)][ind] #номер такси, соответствующий максиальной вероятн
plt.scatter(i,p, c='red')
plt.xlabel('n - общее количество такси')
plt.ylabel('p - вероятность')
plt.title('График функции правдоподобия')
plt.show()
print(f'p = {p}')
print(f'n = {i}')
```




```
p = 0.02433300018310547
n = 8
```

In [11]:

```
# б
def E_t(n):
    return (1-(1-1/n)**10)*n #функция, которая находит мат. ожидание
es = []
for i in range(1, 100):
    es.append(E_t(i)) #находим мат. ожидания для n от 1 до 99
plt.plot([i for i in range(1,100)],es)
n_mm = [n for n in range(1,100)][es.index(min(es, key=lambda x: abs(x - 6)))] #наход
e = min(es, key=lambda x: abs(x - 6))#                                          ближе
plt.scatter(n_mm, e, c='red')
plt.xlabel('n - общее количество такси')
plt.ylabel('E - мат.ожидание')
plt.title('График функции математического ожидания от n')
plt.show()
print(f'E = {e}')
print(f'n = {n_mm}')
```




```
E = 5.895395390689373
n = 8
```

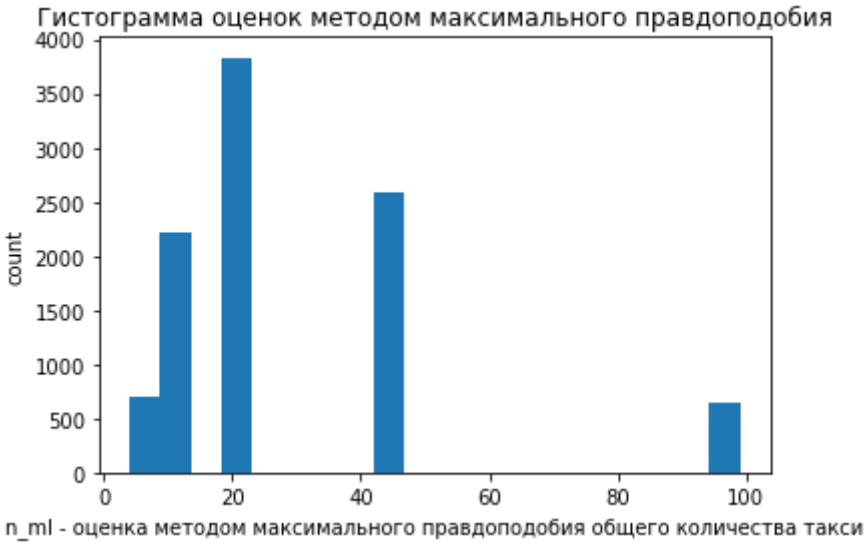
In [3]:

```
#в.1
np.random.seed(42)
taxi = [t for t in range (1,21)] #все возможные таксисты
tx=[]
for i in range(10000): #генерируем 10000 вызовов такси по 10 раз
```

```
    ts = np.random.choice(taxi, 10, True)
    tx.append(len(set(ts))) #сохраняем количество имен
```

In [13]:

