Содержание задачи.

1. Постановка задачи:

- Описание задачи.
- Информация о предоставленных данных.
- Выделение подзадач.

2. Импорт:

- Импорт основных библиотек
- Импорт данных

3. Первая подзадача (EDA):

- Описание.
- Знакомство с данными, предобработка и визуализация.
- Вывод по данным.
- Описание метода решения кластерного анализа.
- Нормализация предикторов и подбор параметров.
- Получение кластеров и их анализ.
- Вывод.

4. Вторая подзадача (Кредитный скоринг):

- Описание метода решения.
- Импорт библиотек.
- Преобразование данных.
- Выделение целевой переменной и предикторов, разбиение на train и test.
- Построение классификаторов:
    1) Logistic regression
    2) RandomForest
    3) XGboost
- Выдвижение гипотез.
- Проверка гипотез.

5. Вывод:

- Общее заключение.

1. Постановка задачи:

- **Описание задачи :**

  Перед нами стоит задача оценки кредитного риска.

Кредитный скоринг — система оценки кредитоспособности (кредитных рисков) лица, основанная на численных статистических методах. Кредитный скоринг широко используется как крупными банками, микрофинансовыми организациями, так и в потребительском (магазинном) экспресс-кредитовании на небольшие суммы.

Скоринг заключается в присвоении баллов по заполнению некой анкеты, разработанной оценщиками кредитных рисков андеррайтерами. По результатам набранных баллов системой автоматически принимается решение об одобрении или отказе в выдаче кредита.

- **Информация о предоставленных данных :**

  Исходный набор данных содержит 1000 записей с 10 категориальными / символическими атрибутами. В этом наборе данных каждая запись представляет человека, который берет кредит в банке. Каждый человек классифицируется как хороший или плохой кредитный риск в соответствии с набором атрибутов.

  Описание атрибутов: 1) Возраст (числовой)

  2) Пол (текст: мужской, женский)

  3) Работа (числовые: 0 - неквалифицированный и нерезидент, 1 - неквалифицированный и резидент, 2 - квалифицированный, 3 - высококвалифицированный)

  4) Жилье (текст: собственное, арендное или бесплатное)

  5) Сберегательные счета (текст - маленький, средний, довольно богатый, богатый)

  6) Расчетный счет (числовой, в DM - немецкая марка)

  7) Сумма кредита (числовая, в немецких марках)

  8) Продолжительность (числовое значение в месяцах)

  9) Назначение (текст: машина, мебель / техника, радио / ТВ, бытовая техника, ремонт, образование, бизнес, отдых / прочее)

  Целевой признак:

  default - неспособность должника погасить обязательные платежи банку(1 - неспособен выплатить, о - способен выплотить)

- **Выделение подзадач :**
  - Разведочный анализ данных (exploratory data analysis, EDA)
  - Построение моделей бинарной классификации для кредитного скоринга и выбор наилучшей.

2. Импорт:

- **Импорт основных библиотек:**

In [1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import os
import seaborn as sns
```

- Импорт данных:

In [2]:
```
df = pd.read_csv('credit.csv',sep=',')
df.drop('Unnamed: 0',axis=1,inplace=True)
```

3. Первая подзадача (EDA):

- **Описание:**

В данной подзадаче будем производить предварительное исследование датасета с целью определения его основных характеристик, взаимосвязей между признаками, а также для создания модели ML, для реализации задачи кластерного анализа.

Кластерный анализ занимается тем, что позволяет классифицировать многомерные наблюдения и образовывать схожие между собой группы объектов (кластеры). В данном случае будем распределять клиентов на класстеры, для определения целевой аудитории.

- **Знакомство с данными, предобработка и визуализация:**

In [3]:
```
df.head()
```

Out[3]:

	Age	Sex	Job	Housing	Saving accounts	Checking account	Credit amount	Duration	Purpose	default
0	67	male	2	own	NaN	little	1169	6	radio/TV	0
1	22	female	2	own	little	moderate	5951	48	radio/TV	1
2	49	male	1	own	little	NaN	2096	12	education	0
3	45	male	2	free	little	little	7882	42	furniture/equipment	0
4	53	male	2	free	little	little	4870	24	car	1

In [4]:
```
# посмотрим на расзмерность
df.shape
```

Out[4]: (1000, 10)

In [5]:
```
# удалим дубликаты, если такие существуют
df = df.drop_duplicates()
df.shape
```

Out[5]: (1000, 10)

In [6]:
```
#Обработка пропусков
#Стоит помнить, что в случае, если пропусков у признака слишком много (более 70%), m
```

```
df.isnull().sum()
```

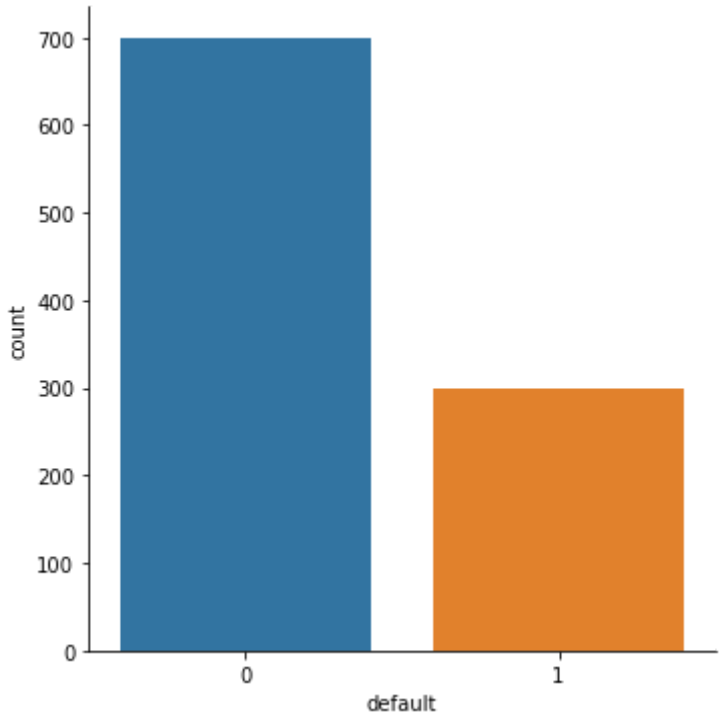
Out[6]:
```
Age                  0
Sex                  0
Job                  0
Housing              0
Saving accounts    183
Checking account   394
Credit amount        0
Duration             0
Purpose              0
default              0
dtype: int64
```

In [7]:
```
 #видим, что по данных сберегательного и расчетного счета достаточного много пропуск
df['Checking account'] = df['Checking account'].fillna('no_info_checking')
df['Saving accounts'] = df['Saving accounts'].fillna('no_info_saving')
```

In [8]:
```
#посмотрим на сбалансированность выборки
print(df['default'].value_counts())
sns.color_palette("Paired")
sns.catplot(x="default", kind="count",data = df )
```

```
0    700
1    300
Name: default, dtype: int64
```

Out[8]:
```
<seaborn.axisgrid.FacetGrid at 0x7fa444faa4a8>
```



В данном случае не наблюдаем существенного дисбаланса классов.

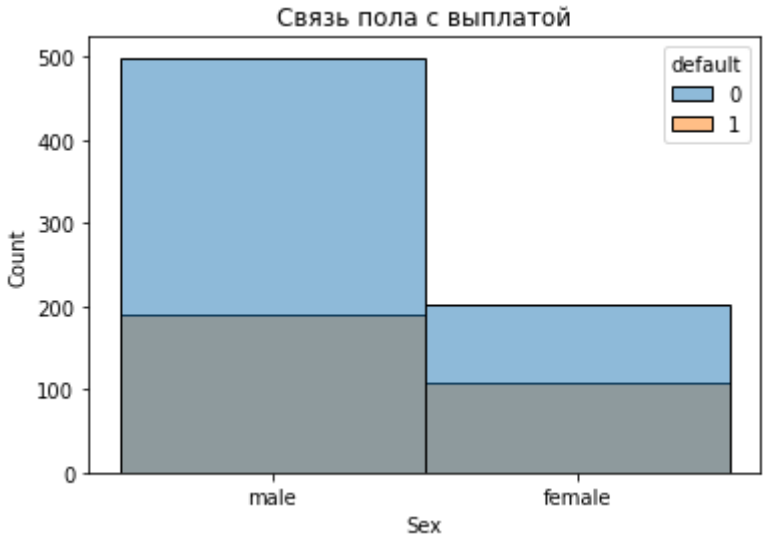
70% - возвратили займ.

30% - не возвратили займ.

Посмотрим на столбцы с номинативными переменными.

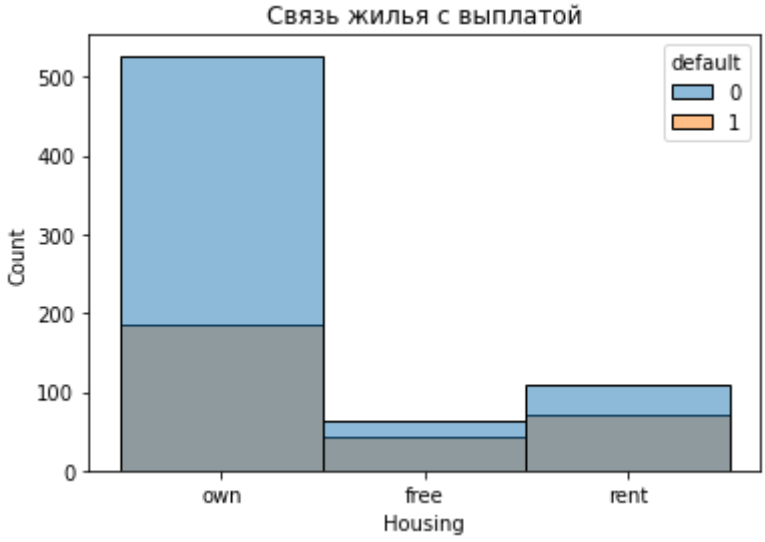
In [9]:
```
ax = sns.histplot(data=df, x="Sex", hue="default")
ax.set_title("Связь пола с выплатой")
```

Out[9]: Text(0.5, 1.0, 'Связь пола с выплатой')



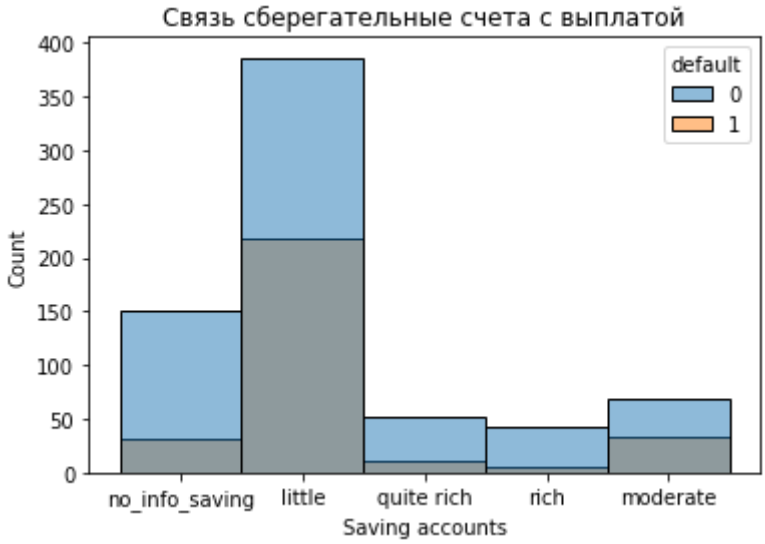
In [10]:
```
ax = sns.histplot(data=df, x="Housing", hue="default")
ax.set_title("Связь жилья с выплатой")
```

Out[10]: Text(0.5, 1.0, 'Связь жилья с выплатой')



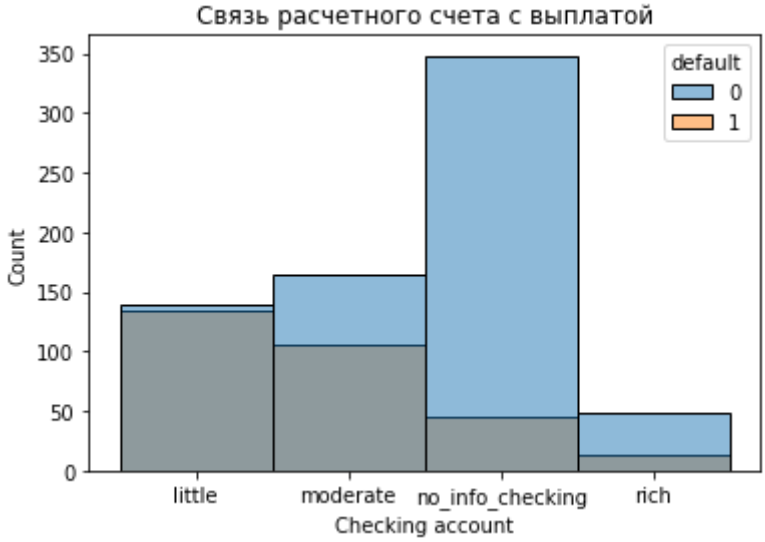
In [11]:
```
ax = sns.histplot(data=df, x="Saving accounts", hue="default")
ax.set_title("Связь сберегательные счета с выплатой")
```

Out[11]: Text(0.5, 1.0, 'Связь сберегательные счета с выплатой')

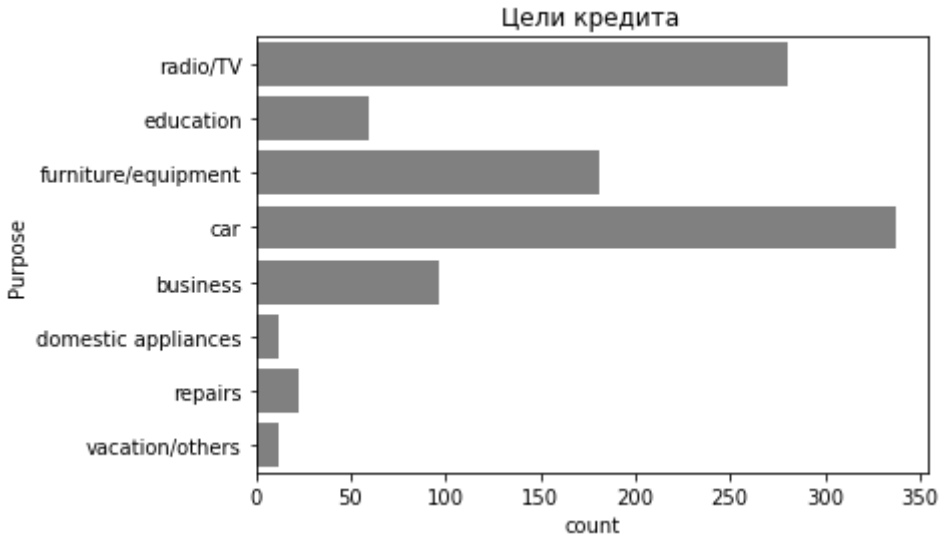


In [12]:
```
ax = sns.histplot(data=df, x="Checking account", hue="default")
ax.set_title("Связь расчетного счета с выплатой")
```

Out[12]:
```
Text(0.5, 1.0, 'Связь расчетного счета с выплатой')
```

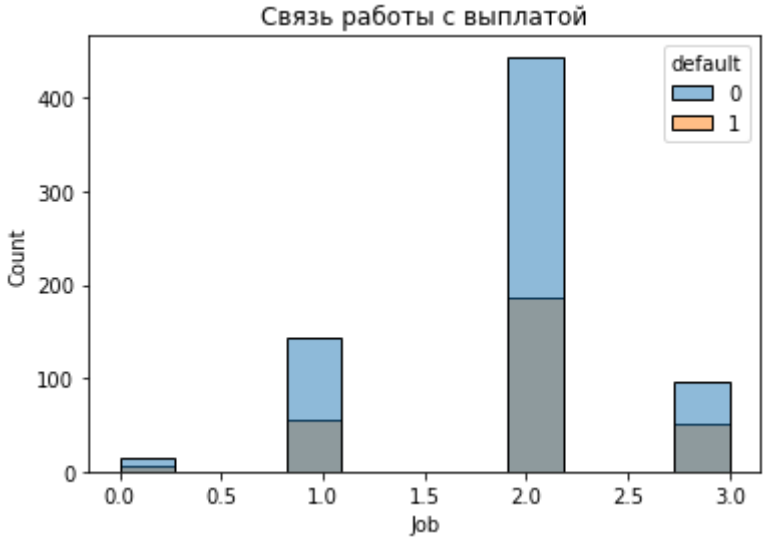


In [13]:
```
sns.countplot(y ="Purpose", data = df,color = "grey",alpha = 1
              )
plt.title("Цели кредита")
plt.show()
```



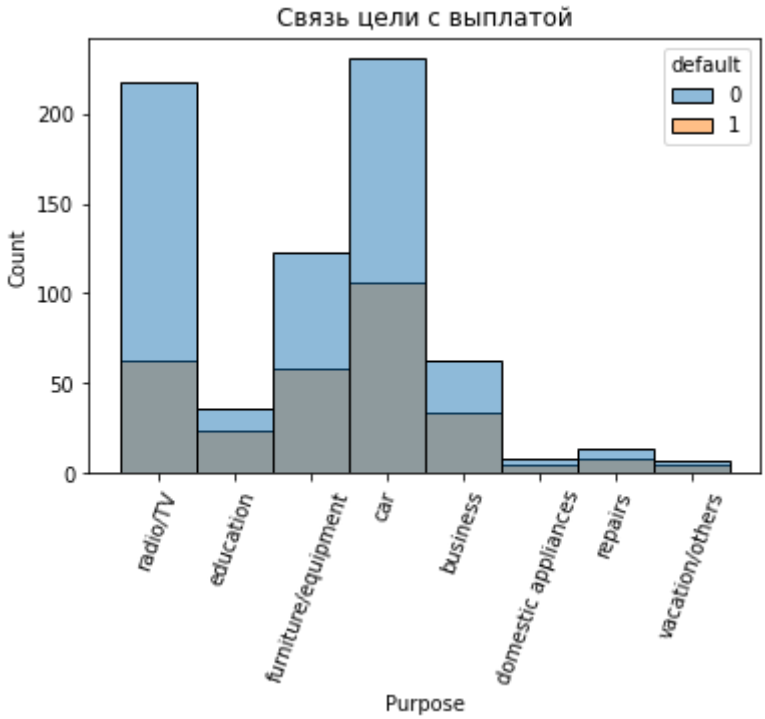
In [14]:
```
ax = sns.histplot(data=df, x="Job", hue="default")
ax.set_title("Связь работы с выплатой")
```

Out[14]:
```
Text(0.5, 1.0, 'Связь работы с выплатой')
```



In [15]:
```
ax = sns.histplot(data=df, x="Purpose", hue="default")
ax.set_title("Связь цели с выплатой")
plt.xticks(rotation=70)
```

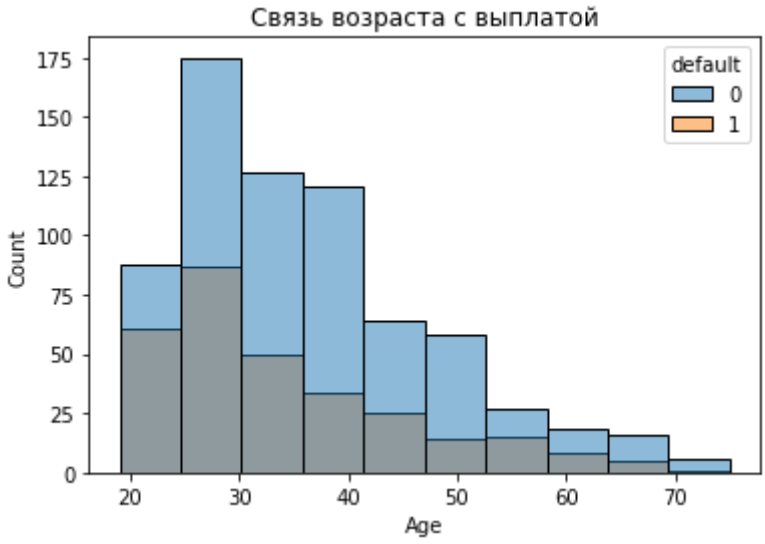
Out[15]:
```
([0, 1, 2, 3, 4, 5, 6, 7],
 [Text(0, 0, ''),
  Text(0, 0, ''),
  Text(0, 0, ''),
  Text(0, 0, ''),
  Text(0, 0, ''),
  Text(0, 0, ''),
  Text(0, 0, ''),
  Text(0, 0, '')])
```



Посмотрим на столбцы с количественной переменной.

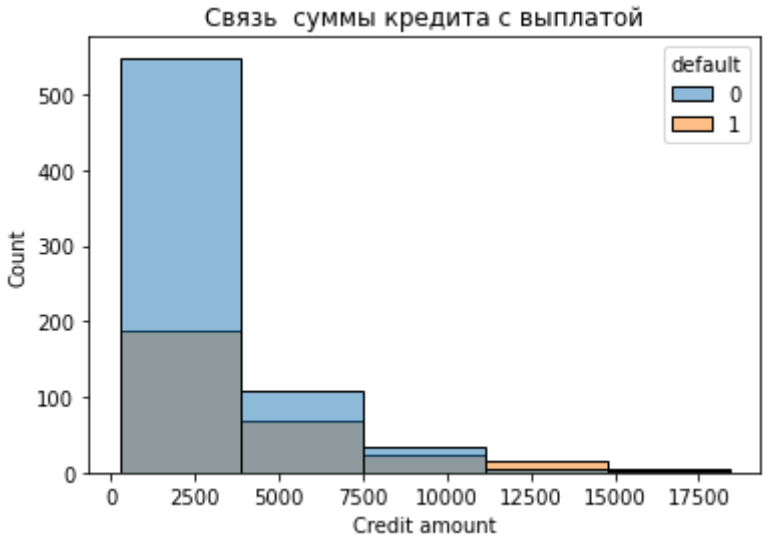
In [16]:
```
ax = sns.histplot(data=df, x="Age", hue="default",bins = 10)
ax.set_title("Связь возраста с выплатой")
```

Out[16]:
```
Text(0.5, 1.0, 'Связь возраста с выплатой')
```



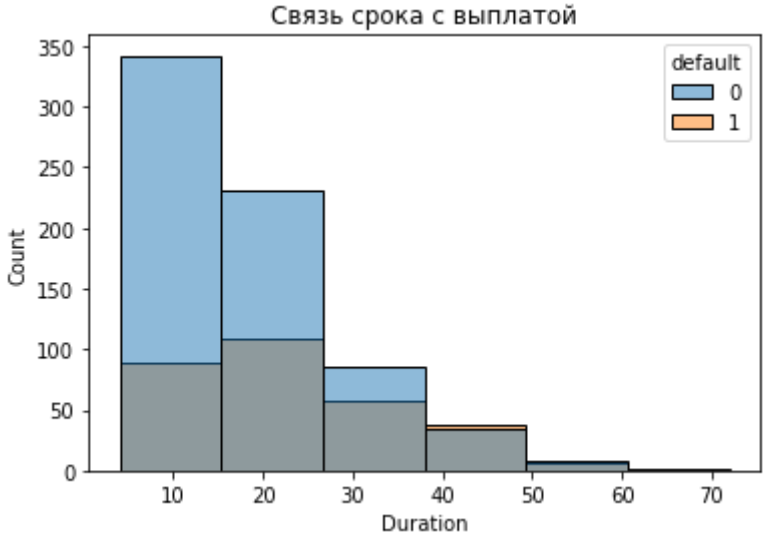
In [17]:
```
ax = sns.histplot(data=df, x="Credit amount", hue="default",bins = 5)
ax.set_title("Связь  суммы кредита с выплатой")
```

Out[17]:
```