```
def p(n,x):
    pr=1
    for i in range(x):
        pr*=(n-i)
    pr*=(1/n)**10
    pr*=x**(10-x)
    return pr #функция правдоподобия от количества имен и количества n
ml=[]
ns=[n for n in range(1,100)]
for x in tx: #для каждого количества имен из сгенерированного списка создаем новую ф
    probs=[] #наибольшую вероятность и n, соответствующее этой вероятности
    for n in ns:
        probs.append(p(n,x))
    ind=probs.index(max(probs))
    ml.append(ns[ind])
plt.hist(ml, bins=20)
plt.xlabel('n_ml - оценка методом максимального правдоподобия общего количества такс
plt.ylabel('count')
plt.title('Гистограмма оценок методом максимального правдоподобия');
```



In [14]:

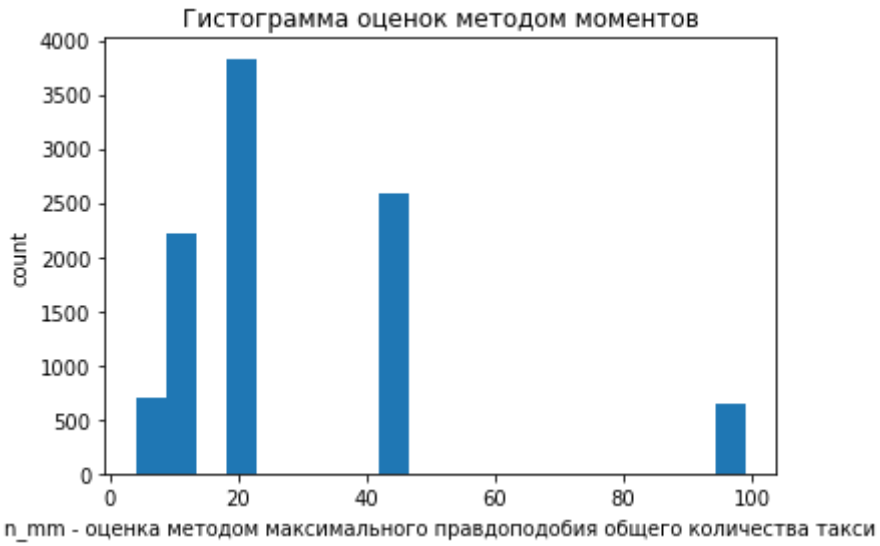
```
print('Смещение =', np.mean(ml)-20)
print('Дисперсия =', np.sum((np.array(ml)-20)**2)/10000)
print('Среднеквадратическая ошибка =', math.sqrt(np.sum((np.array(ml)-20)**2)/10000)
```

```
Смещение = 7.8203
Дисперсия = 557.1171
Среднеквадратическая ошибка = 23.603328155156426
```

In [15]:

```
#в.2
def E_t(n):
    return (1-(1-1/n)**10)*n #функция, которая находит мат.ожидание
mm=[]
es=[]
for n in ns:
    es.append(E_t(n))
for t in tx:#для каждого количества имен из сгенерированного списка выбираем то n, г
                #ближе всего к t-му количеству имен
    n_mm = [n for n in range(1,100)][es.index(min(es, key=lambda x: abs(x - t)))]
    mm.append(n_mm)
plt.hist(mm, bins=20)
```

```
plt.xlabel('n_mm - оценка методом максимального правдоподобия общего количества такс
plt.ylabel('count')
plt.title('Гистограмма оценок методом моментов');
```



In [16]:
```
print('Смещение =', np.mean(mm)-20)
print('Дисперсия =', np.sum((np.array(mm)-20)**2)/10000)
print('Среднеквадратическая ошибка =', math.sqrt(np.sum((np.array(mm)-20)**2)/10000)
```

```
Смещение = 8.209499999999998
Дисперсия = 556.5543
Среднеквадратическая ошибка = 23.591403095195503
```

Задача 3

In [4]:
```
#a
fact_E = 1
z = norm.ppf(0.975)
```

In [5]:
```
#a1 - асимптотический ДИ
np.random.seed(42)
good = 0
for i in range(10000):
    rvs = expon.rvs(size = 20, scale = 1)
    mean = rvs.mean()
    std = rvs.std()
    if (mean - z*std/math.sqrt(20))<fact_E and (mean + z*std/math.sqrt(20))>fact_E:#
        good +=1
good/10000 #делим результаты, где ДИ покрыл мат. ожидание, на все
```

Out[5]:
```
0.8971
```

In [19]:
```
#a2 - наивный бутстрэп
np.random.seed(42)
good = 0
for i in range(10000):
    rvs = expon.rvs(size = 20, scale = 1)
    values = []
    for i in range(1000): #бутстрапируем
        bootstrap_values = np.random.choice(rvs, 20, True)
        values.append(bootstrap_values.mean())
    quantile_l = np.quantile(values, 0.025)
```