Text(0.5, 1.0, 'Связь  суммы кредита с выплатой')
```

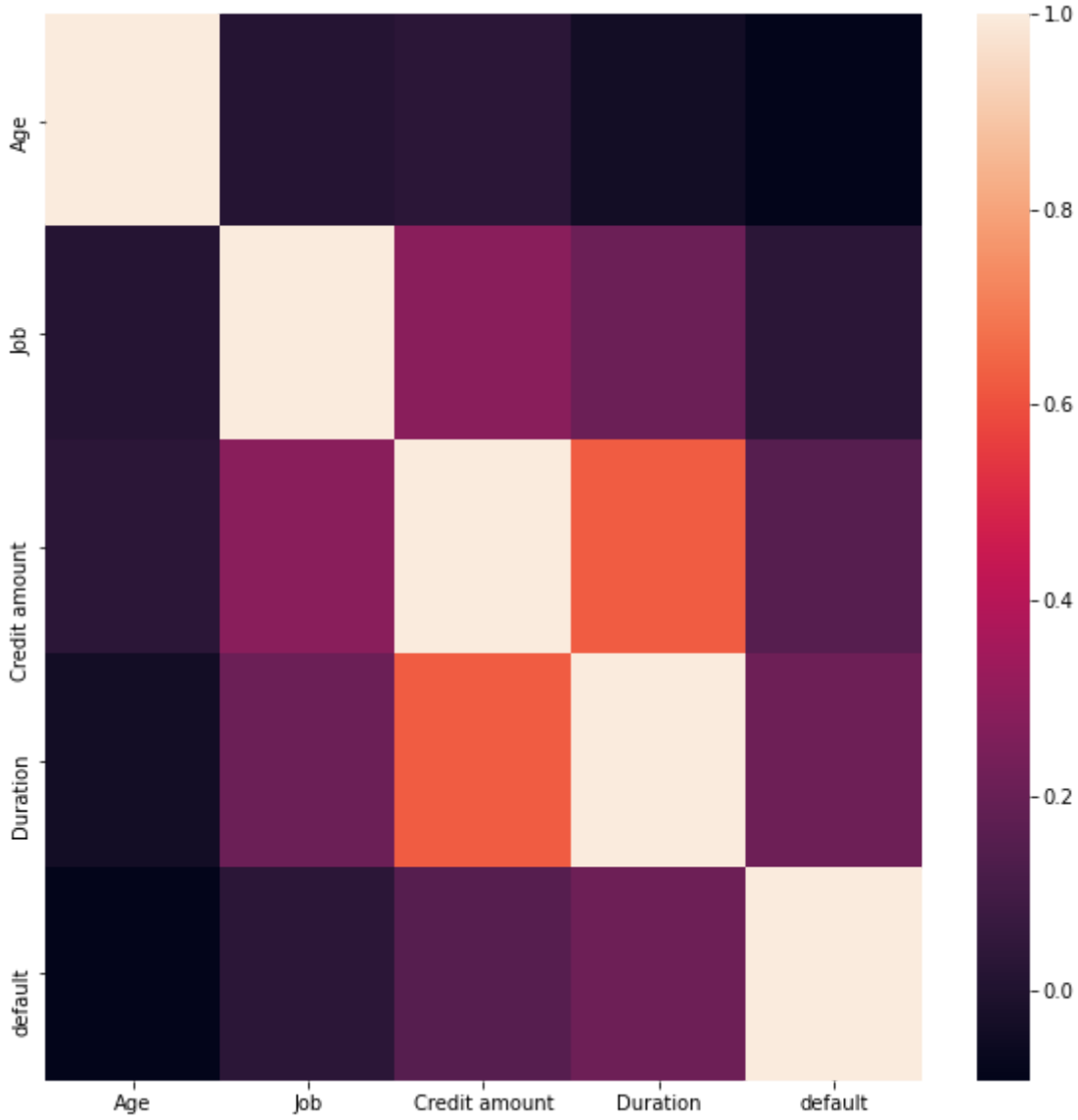


In [18]:
```
ax = sns.histplot(data=df, x="Duration", hue="default", bins = 6)
ax.set_title("Связь срока с выплатой")
```

Out[18]: Text(0.5, 1.0, 'Связь срока с выплатой')



In [19]:
```
plt.figure(figsize=(10,10));
sns.heatmap(df.corr());
```



- Вывод по данным.

1) Большинство клиентов - мужчины. Возврат денежных средств и у мужчин, и у женщин примерно на обном уровне.

2) Также большинство клиентов имеют собственное жилье.Процент возврата у них выше (≈10%).

3) Сберегательный счет у большинства клиентов небольшой.

4) Срок кредита от 4 до 72 месяцев. Чаще всего используются кредиты на год или два.

5) Клиенты в большинстве случаев берут кредит на машины, радио/тв, мебель / техника, бизнес и образование.

6) Продолжительность и сумма кредита коррелируют между собой.

- **Описание метода для решения кластерного анализа.**

  Для реализации кластерного анализа воспользуемся методом k-means. Кратко о данном методе можно почитать тут.

Далее, отбросим целевую переменную, т.к это "обучение без учителя". Также прологарифмируем переменные 'Age' и 'Credit amount', это приведет к нормальному распределению для этих предикторов.

Не будет лишним произвести нормализацию данных. Цель такого преобразования – изменить значения числовых столбцов в наборе данных так, чтобы сохранить различия их диапазонов. В машинном обучении датасет требует нормализации, когда признаки имеют разные диапазоны и тем самым способствуют искажению восприятия взаимоотношений между Переменными-предикторами.

У данного метода есть минусы:

1) Только евклидово расстрояние.

2) Решение зависит от начальных центров.

3) Заранее неизвестно кол-во центроидов

Определять кол-во центроидов будем с помощью построения дендрограммы, используя иерархический класстерный анализ, также построим график "каменистая осыпь" для точного определения кол-во центроидов.

Также нужно незабыть закодировать текст в цифры, иначе алгаритм не будет работать.

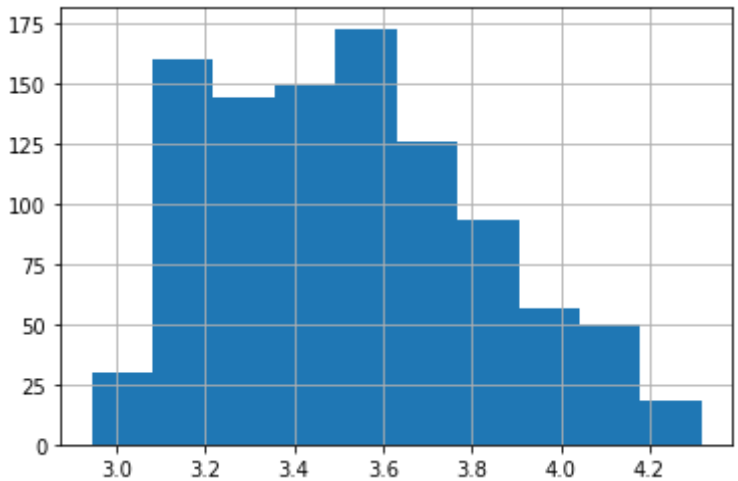
- **Нормализация предикторов и подбор параметров.**

In [20]:
```
df_class = df.copy()
```

In [21]:
```
# удаление целевой переменной
df_class.drop('default', axis=1, inplace=True)
```

In [22]:
```
np.log(df_class['Age']).hist()
```

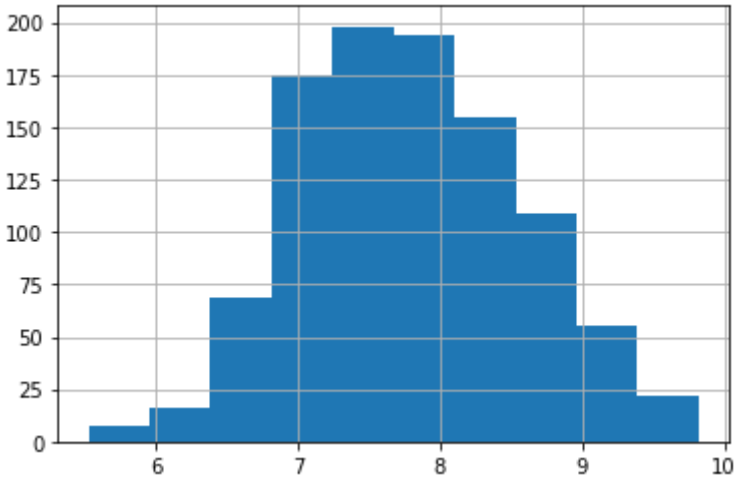
Out[22]: <AxesSubplot:>



In [23]:
```
np.log(df_class['Credit amount']).hist()
```

<AxesSubplot:>

Out[23]:



In [24]:

```
#Преобразуем переменые 'Age' и 'Credit amount'(почти нормальное распределение)
df_class['Age'] = np.log(df_class['Age'])
df_class['Credit amount'] = np.log(df_class['Credit amount'])
df_class.rename({'Age': 'log_age', 'Credit amount': 'log_credit_amount'}, axis=1, in
```

In [25]:

```
# Кодировка
from sklearn.preprocessing import LabelEncoder
encoder = LabelEncoder()
categ = ['Sex','Housing','Saving accounts','Checking account','Purpose']
for label in categ:
    df_class[label] = encoder.fit_transform(df_class[label])
```

In [26]:

```
#Стандартизация
from sklearn import preprocessing
norm = preprocessing.StandardScaler()
norm.fit(df_class)
X = norm.transform(df_class)
X = pd.DataFrame(X, index=df_class.index, columns=df_class.columns)
```

In [27]:

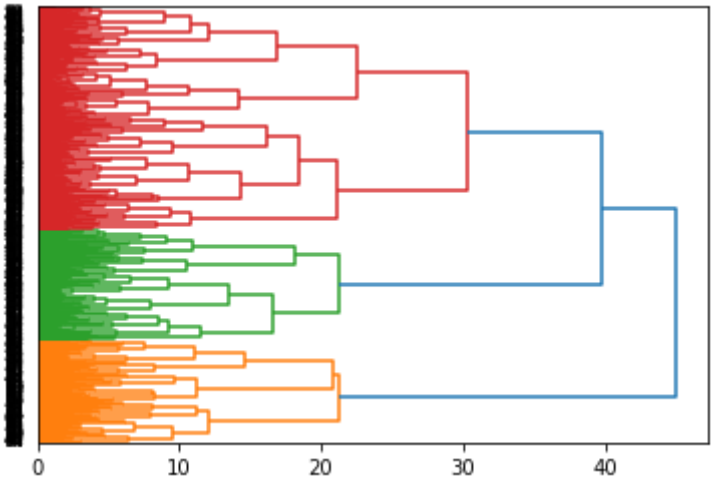
```
# получаем следующий датасет:
X.head()
```

Out[27]:

	log_age	Sex	Job	Housing	Saving accounts	Checking account	log_credit_amount	Duration	P
0	2.271006	0.670280	0.146949	-0.133710	0.955847	-1.344000	-0.933901	-1.236478	1.
1	-1.446152	-1.491914	0.146949	-0.133710	-0.706496	-0.265348	1.163046	2.248194	1.
2	1.226696	0.670280	-1.383771	-0.133710	-0.706496	0.813303	-0.181559	-0.738668	0.
3	0.942455	0.670280	0.146949	-2.016956	-0.706496	-1.344000	1.525148	1.750384	0.
4	1.488620	0.670280	0.146949	-2.016956	-0.706496	-1.344000	0.904743	0.256953	-0.

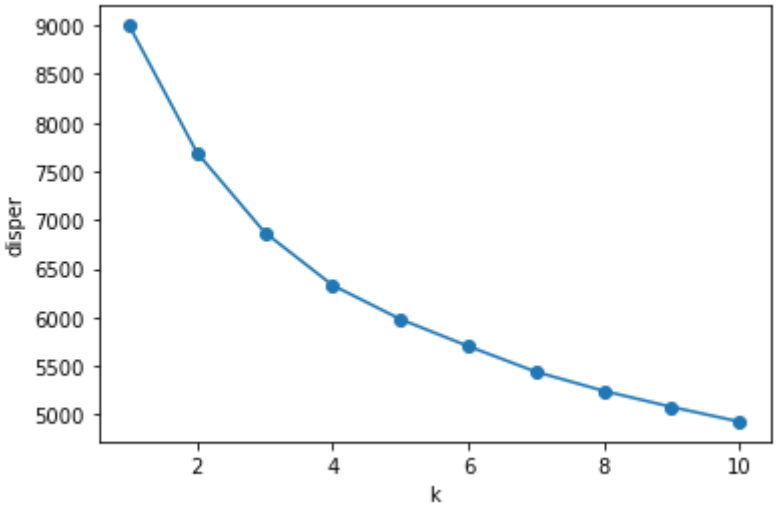
In [28]:

```
#Построим дендрограмму:
from scipy.cluster.hierarchy import dendrogram, linkage,fcluster
link = linkage(X,'ward','euclidean') #ward - метод вычисления растояния ммежду класс
                                     # euclidean - метод вычисления растояния между
dn = dendrogram(link, orientation = 'right')
```



По данной дендрограмме видим 3 кластера разных цветов, но есть предположение, что "красный класстер" можно разбить на 2.

In [29]:
```
# нахождение оптимального кол-ва кластеров с помощью графика "каменистая осыпь"
from sklearn.cluster import KMeans
K = range(1,11)
models = [KMeans(n_clusters=k, random_state=42).fit(X) for k in K]
dist = [model.inertia_ for model in models]
#график
plt.plot(K,dist,marker = 'o')
plt.xlabel('k')
plt.ylabel('disper')
plt.show()
```



- **Получение кластеров и их анализ.**

In [30]:
```
model = KMeans(n_clusters = 4,random_state = 42)
model.fit(X)
df_class['cluster'] = model.labels_
df_class.groupby('cluster').mean()
```

Out[30]:

cluster	log_age	Sex	Job	Housing	Saving accounts	Checking account	log_credit_amount	Duration	I
0	3.637166	0.811518	1.910995	1.026178	2.727749	1.748691	7.561564	17.120419	3

cluster	log_age	Sex	Job	Housing	Saving accounts	Checking account	log_credit_amount	Duration	I
1	3.523066	1.000000	1.721212	1.106061	0.172727	1.130303	7.478015	15.251515	3
2	3.578475	0.847107	2.297521	0.797521	0.599174	1.086777	8.684383	35.714876	2
3	3.379765	0.000000	1.751055	1.337553	0.535865	1.164557	7.489734	16.696203	3

In [31]:
```
# Кол-во попаданий в каждый кластер.
df_class['cluster'].value_counts()
```

Out[31]:
```
1    330
2    242
3    237
0    191
Name: cluster, dtype: int64
```

In [32]:
```
df['cluster'] = df_class['cluster']
```

In [33]:
```
# каждого клиенту присвоили определенный класс.
df.head()
```

Out[33]:

	Age	Sex	Job	Housing	Saving accounts	Checking account	Credit amount	Duration	Purpo
0	67	male	2	own	no_info_saving	little	1169	6	radio/T
1	22	female	2	own	little	moderate	5951	48	radio/T
2	49	male	1	own	little	no_info_checking	2096	12	educatic
3	45	male	2	free	little	little	7882	42	furniture/equipme
4	53	male	2	free	little	little	4870	24	c

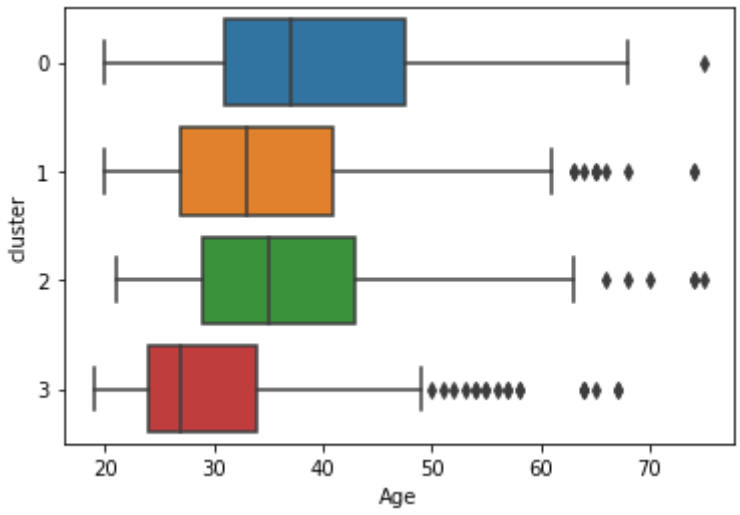
Пройдемся по каждому кластеру .

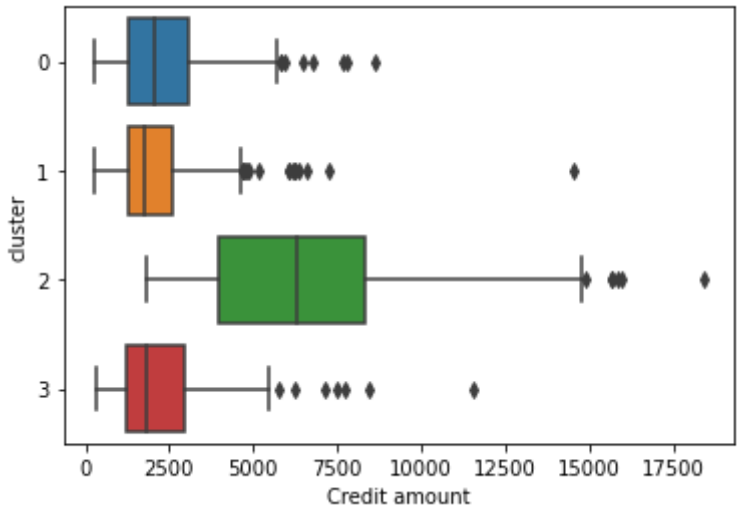
In [34]:
```
# Т.к по графикам boxplot можно заметить присутствие выбросов, для описания кластеро
df_compere = df.groupby('cluster').median()[['Age']]
df_compere['Credit amount'] = df.groupby('cluster').median()[['Credit amount']]
df_compere['Duration'] = df.groupby('cluster').median()[['Duration']]