```
    quantile_r = np.quantile(values, 0.975)
    if quantile_l<fact_E and quantile_r>fact_E: #смотрим, попадает ли мат.ожидание м
        good+=1
good/10000
```

Out[19]: 0.902

In [20]:
```
#а3 - бутстрэп t-статистики
np.random.seed(42)
good = 0
for i in range(10000):
    rvs = expon.rvs(size = 20, scale = 1)
    mean = rvs.mean()
    se = rvs.std()/math.sqrt(20)
    values = []
    for i in range(1000):
        new_rvs = np.random.choice(rvs, 20, True)
        se_new = new_rvs.std()/math.sqrt(20)
        new_mean = new_rvs.mean()
        R = (new_mean - mean)/se_new
        values.append(R)
    quantile_l = np.quantile(values, 0.025)
    quantile_r = np.quantile(values, 0.975)
    if mean - quantile_l*se>fact_E and mean - quantile_r * se<fact_E: #смотрим, попа
        good+=1
good/10000
```

Out[20]: 0.9445

In [6]:
```
#б (то же, что в п.а, только с t-распределением)
fact_E = 0
```

In [11]:
```
#б1 - асимптотический ДИ
np.random.seed(42)
good = 0
for i in range(10000):
    rvs = t.rvs(size = 20, df = 3)
    mean = rvs.mean()
    std = rvs.std()
    if (mean - z*std/math.sqrt(20))<fact_E and (mean + z*std/math.sqrt(20))>fact_E:
        good +=1
good/10000
```

Out[11]: 0.9359

In [27]:
```
#б2 - наивный бустрэп
np.random.seed(42)
good = 0
for i in range(10000):
    rvs = t.rvs(size = 20, df = 3)
    values = []
    for i in range(1000):
        bootstrap_values = np.random.choice(rvs, 20, True)
        values.append(bootstrap_values.mean())
    quantile_l = np.quantile(values, 0.025)
    quantile_r = np.quantile(values, 0.975)
    if quantile_l<fact_E and quantile_r>fact_E:
        good+=1
good/10000
```

Out[27]:
```
0.9205
```

In [28]:
```
#б3 - бутстрэп t-статистики
np.random.seed(42)
good = 0
for i in range(10000):
    rvs = t.rvs(size = 20, df = 3)
    mean = rvs.mean()
    se = rvs.std()/math.sqrt(20)
    values = []
    for i in range(1000):
        new_rvs = np.random.choice(rvs, 20, True)
        se_new = new_rvs.std()/math.sqrt(20)
        new_mean = new_rvs.mean()
        R = (new_mean - mean)/se_new
        values.append(R)
    quantile_l = np.quantile(values, 0.025)
    quantile_r = np.quantile(values, 0.975)
    if mean - quantile_l*se>fact_E and mean - quantile_r * se<fact_E:
        good+=1
good/10000
```

Out[28]:
```
0.925
```

в) В первом случае бутстрэп t-статистики показал наилучший результат. Однако во втором лучше всего сработал асимптотический ДИ. Возможно, дело в том, что надо делать больше бутстрапированных выборок, но, честно говоря, это очень долго(

Задача 4

In [2]:
```
data = pd.read_csv('для стата.csv', sep = ';')
data
```

Out[2]:

	Фамилия	Балл
0	Репенкова	16
1	Ролдугина	0
2	Сафина	19
3	Сидоров	26
4	Солоухин	21
...	...	...
327	Сенников	19
328	Ся	0
329	Сятова	0
330	Темиркулов	0
331	Эшмеев	16

332 rows × 2 columns

In [3]:
```
#создадим колонку, в которой отобразим, начинается ли фамилия с гласной буквы (1-нач
def starts_with_vowel(word):
    vowels =  ["а", "о", "у", "ы", "и", "е", "ё", "ю", "я","э"]
    if word[0].lower() in vowels:
        return 1
    else:
        return 0
data['Starts_with_vowel'] = data['Фамилия'].apply(starts_with_vowel)
data
```

Out[3]:

	Фамилия	Балл	Starts_with_vowel
0	Репенкова	16	0
1	Ролдугина	0	0
2	Сафина	19	0
3	Сидоров	26	0
4	Солоухин	21	0
...	...	...	...
327	Сенников	19	0
328	Ся	0	0
329	Сятова	0	0
330	Темиркулов	0	0
331	Эшмеев	16	1

332 rows × 3 columns

In [19]:
```
#а - тест Уэлча
gl = data[data['Starts_with_vowel']==1]['Балл'].values
sogl = data[data['Starts_with_vowel']==0]['Балл'].values
p_value = stats.ttest_ind(gl, sogl, equal_var = False)[1]
p_value #p_value > alpha=0.05 => нулевая гипотеза не отвергается
```

Out[19]:
```
0.3974027153843839
```

In [32]:
```
#б- наивный бутстрэп
#проверяем гипотезу о том, что разница равна 0
np.random.seed(42)
diff = sogl.mean()-gl.mean()#находим разницу средних
diffs = []
for i in range(10000):
    sogl_values = np.random.choice(sogl, len(sogl),True)
    gl_values = np.random.choice(gl, len(gl), True)
    sogl_mean = sogl_values.mean()
    gl_mean = gl_values.mean()
    diffs.append(sogl_mean - gl_mean)#с помощью бутстрэпа генерируем новые разности
l,r = np.percentile(diffs, [2.5, 97.5])# находим перцентили
if 0 > l and 0 < r:
    print('Гипотеза не отвергается')
else:
    print('Гипотеза отвергается')
```