df_compere.head()
```

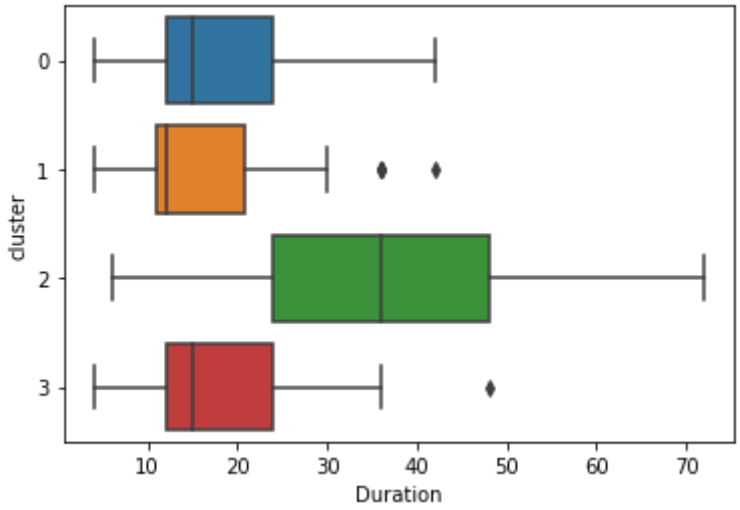
Out[34]:

cluster	Age	Credit amount	Duration
0	37.0	2028.0	15.0
1	33.0	1767.0	12.0
2	35.0	6308.5	36.0
3	27.0	1808.0	15.0

In [35]:
```
# сначала по количественным переменным
num = ['Age','Credit amount','Duration']
for col in df[num].columns:
    sns.boxplot(data=df, x=col, y='cluster', orient='h')
    plt.show();
```







Теперь по номинативным.

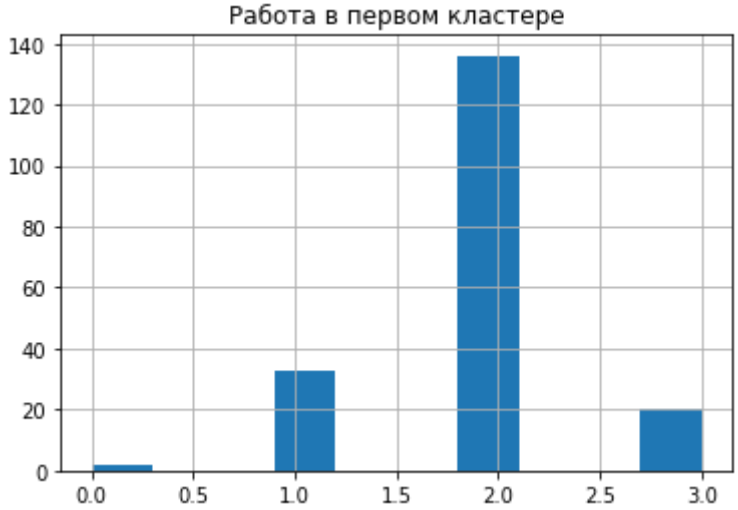
Cluster 0

In [36]:
```
df[df['cluster']==0]['Sex'].hist()
plt.title('Пол в первом кластере')
```

Out[36]: Text(0.5, 1.0, 'Пол в первом кластере')

In [37]:
```
df[df['cluster']==0]['Job'].hist()
plt.title('Работа в первом кластере')
```

Out[37]: Text(0.5, 1.0, 'Работа в первом кластере')

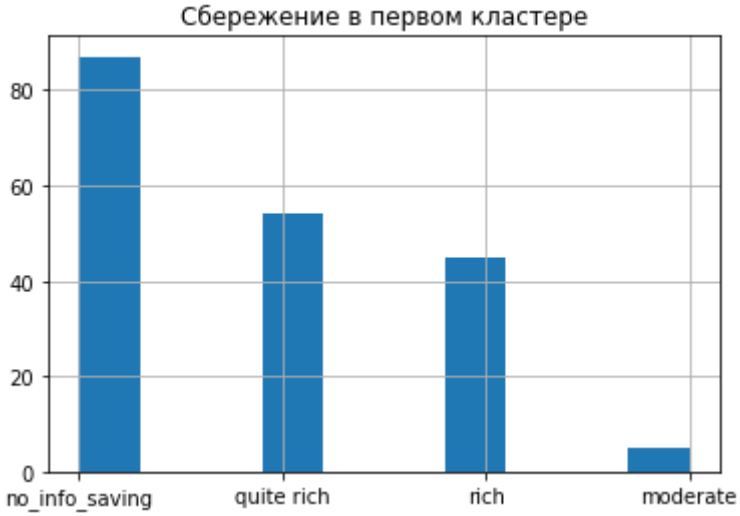


In [38]:
```
df[df['cluster']==0]['Housing'].hist()
plt.title('Недвижимость в первом кластере')
```

Out[38]: Text(0.5, 1.0, 'Недвижимость в первом кластере')

In [39]:
```
df[df['cluster']==0]['Saving accounts'].hist()
plt.title('Сбережение в первом кластере')
```

Out[39]:
```
Text(0.5, 1.0, 'Сбережение в первом кластере')
```

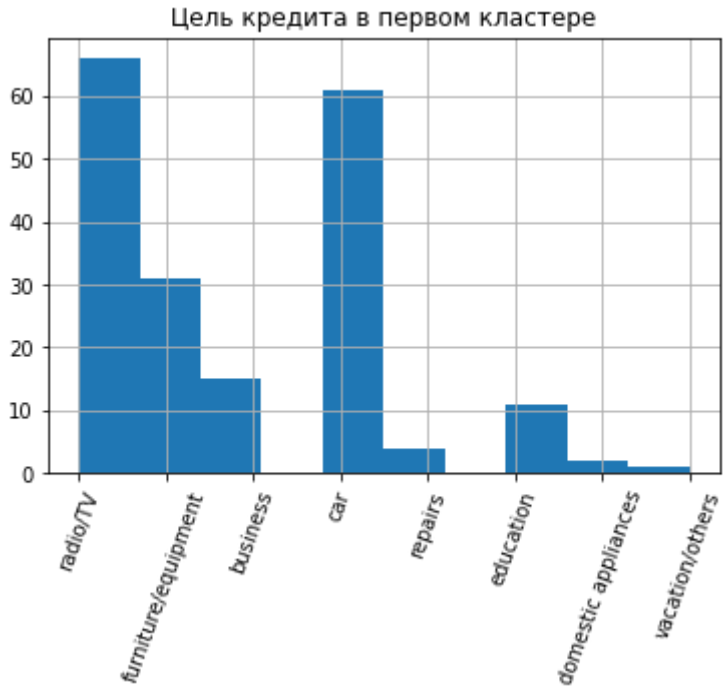


In [40]:
```
df[df['cluster']==0]['Checking account'].hist()
plt.title('Расчетный счет  в первом кластере')
```

Out[40]:
```
Text(0.5, 1.0, 'Расчетный счет  в первом кластере')
```

In [41]:
```
df[df['cluster']==0]['Purpose'].hist()
plt.xticks(rotation=70)
plt.title('Цель кредита в первом кластере')
```

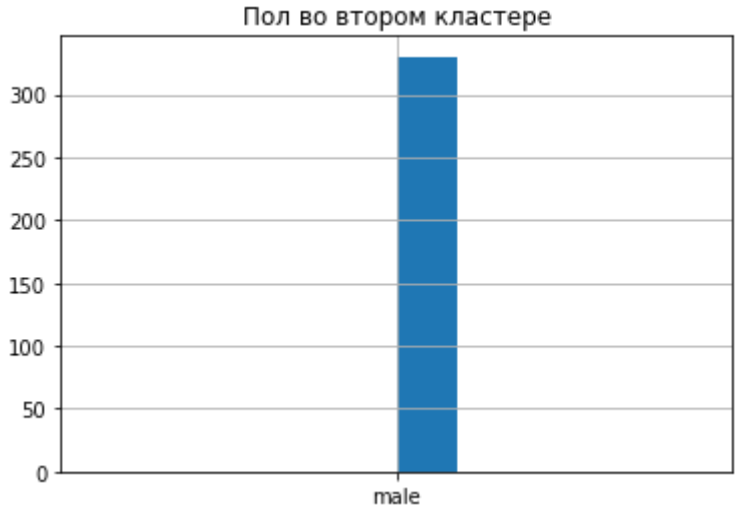
Out[41]:
```
Text(0.5, 1.0, 'Цель кредита в первом кластере')
```



Cluster 1

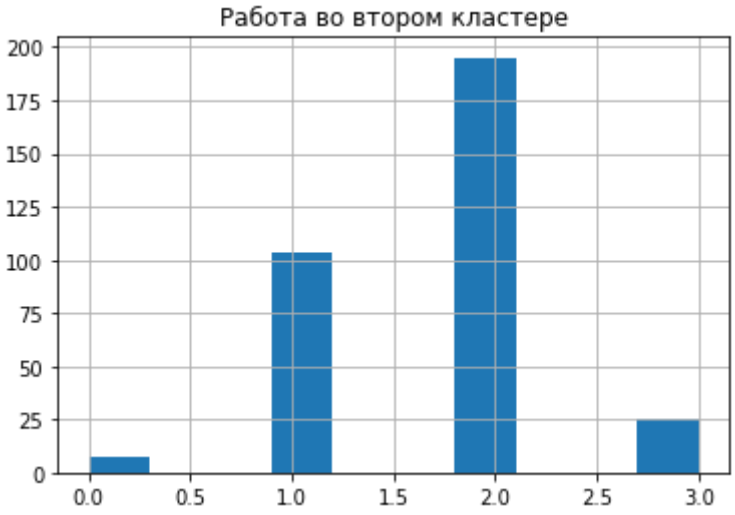
In [42]:
```
df[df['cluster']==1]['Sex'].hist()
plt.title('Пол во втором кластере')
```

Out[42]:
```
Text(0.5, 1.0, 'Пол во втором кластере')
```



In [43]:
```
df[df['cluster']==1]['Job'].hist()
plt.title('Работа во втором кластере')
```

Out[43]:
```
Text(0.5, 1.0, 'Работа во втором кластере')
```

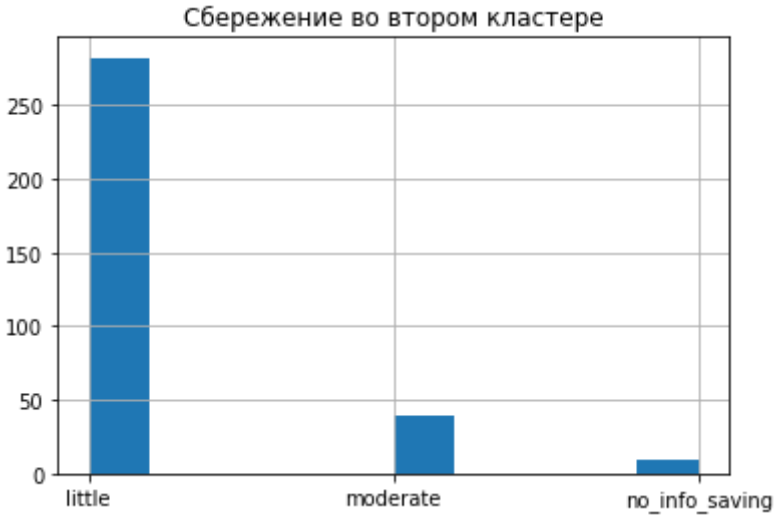


In [44]:
```
df[df['cluster']==1]['Housing'].hist()
plt.title('Недвижимость во втором кластере')
```

Out[44]: Text(0.5, 1.0, 'Недвижимость во втором кластере')

In [45]:
```
df[df['cluster']==1]['Saving accounts'].hist()
plt.title('Сбережение во втором кластере')
```

Out[45]: Text(0.5, 1.0, 'Сбережение во втором кластере')



In [46]:

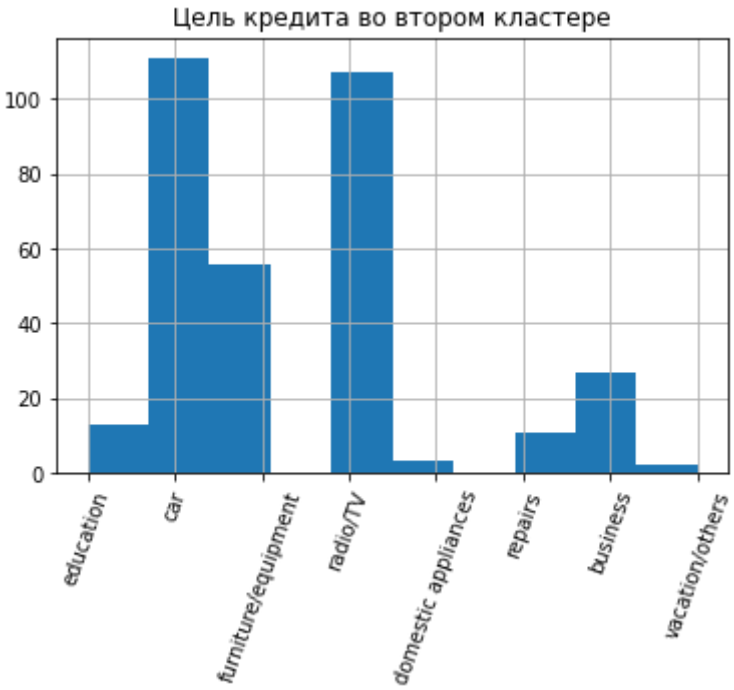
```
df[df['cluster']==1]['Checking account'].hist()
plt.title('Расчетный счет  во втором кластере')
```

Out[46]: Text(0.5, 1.0, 'Расчетный счет во втором кластере')

In [47]:

```
df[df['cluster']==1]['Purpose'].hist()
plt.xticks(rotation=70)
plt.title('Цель кредита во втором кластере')
```

Out[47]: Text(0.5, 1.0, 'Цель кредита во втором кластере')

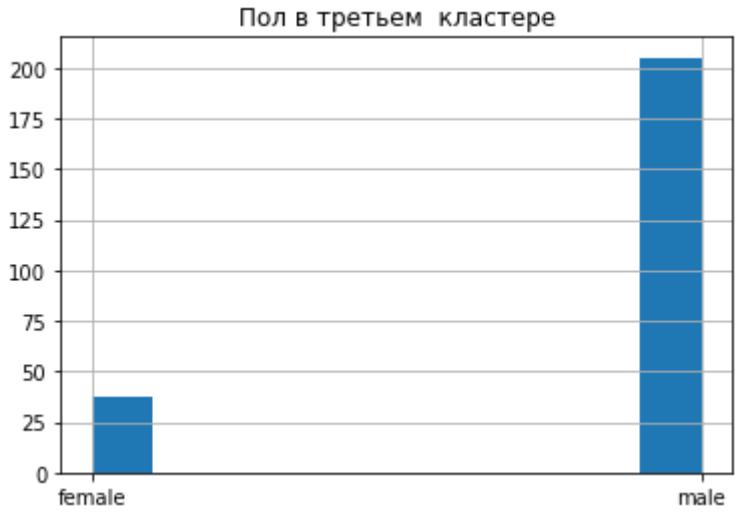


Cluster 2

In [48]:

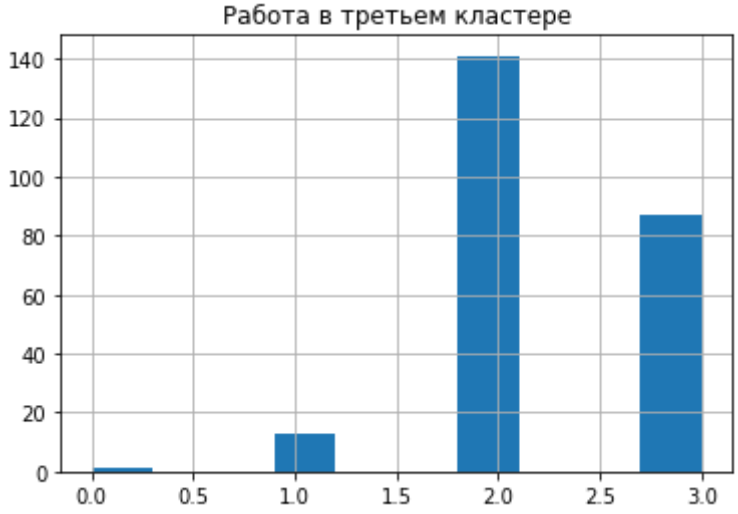
```
df[df['cluster']==2]['Sex'].hist()
plt.title('Пол в третьем  кластере')
```

Out[48]: Text(0.5, 1.0, 'Пол в третьем кластере')



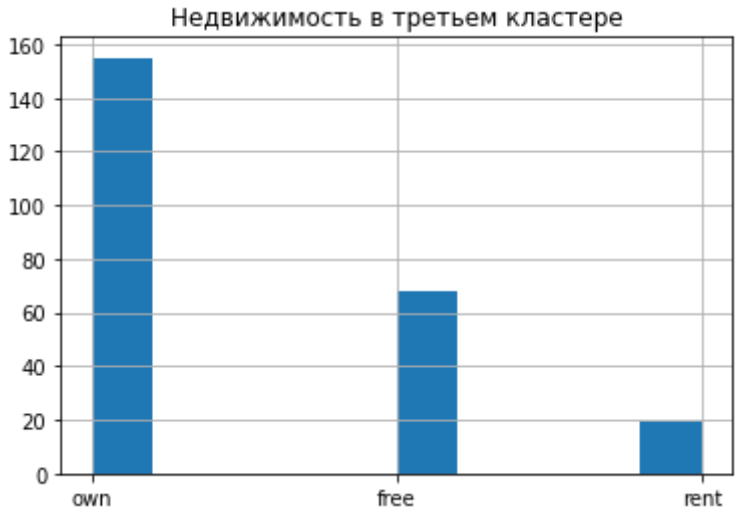
In [49]:
```
df[df['cluster']==2]['Job'].hist()
plt.title('Работа в третьем кластере')
```

Out[49]: Text(0.5, 1.0, 'Работа в третьем кластере')



In [50]:
```
df[df['cluster']==2]['Housing'].hist()
plt.title('Недвижимость в третьем кластере')
```

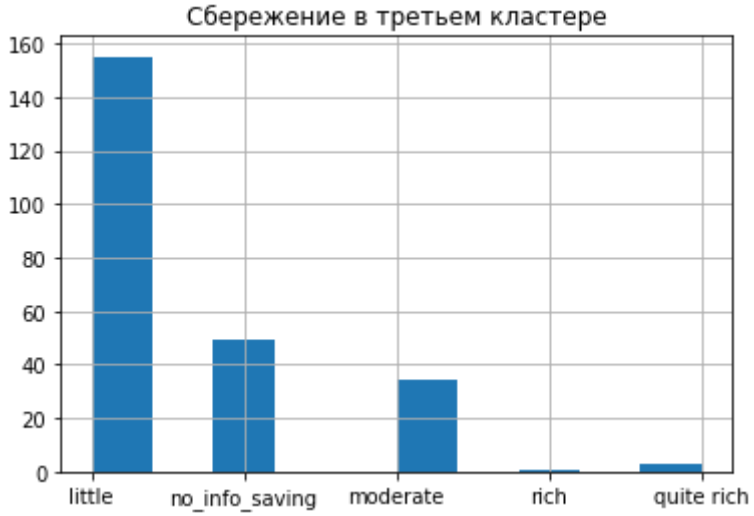
Out[50]: Text(0.5, 1.0, 'Недвижимость в третьем кластере')



In [51]:

```
df[df['cluster']==2]['Saving accounts'].hist()
plt.title('Сбережение в третьем кластере')
```

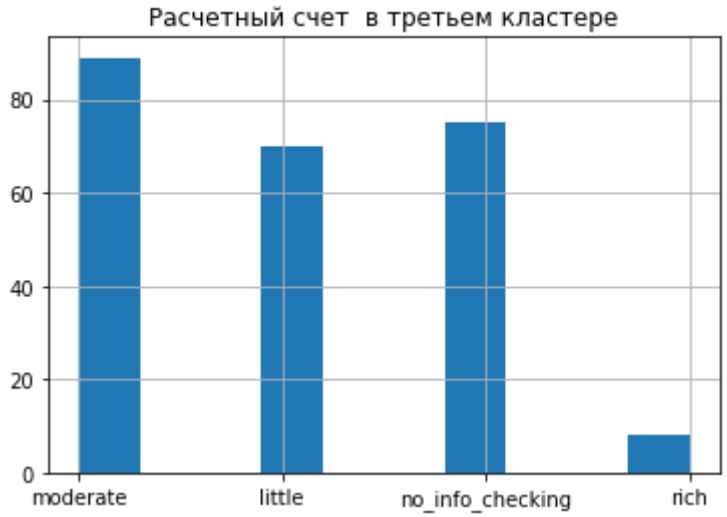
Out[51]: Text(0.5, 1.0, 'Сбережение в третьем кластере')



In [52]:

```
df[df['cluster']==2]['Checking account'].hist()
plt.title('Расчетный счет  в третьем кластере')
```

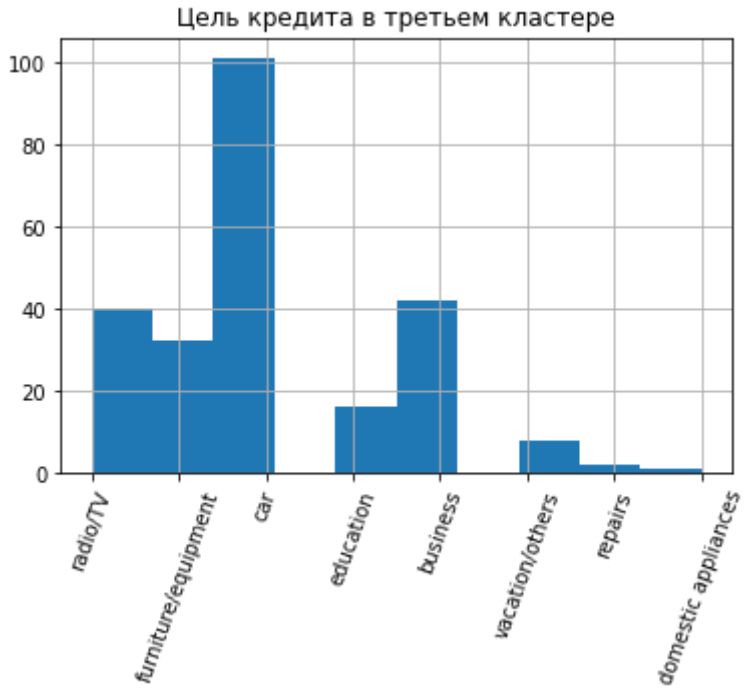
Out[52]: Text(0.5, 1.0, 'Расчетный счет в третьем кластере')



In [53]:

```
df[df['cluster']==2]['Purpose'].hist()
plt.xticks(rotation=70)
plt.title('Цель кредита в третьем кластере')
```

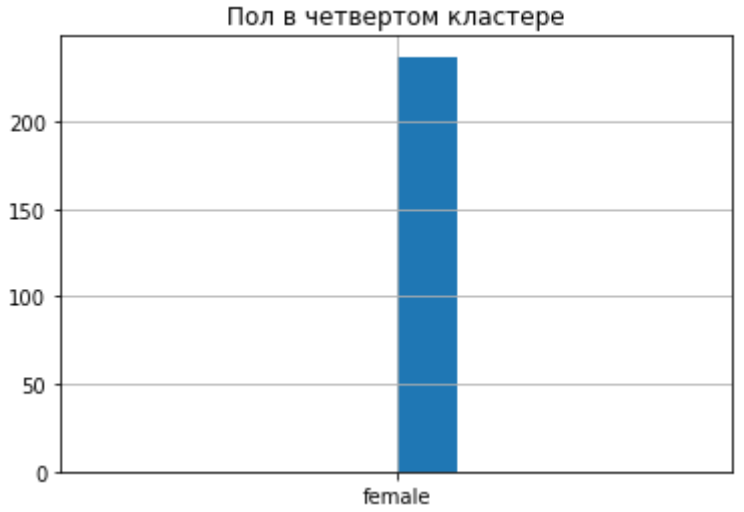
Out[53]: Text(0.5, 1.0, 'Цель кредита в третьем кластере')



Cluster 3

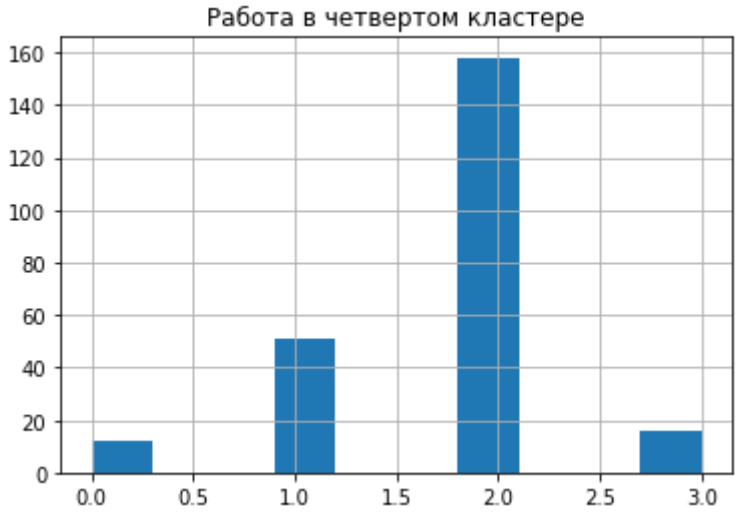
In [54]:
```
df[df['cluster']==3]['Sex'].hist()
plt.title('Пол в четвертом кластере')
```

Out[54]: Text(0.5, 1.0, 'Пол в четвертом кластере')



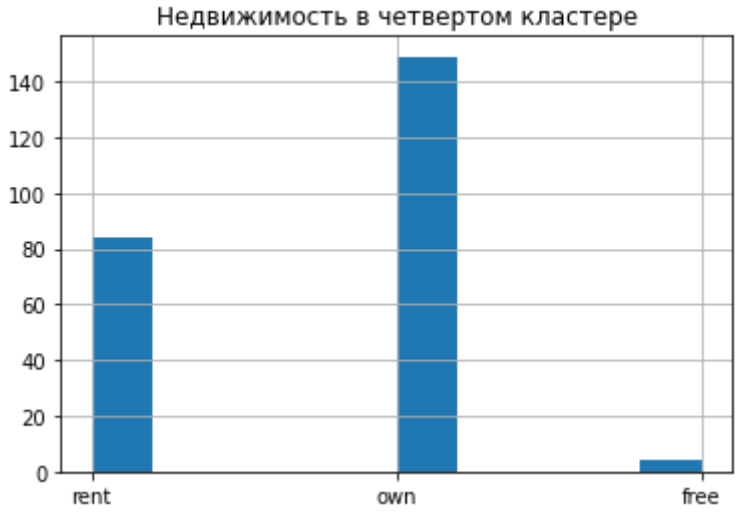
In [55]:
```
df[df['cluster']==3]['Job'].hist()
plt.title('Работа в четвертом кластере')
```

Out[55]: Text(0.5, 1.0, 'Работа в четвертом кластере')



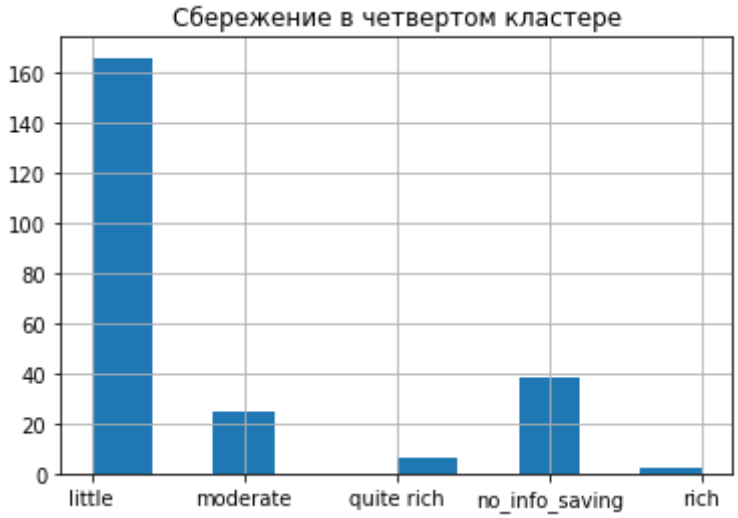
In [56]:
```
df[df['cluster']==3]['Housing'].hist()
plt.title('Недвижимость в четвертом кластере')
```

Out[56]: Text(0.5, 1.0, 'Недвижимость в четвертом кластере')



In [57]:
```
df[df['cluster']==3]['Saving accounts'].hist()
plt.title('Сбережение в четвертом кластере')
```

Out[57]: Text(0.5, 1.0, 'Сбережение в четвертом кластере')



In [58]:

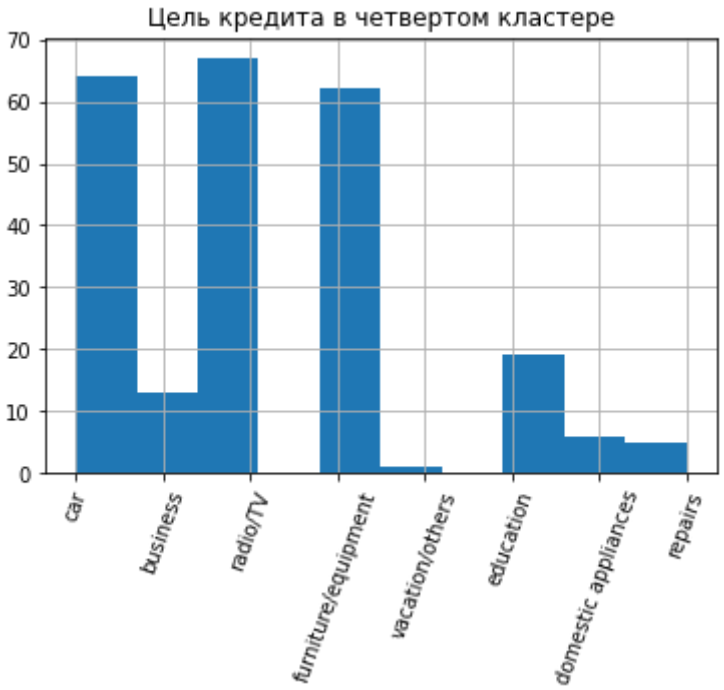
```
df[df['cluster']==3]['Checking account'].hist()
plt.title('Расчетный счет  в четвертом кластере')
```

Out[58]: Text(0.5, 1.0, 'Расчетный счет в четвертом кластере')

In [59]:

```
df[df['cluster']==3]['Purpose'].hist()
plt.xticks(rotation=70)
plt.title('Цель кредита в четвертом кластере')
```

Out[59]: Text(0.5, 1.0, 'Цель кредита в четвертом кластере')



- **Выводы :**

Анализирую результаты, можно составить картину "типичного представителя каждого кластера"

Cluster 0: Клиент - женщина (класс работы:1,2), средний возраст которой примерно 27 лет. Средняя сумма кредита составляет 1808 DM 15 мес.

Cluster 1: Клиент - мужчина (класс работы:1,2), средний возраст которого примерно 33 года. Средняя сумма кредита составляет 1766 DM на 12 мес.

Cluster 2: Клиент - в большей чати мужчина (класс работы:1,2), средний возраст которого(ой) примерно 35 лет. Средняя сумма кредита составляет 6304 DM на 36 мес.Также изменяется цель: бизнес.

Cluster 3: Клиент - в большей чати женщина (класс работы:2), средний возраст которой(ого) примерно 37 лет. Средняя сумма кредита составляет 2028 DM на 15 мес.

Что касается остальных параметров, то они схожи.

Отметим, что алгоритм сошелся довольно неплохо, хотя не всегда в данных можно проследить структуру, и соответственно разделить на кластеры у нас бы не получилось.

4. Вторая подзадача (Кредитный скоринг)

- **Описание метода решения:**

  Для решения задачи оценки кредитного риска воспользуемся алгоритмами классификации (с учителем). Выберем несколько моделей (Logistic regression, RandomForest, XGboost) и на основе их качества остановимся на одной. Можно было сразу остановиться на фаворите, но для определенных классификаторов есть виды задач, где они себя хорошо показывают.

  Источники по каждому классификатору:

  Logistic regression

  RandomForest

  XGboost

  Для начала преобразуем данные (произведем кодировку). Выделим целевую переменную и предикторы, также необходимо разбить выборку на обучающую и валидационную. Следующим шагом построим по ним модели. Посмотрим на качество каждой модели и влияние предикторов. Выдвинем определенные гипотезы: например, преобразуем переменные или вовсе удалим некоторые. Также сделаем для самой качественной модели валидационный подбор параметров с целью избежать переобучение.

- **Импорт библиотек.**

In [60]:
```
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score,classification_report
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import  GridSearchCV
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import classification_report
from sklearn.model_selection import  GridSearchCV
from xgboost import XGBClassifier
```

- **Преобразование данных.**

In [61]:
```

```

```
df_model = df.copy()
df_model.drop(columns='cluster',inplace=True)
df_model.head()
```

Out[61]:

	Age	Sex	Job	Housing	Saving accounts	Checking account	Credit amount	Duration	Purpo
0	67	male	2	own	no_info_saving	little	1169	6	radio/T
1	22	female	2	own	little	moderate	5951	48	radio/T
2	49	male	1	own	little	no_info_checking	2096	12	educatic
3	45	male	2	free	little	little	7882	42	furniture/equipme
4	53	male	2	free	little	little	4870	24	c

In [62]:

```
df_model = pd.get_dummies(data=df_model, columns=['Sex','Housing' , 'Saving accounts
df_model.head()
```

Out[62]:

	Age	Job	Credit amount	Duration	default	Sex_female	Sex_male	Housing_free	Housing_own	Hous
0	67	2	1169	6	0	0	1	0	1	
1	22	2	5951	48	1	1	0	0	1	
2	49	1	2096	12	0	0	1	0	1	
3	45	2	7882	42	0	0	1	1	0	
4	53	2	4870	24	1	0	1	1	0	

5 rows × 27 columns

- **Выделение целевой переменной и предикторов, разбиение на train и test.**

In [63]:

```
# выбросим колонку-отклик и соберем все данные в матрицу X
y = df_model['default']
X = df_model.drop(['default'], axis = 1)
```

In [64]:

```
# расщепляем на test и train
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split(X, y, test_size = 0.10, random_s
```

- **Построение классификаторов.**

  1) Logistic regression

In [65]:

```
model_1 = LogisticRegression(
    # метод для поиска решения. Для небольших датасетов лучше подходит liblinear, sa
    # Варианты: newton-cg, lbfgs, liblinear, sag, saga
    solver='liblinear',
    # норма для регуляризации
    penalty='l2',
    # параметр регуляризации. Чем меньше, тем сильнее регуляризация. Положительный.
```

```
    C=1,
    # параметр для остановки поиска решения.
    tol=1e-4,
    # Так как распознается 2 класса.
    multi_class='ovr'
    )
```

In [66]:
```
model_1.fit(x_train,y_train)
print(model_1,'Accuracy = ',accuracy_score(y_test, model_1.predict(x_test)))
print(classification_report(y_test, model_1.predict(x_test)))
```

```
LogisticRegression(C=1, multi_class='ovr', solver='liblinear') Accuracy =  0.79
              precision    recall  f1-score   support