```
Гипотеза не отвергается
```

In [33]:
```
p_value = 2*np.sum(np.array(diffs)<0)/10000
p_value
```

Out[33]:
```
0.3984
```

In [34]:
```
#в - бутстрэп t-статистики
np.random.seed(42)
Rs = []
sogl_values = np.random.choice(sogl, len(sogl),True)
sogl_mean = sogl_values.mean()
sogl_se = sogl_values.std()
gl_values = np.random.choice(gl, len(gl), True)
gl_mean = gl_values.mean()
gl_se = gl_values.std()
diff_ = sogl_mean - gl_mean
for i in range(10000): #генерируем новые R*
    new_sogl = np.random.choice(sogl_values, len(sogl),True)
    new_gl = np.random.choice(gl_values, len(gl),True)
    new_sogl_mean = new_sogl.mean()
    new_gl_mean = new_gl.mean()
    new_sogl_se = new_sogl.std()
    new_gl_se = new_gl.std()
    se = math.sqrt((new_sogl_se**2)/len(sogl)+(new_gl_se)**2/len(gl))
    Rs.append(((new_sogl_mean-new_gl_mean) - (sogl_mean - gl_mean)) / se)
l,r = np.percentile(Rs, [2.5, 97.5])
se = math.sqrt((sogl_se**2)/len(sogl)+(gl_se)**2/len(gl))
if 0 > diff_ - se* r and 0 < diff_ - se*l:
    print('Гипотеза не отвергается')
else:
    print('Гипотеза отверггается')
```

```
Гипотеза не отвергается
```

In [35]:
```
p_value = 2*np.sum(np.array(Rs)>0)/10000
p_value
```

Out[35]:
```
0.9986
```

In [36]:
```
#г - перестановочный тест
np.random.seed(42)
diffs = []
for i in range(10000):
    per = np.random.permutation(data['Балл'])
    per_sogl = per[data['Starts_with_vowel']==0]
    per_gl = per[data['Starts_with_vowel']==1]
    diffs.append(per_sogl.mean() - per_gl.mean())
l,r = np.percentile(diffs, [2.5, 97.5])
if 0 > l and diff < 0:
    print('Гипотеза не отвергается')
else:
    print('Гипотеза отвергается')
```

```
Гипотеза отвергается
```

In [38]:
```
p_value = 2*np.sum(np.array(diffs)<0)/10000
p_value
```

Out[38]:
```
0.9948
```

Задача 5

In [4]:
```
# создадим колонку, в которой отобразим, балл выше медианы (1) или нет (0)
data['Median'] = np.where(data['Балл']>=data['Балл'].median(), 1, 0)
data
```

Out[4]:

	Фамилия	Балл	Starts_with_vowel	Median
0	Репенкова	16	0	0
1	Ролдугина	0	0	0
2	Сафина	19	0	1
3	Сидоров	26	0	1
4	Солоухин	21	0	1
...	...	...	...	...
327	Сенников	19	0	1
328	Ся	0	0	0
329	Сятова	0	0	0
330	Темиркулов	0	0	0
331	Эшмеев	16	1	0

332 rows × 4 columns

In [5]:
```
ptable = pd.pivot_table(data, values='Балл', index='Starts_with_vowel', columns='Med
ptable #таблица сопряженности
```

Out[5]:

Median	0	1
Starts_with_vowel		
0	138	145
1	28	21

In [29]:
```
# a - асимптотический ДИ для отношения шансов
sogl_chances = 145/138 #шансы получить хороший балл у людей с фамилией на согласную
gl_chances = 21/28 #шансы людей с фамилией на гласную букву
OR = 1 #предполагаемое отношение шансов
OR_ = gl_chances/ sogl_chances #выборочное отношение
se = math.sqrt(1/138 + 1/145 + 1/28 + 1/21)
l = OR_ * math.exp(-1.96 * se)
r = OR_ * math.exp(1.96 * se)
if OR > l and OR<r : #проверем, попадает ли OR в ДИ
    print('Гипотеза не отвергается')
else:
    print('Гипотеза отвергается')
```

```
Гипотеза не отвергается
```

In [33]:
```
z_obs = norm.cdf((OR_-1)/se)
p_value = z_obs*2
p_value
```

Out[33]:
```
0.3592960710742057
```

б - асимптотический ДИ для отношения вероятностей\

$\widehat{p_{sogl}} = \frac{145}{145+138}$

$\widehat{p_{gl}} = \frac{21}{21+28}$

дельта − метод :

$$\frac{\widehat{p_{gl}}}{\widehat{p_{sogl}}} = \frac{p_{gl}}{p_{sogl}} + \frac{1}{p_{sogl}}(\widehat{p_{gl}} - p_{gl}) + \frac{p_{gl}}{p_{sogl}^2}(\widehat{p_{sogl}} - p_{sogl})$$

$$Var(\frac{\widehat{p_{gl}}}{\widehat{p_{sogl}}}) = \frac{1}{p_{sogl}^2}\frac{p_{gl}(1-p_{gl})}{n_{gl}} + \frac{p_{gl}^2}{p_{sogl}^4}\frac{p_{sogl}(1-p_{sogl})}{n_{sogl}}$$

In [8]:
```
p_sogl_hat = 145/(145+138)
p_gl_hat = 21/(21+28) #выборочные вероятности
teta = p_gl_hat / p_sogl_hat #выборочное отношение вероятностей
var_hat = p_gl_hat*(1-p_gl_hat)/(p_sogl_hat**2 * (21+28)) + p_gl_hat**2 * p_sogl_hat
l = teta - 1.96*math.sqrt(var_hat)
r = teta + 1.96*math.sqrt(var_hat) #границы ДИ
if 1 > l and 1 <r :
    print('Гипотеза не отвергается')
else:
    print('Гипотеза отвергается')
```

```
Гипотеза не отвергается
```

In [17]:
```
z_obs = (teta-1)/math.sqrt(var_hat)
p_value = 2* norm.cdf(z_obs)
p_value
```

Out[17]:
```
0.26338281381111617
```

In [20]:
```
#в - наивный бутстрэп
ot = []
for i in range(10000): #создаем 10000 звезданутых отношений шансов
    boot_sogl = np.random.choice(sogl, len(sogl),True)
    boot_gl = np.random.choice(gl, len(gl),True)
    ch_sogl = np.count_nonzero(boot_sogl >= 17.5)/np.count_nonzero(boot_sogl < 17.5)
    ch_gl = np.count_nonzero(boot_gl >= 17.5)/np.count_nonzero(boot_gl < 17.5)
    ot.append(ch_gl/ch_sogl)
l,r = np.percentile(ot, [2.5, 97.5]) #находим перцентили
if 1 > l and 1 < r:
    print('Гипотеза не отвергается')
else:
    print('Гипотеза отвергается')
```

```
Гипотеза не отвергается
```

In [28]:
```
p_value = 2*(np.array(ot)>1).sum()/10000
p_value
```

Out[28]:
```
0.2612
```

Задача 6

In [34]:
```
# а
# добавим колонку с длинами фамилий
def len_of_surname(word):
    return len(word)
data['Len_of_surname'] = data['Фамилия'].apply(len_of_surname)
data
```

Out[34]:

	Фамилия	Балл	Starts_with_vowel	Median	Len_of_surname
0	Репенкова	16	0	0	9
1	Ролдугина	0	0	0	9
2	Сафина	19	0	1	6
3	Сидоров	26	0	1	7
4	Солоухин	21	0	1	8
...	...	...	...	...	...
327	Сенников	19	0	1	8
328	Ся	0	0	0	2
329	Сятова	0	0	0	6
330	Темиркулов	0	0	0	10
331	Эшмеев	16	1	0	6

332 rows × 5 columns

$E(Y) = \beta E(F) = \beta$ * (сумма всех длин фамилий / количество студентов)

In [35]:
```
#найдем beta
E_Y_mean = sum(data['Балл'])/len(data['Балл'])
E_F_mean = sum(data['Len_of_surname'])/len(data['Len_of_surname'])
beta = E_Y_mean/E_F_mean
beta
```

Out[35]:
```
2.0613026819923372
```

In [36]:
```
#найдем корреляцию
s_Y = math.sqrt(sum((data['Балл'] - E_Y_mean)**2)/(len(data['Балл'])-1))#выборочные
s_F = math.sqrt(sum((data['Len_of_surname'] - E_F_mean)**2)/(len(data['Len_of_surnam
cov = sum((data['Len_of_surname'] - E_F_mean)*(data['Балл'] - E_Y_mean))/(len(data['
corr_obs = cov/(s_Y*s_F)
corr_obs
```

Out[36]:
```
0.025328052669147675
```

In [38]:
```
# б - перестановочный тест
np.random.seed(42)
df = data[['Балл', 'Len_of_surname']]
corrs = []
for i in range(10000):
    df['Перестановка'] = np.random.permutation(df['Балл'])
    E_Y_mean = sum(df['Перестановка'])/len(df['Перестановка'])
    E_F_mean = sum(df['Len_of_surname'])/len(df['Len_of_surname'])
    s_Y = math.sqrt(sum((df['Перестановка'] - E_Y_mean)**2)/(len(df['Перестановка'])
    s_F = math.sqrt(sum((df['Len_of_surname'] - E_F_mean)**2)/(len(df['Len_of_surnam
    cov = sum((df['Len_of_surname'] - E_F_mean)*(df['Перестановка'] - E_Y_mean))/(le
    corr = cov/(s_Y*s_F)
    corrs.append(corr)
plt.hist(corrs, bins = 100)
plt.title('Гистограмма корреляций в бутстрапированных выборках');
```

```
C:\Users\1\AppData\Local\Temp/ipykernel_21604/1333845523.py:6: SettingWithCopyWarnin
g:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/u
ser_guide/indexing.html#returning-a-view-versus-a-copy
  df['Перестановка'] = np.random.permutation(df['Балл'])
```

In [39]:
```
quantile_l = np.quantile(corrs, 0.025)
quantile_r = np.quantile(corrs, 0.975)
quantile_l, quantile_r
p_value = (len([i for i in corrs if i<-corr_obs])+len([i for i in corrs if i>corr_ob
p_value
#p-value достаточно большой, что говорит о том. что мы не можем отвергнуть гипотезу
```

Out[39]: 0.6458

Задача 7

39 Иноагент Иннокентий по 20 наблюдениям строит 95%-й доверительный интервал для математического ожидания несколькими способами: классический асимптотический нормальный
интервал, с помощью наивного бутстрэпа, с помощью бутстрэпа t-статистики.
а) [15] Для каждого способа с помощью 10000 симуляций оцените вероятность того, что номинально 95%-й доверительный интервал фактически накрывает математическое ожидание,
если наблюдения распределены экспоненциально с интенсивностью 1.

Для каждого из трех способов построения доверительного интервала (классический асимптотический нормальный интервал, наивный бутстрэп и бутстрэп t-статистики) можно оценить вероятность накрытия математического ожидания, используя симуляции.

Для симуляций нам необходимо генерировать 10000 выборок экспоненциального распределения с интенсивностью 1 и строить для каждой выборки доверительный интервал. Затем мы будем смотреть, какая доля сгенерированных доверительных интервалов накрывает истинное математическое ожидание.

Давайте рассмотрим каждый способ построения доверительного интервала по отдельности.

Regenerate response

1. Классический асимптотический нормальный интервал:
   - Генерируем 10000 выборок экспоненциального распределения с интенсивностью 1.
   - Для каждой выборки вычисляем выборочное среднее и стандартное отклонение.
   - Строим доверительный интервал, используя формулу: выборочное среднее ± Z * (стандартное отклонение / sqrt(n)), где Z - критическое значение стандартного нормального распределения для 95%-го доверительного интервала, n - размер выборки.
   - Проверяем, сколько из 10000 доверительных интервалов накрывают истинное математическое ожидание.
2. Наивный бутстрэп:
   - Генерируем 10000 выборок экспоненциального распределения с интенсивностью 1.
   - Для каждой выборки вычисляем выборочное среднее.
   - Случайным образом выбираем с повторениями из каждой выборки n элементов, где n - размер выборки.
   - Строим доверительный интервал, используя перцентильный метод: выборочные средние, соответствующие 2.5% и 97.5% перцентилям, представляют собой границы доверительного интервала.
   - Проверяем, сколько из 10000 доверительных интервалов накрывают истинное математическое ожидание.
3. Бутстрэп t-статистики:

Regenerate response

3. Бутстрэп t-статистики:
   - Генерируем 10000 выборок экспоненциального распределения с интенсивностью 1.
   - Для каждой выборки вычисляем выборочное среднее и стандартное отклонение.
   - Случайным образом выбираем с повторениями из каждой выборки n элементов, где n - размер выборки.
   - Строим доверительный интервал, используя бутстрэп t-статистику: выборочное среднее ± t * (стандартное отклонение), где t - критическое значение t-распределения для 95%-го доверительного интервала с (n-1) степенями свободы.
   - Проверяем, сколько из 10000 доверительных интервалов накрывают истинное математическое ожидание.

Интервал считается "накрывающим" математическое ожидание, если истинное математическое ожидание попадает внутрь доверительного интервала.

Пожалуйста, дайте мне некоторое время, чтобы провести вычисления и предоставить результаты для каждого из трех способов.

для каждого способа нужен код на питоне

Конечно! Вот примеры кода на Python для каждого из трех способов построения доверительных интервалов:

1. Классический асимптотический нормальный интервал:

python Copy code

```python
import numpy as np
from scipy.stats import norm

def calculate_confidence_interval(data):
    sample_mean = np.mean(data)
    sample_std = np.std(data, ddof=1)
    n = len(data)
    z_critical = norm.ppf(0.975)  # Значение Z для 95%-го доверительного интер

    margin_of_error = z_critical * (sample_std / np.sqrt(n))
    confidence_interval = (sample_mean - margin_of_error, sample_mean + marg

    return confidence_interval

# Генерация 10000 выборок экспоненциального распределения с интенсивностью 1
simulations = [np.random.exponential(scale=1, size=20) for _ in range(10000)

# Вычисление доверительного интервала для каждой выборки
confidence_intervals = [calculate_confidence_interval(data) for data in simu

# Проверка, сколько доверительных интервалов накрывают математическое ожидание
mean_in_interval = [interval[0] <= 1 <= interval[1] for interval in confiden
probability = sum(mean_in_interval) / len(mean_in_interval)
print("Вероятность накрытия математического ожидания (классический асимптотически
```

Regenerate response

2. Наивный бутстрэп:

python Copy code

```python
import numpy as np

def calculate_confidence_interval(data):
    sample_mean = np.mean(data)
    n = len(data)

    bootstrap_means = [np.mean(np.random.choice(data, size=n, replace=True))
    lower_percentile = np.percentile(bootstrap_means, 2.5)
    upper_percentile = np.percentile(bootstrap_means, 97.5)

    confidence_interval = (lower_percentile, upper_percentile)

    return confidence_interval

# Генерация 10000 выборок экспоненциального распределения с интенсивностью 1
simulations = [np.random.exponential(scale=1, size=20) for _ in range(10000)

# Вычисление доверительного интервала для каждой выборки
confidence_intervals = [calculate_confidence_interval(data) for data in simu

# Проверка, сколько доверительных интервалов накрывают математическое ожидание
mean_in_interval = [interval[0] <= 1 <= interval[1] for interval in confiden
probability = sum(mean_in_interval) / len(mean_in_interval)
print("Вероятность накрытия математического ожидания (наивный бутстрэп):", probab
```

3. Бутстрэп t-статистики:

```python
import numpy as np
from scipy.stats import t

def calculate_confidence_interval(data):
    sample_mean = np.mean(data)
    sample_std = np.std(data, ddof=1)
    n = len(data)
    t_critical = t.ppf(0.975, df=n-1)  # Значение t-статистики для 95%-го дове

    margin_of_error = t_critical * (sample_std / np.sqrt(n))
    confidence_interval = (sample_mean - margin_of_error, sample_mean + marg

    return confidence_interval

# Генерация 10000 выборок экспоненциального распределения с интенсивностью 1
simulations = [np.random.exponential(scale=1, size=20) for _ in range(10000)

# Вычисление доверительного интервала для каждой выборки
confidence_intervals = [calculate_confidence_interval(data) for data in simu

# Проверка, сколько доверительных интервалов накрывают математическое ожидание
mean_in_interval = [interval[0] <= 1 <= interval[1] for interval in confiden
probability = sum(mean_in_interval) / len(mean_in_interval)
print("Вероятность накрытия математического ожидания (бутстрэп t-статистики):", p
```

Обратите внимание, что во всех трех примерах кода предполагается, что выборки экспоненциального распределения с интенсивностью 1 уже сгенерированы.

In [10]:

```python
np.random.seed(42)
import numpy as np
from scipy.stats import norm

def calculate_confidence_interval(data):
    sample_mean = np.mean(data)
    sample_std = np.std(data, ddof=1)
    n = len(data)
    z_critical = norm.ppf(0.975)  # Значение Z для 95%-го доверительного интервала

    margin_of_error = z_critical * (sample_std / np.sqrt(n))
    confidence_interval = (sample_mean - margin_of_error, sample_mean + margin_of_er

    return confidence_interval

# Генерация 10000 выборок экспоненциального распределения с интенсивностью 1
simulations = [np.random.exponential(scale=1, size=20) for _ in range(10000)]

# Вычисление доверительного интервала для каждой выборки
confidence_intervals = [calculate_confidence_interval(data) for data in simulations]

# Проверка, сколько доверительных интервалов накрывают математическое ожидание
mean_in_interval = [interval[0] <= 1 <= interval[1] for interval in confidence_inter
```

```
probability = sum(mean_in_interval) / len(mean_in_interval)
print("Вероятность накрытия математического ожидания (классический асимптотический н
```

```
Вероятность накрытия математического ожидания (классический асимптотический нормальн
ый интервал): 0.9036
```

In [ ]:

```
np.random.seed(42)
import numpy as np
def calculate_confidence_interval(data):
    sample_mean = np.mean(data)
    n = len(data)

    bootstrap_means = [np.mean(np.random.choice(data, size=n, replace=True)) for _ i
    lower_percentile = np.percentile(bootstrap_means, 2.5)
    upper_percentile = np.percentile(bootstrap_means, 97.5)

    confidence_interval = (lower_percentile, upper_percentile)

    return confidence_interval

# Генерация 10000 выборок экспоненциального распределения с интенсивностью 1
simulations = [np.random.exponential(scale=1, size=20) for _ in range(10000)]

# Вычисление доверительного интервала для каждой выборки
confidence_intervals = [calculate_confidence_interval(data) for data in simulations]

# Проверка, сколько доверительных интервалов накрывают математическое ожидание
mean_in_interval = [interval[0] <= 1 <= interval[1] for interval in confidence_inter
probability = sum(mean_in_interval) / len(mean_in_interval)
print("Вероятность накрытия математического ожидания (наивный бутстрэп):", probabili
```

In [ ]:

```
np.random.seed(42)

import numpy as np
from scipy.stats import t

def calculate_confidence_interval(data):
    sample_mean = np.mean(data)
    sample_std = np.std(data, ddof=1)
    n = len(data)
    t_critical = t.ppf(0.975, df=n-1)  # Значение t-статистики для 95%-го доверитель

    margin_of_error = t_critical * (sample_std / np.sqrt(n))
    confidence_interval = (sample_mean - margin_of_error, sample_mean + margin_of_er

    return confidence_interval

# Генерация 10000 выборок экспоненциального распределения с интенсивностью 1
simulations = [np.random.exponential(scale=1, size=20) for _ in range(10000)]

# Вычисление доверительного интервала для каждой выборки
confidence_intervals = [calculate_confidence_interval(data) for data in simulations]

# Проверка, сколько доверительных интервалов накрывают математическое ожидание
mean_in_interval = [interval[0] <= 1 <= interval[1] for interval in confidence_inter
probability = sum(mean_in_interval) / len(mean_in_interval)
print("Вероятность накрытия математического ожидания (бутстрэп t-статистики):", prob
```

Задача 8

Последнее время очень помог этот плейлист https://www.youtube.com/watch?v=2p24KPez62U&list=PLCf-cQCe1FRyCcf47wwBlDObNcz4ud2_L, потому что я совсем запуталась в тестах и критериях. Даже если что-то говорилос на лекциях и семинарах, информация лучше усваивается в моей голове, если я слышу ее в разных прочтениях от разных людей). Вообще канал топ.

Источник вдохновения: https://www.youtube.com/watch?v=wcibw-m8tRk