           0       0.81      0.92      0.86        71
           1       0.70      0.48      0.57        29

    accuracy                           0.79       100
   macro avg       0.76      0.70      0.72       100
weighted avg       0.78      0.79      0.78       100
```

2) RandomForest

In [67]:
```
clf_rf = RandomForestClassifier(random_state=42,
                                n_estimators=90,
                                min_samples_leaf = 1,
                                min_samples_split = 5,
                                criterion='gini',
                                max_depth=7,
                                warm_start=False,
                                class_weight=None
                               )
```

In [68]:
```
# валидационный подбор параметров.
'''
parametrs = {
    "n_estimators": [80,90],
    'min_samples_leaf':[1,2,3],
    "min_samples_split": [4,5,3],
    'criterion':['entropy','gini'],
    'max_depth':[6,7,8]
}
#grid_search_clf = GridSearchCV(clf_rf,parametrs,cv=5)
#grid_search_clf.fit(x_train,y_train)
'''
```

Out[68]:
```
'\nparametrs = {\n    "n_estimators": [80,90],\n    \'min_samples_leaf\':[1,2,3],\n
"min_samples_split": [4,5,3],\n    \'criterion\':[\'entropy\',\'gini\'],\n    \'max_
depth\':[6,7,8]\n}\n#grid_search_clf = GridSearchCV(clf_rf,parametrs,cv=5)\n#grid_se
arch_clf.fit(x_train,y_train)\n'
```

In [69]:
```
model_2 = RandomForestClassifier(random_state=42,
                                n_estimators=90,
                                min_samples_leaf = 1,
                                min_samples_split = 5,
                                criterion='entropy',
                                max_depth=8,
                                warm_start=False,
                                class_weight=None
                               )
```

In [70]:
```
model_2.fit(x_train,y_train)
print(model_2,'Accuracy = ',accuracy_score(y_test, model_2.predict(x_test)))
print(classification_report(y_test, model_2.predict(x_test)))
```

```
RandomForestClassifier(criterion='entropy', max_depth=8, min_samples_split=5,
                       n_estimators=90, random_state=42) Accuracy =  0.81
              precision    recall  f1-score   support

           0       0.82      0.93      0.87        71
           1       0.75      0.52      0.61        29

    accuracy                           0.81       100
   macro avg       0.79      0.72      0.74       100
weighted avg       0.80      0.81      0.80       100
```

3) XGboost

In [71]:
```
clf_xg = XGBClassifier(seed=42,
                       n_estimators=100,
                       max_depth=6,
                       learning_rate=0.3)
```

In [72]:
```
# валидационный подбор параметров.
'''
params = {
    "learning_rate"    :[0.05,0.15,0.25,0.35],
    "max_depth"        :[ 3,6,12],
    "min_child_weight" :[ 1,3,5],
    "gamma"            :[ 0.0,0.1,0.2],
    "colsample_bytree" :[ 0.3, 0.4, 0.5, 0.7 ]
}
grid_search_clf = GridSearchCV(model,params,cv=5)
grid_search_clf.fit(x_train,y_train)

grid_search_clf.best_params_
'''
```

Out[72]:
```
'\nparams = {\n    "learning_rate"    :[0.05,0.15,0.25,0.35],\n    "max_depth"
:[ 3,6,12],\n    "min_child_weight" :[ 1,3,5],\n    "gamma"            :[ 0.0,0.1,0.
2],\n    "colsample_bytree" :[ 0.3, 0.4, 0.5, 0.7 ]\n}\ngrid_search_clf = GridSearch
CV(model,params,cv=5)\ngrid_search_clf.fit(x_train,y_train)\n\ngrid_search_clf.best_
params_\n'
```

In [73]:
```
model_3 = XGBClassifier(base_score=0.5, booster='gbtree', colsample_bylevel=1,
              colsample_bynode=1, colsample_bytree=1, gamma=0,
              learning_rate=0.1, max_delta_step=0, max_depth=4,
              min_child_weight=1, missing=1, n_estimators=90, n_jobs=1
                        , objective='binary:logistic', random_state=0,
              reg_alpha=0, reg_lambda=1, scale_pos_weight=1, seed=42,
               subsample=1, verbosity=1)
```

In [74]:
```
model_3.fit(x_train,y_train)
```

```
[14:02:06] WARNING: ../src/learner.cc:1095: Starting in XGBoost 1.3.0, the default e
valuation metric used with the objective 'binary:logistic' was changed from 'error'
to 'logloss'. Explicitly set eval_metric if you'd like to restore the old behavior.

/srv/conda/envs/notebook/lib/python3.6/site-packages/xgboost/sklearn.py:1146: UserWa
rning: The use of label encoder in XGBClassifier is deprecated and will be removed i
n a future release. To remove this warning, do the following: 1) Pass option use_lab
```

```
el_encoder=False when constructing XGBClassifier object; and 2) Encode your labels
(y) as integers starting with 0, i.e. 0, 1, 2, ..., [num_class - 1].
  warnings.warn(label_encoder_deprecation_msg, UserWarning)
```

Out[74]:
```
XGBClassifier(base_score=0.5, booster='gbtree', colsample_bylevel=1,
              colsample_bynode=1, colsample_bytree=1, gamma=0, gpu_id=-1,
              importance_type='gain', interaction_constraints='',
              learning_rate=0.1, max_delta_step=0, max_depth=4,
              min_child_weight=1, missing=1, monotone_constraints='()',
              n_estimators=90, n_jobs=1, num_parallel_tree=1, random_state=0,
              reg_alpha=0, reg_lambda=1, scale_pos_weight=1, seed=42,
              subsample=1, tree_method='exact', validate_parameters=1,
              verbosity=1)
```

In [75]:
```
y_pred_train = model_3.predict(x_train)
print (classification_report(y_train, y_pred_train))

y_pred = model_3.predict(x_test)
print (classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.88      0.97      0.93       629
           1       0.92      0.70      0.80       271

    accuracy                           0.89       900
   macro avg       0.90      0.84      0.86       900
weighted avg       0.89      0.89      0.89       900

              precision    recall  f1-score   support

           0       0.83      0.92      0.87        71
           1       0.73      0.55      0.63        29

    accuracy                           0.81       100
   macro avg       0.78      0.73      0.75       100
weighted avg       0.80      0.81      0.80       100
```

- **Выдвижение гипотез.**

Посмотрим на влияния каждой переменной в классификаторе xgboost.

In [76]:
```
pd.DataFrame({'feature': X.columns,
              'importance': model_3.feature_importances_}).sort_values('importance',
```

Out[76]:

	feature	importance
16	Checking account_no_info_checking	0.281039
10	Saving accounts_moderate	0.047679
8	Housing_rent	0.044910
11	Saving accounts_no_info_saving	0.043946
3	Duration	0.043269
9	Saving accounts_little	0.041785
14	Checking account_little	0.038620
19	Purpose_car	0.035656
2	Credit amount	0.035274
13	Saving accounts_rich	0.033196

	feature	importance
22	Purpose_furniture/equipment	0.033184
21	Purpose_education	0.033159
17	Checking account_rich	0.032813
18	Purpose_business	0.031609
1	Job	0.031597
6	Housing_free	0.029733
23	Purpose_radio/TV	0.029533
0	Age	0.028525
7	Housing_own	0.028283
4	Sex_female	0.028123
24	Purpose_repairs	0.022427
15	Checking account_moderate	0.016559
25	Purpose_vacation/others	0.009080
12	Saving accounts_quite rich	0.000000
20	Purpose_domestic appliances	0.000000
5	Sex_male	0.000000

В модели xgboost видим, что большой вклад вносит колонка Checking account со значением 'no_info', также вносит вклад колонка Saving accounts со значением 'no_info'. Из этого можно сделать вывод, что данная модель не является качественной, т.к в ≈32% принятие решение основывается на отсутствующей информации. Для решения этой проблемы попробуем удалить все пропуски. Также удалим дублирующую колонку Sex_male. Purpose_vacation/others, Purpose_domestic appliances, Saving accounts_quite rich тоже подлежат удаления, т.к не вносят вклад.

Посмотрим на влияния каждой переменной в классификаторе RandomForest.

In [77]:
```
pd.DataFrame({'feature': X.columns,
              'importance': model_2.feature_importances_}).sort_values('importance',
```

Out[77]:

	feature	importance
2	Credit amount	0.173105
0	Age	0.140625
3	Duration	0.125116
16	Checking account_no_info_checking	0.114905
14	Checking account_little	0.064729
1	Job	0.044898
11	Saving accounts_no_info_saving	0.028766
15	Checking account_moderate	0.027403
9	Saving accounts_little	0.025868

	feature	importance
23	Purpose_radio/TV	0.024044
18	Purpose_business	0.023836
7	Housing_own	0.021937
5	Sex_male	0.020626
19	Purpose_car	0.018613
4	Sex_female	0.017774
21	Purpose_education	0.017099
6	Housing_free	0.016201
8	Housing_rent	0.016177
22	Purpose_furniture/equipment	0.016072
10	Saving accounts_moderate	0.014560
13	Saving accounts_rich	0.012673
17	Checking account_rich	0.010372
12	Saving accounts_quite rich	0.009520
24	Purpose_repairs	0.006633
20	Purpose_domestic appliances	0.004543
25	Purpose_vacation/others	0.003907

Данная модель по логическим соображениям более качественна (радуют три первые переменные). Но все равно видим, что присутствует влияние предикторов со значанием 'no_info'. Удалим пропуски. Также дублирующую колонку Sex_male.

- **Проверка гипотез:**

  Проверка гипотез прошла неудачно, т.к удалив пропущенные строки мы получили 522 строки, тем самым потеряли половину данных, классификаторы соответственно потеряли в качестве почти на 20 процентов.

In [78]:
```
df = pd.read_csv('credit.csv',sep=',')
df.drop('Unnamed: 0',axis=1,inplace=True)
df.drop('Checking account',axis=1,inplace=True)
```

In [79]:
```
df.head()
```

Out[79]:

	Age	Sex	Job	Housing	Saving accounts	Credit amount	Duration	Purpose	default
0	67	male	2	own	NaN	1169	6	radio/TV	0
1	22	female	2	own	little	5951	48	radio/TV	1
2	49	male	1	own	little	2096	12	education	0
3	45	male	2	free	little	7882	42	furniture/equipment	0

	Age	Sex	Job	Housing	Saving accounts	Credit amount	Duration	Purpose	default
4	53	male	2	free	little	4870	24	car	1

In [80]:
```
df['Saving accounts'] = df['Saving accounts'].fillna('no_info_saving')
```

In [81]:
```
df_new = pd.get_dummies(data=df, columns=['Sex','Housing' , 'Saving accounts','Purpo
df_model.head()
```

Out[81]:

	Age	Job	Credit amount	Duration	default	Sex_female	Sex_male	Housing_free	Housing_own	Hous
0	67	2	1169	6	0	0	1	0	1	
1	22	2	5951	48	1	1	0	0	1	
2	49	1	2096	12	0	0	1	0	1	
3	45	2	7882	42	0	0	1	1	0	
4	53	2	4870	24	1	0	1	1	0	

5 rows × 27 columns

In [82]:
```
# выбросим колонку-отклик и соберем все данные в матрицу X
y =df_new['default']
X =df_new.drop(['default'], axis = 1)
```

In [83]:
```
# расщепляем на test и train
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split(X, y, test_size = 0.10, random_s
```

In [84]:
```
model = XGBClassifier(seed=42,
                      n_estimators=100,
                      max_depth=6,
                      learning_rate=0.3)
```

In [85]:
```
params = {
    "learning_rate"    :[0.05,0.20,0.3],
    "max_depth"        :[ 3,6,9],
    "min_child_weight" :[ 1,3,5],
    "gamma"            :[ 0.0,0.2],
    "colsample_bytree" :[ 0.3, 0.5]
}
```

In [91]:
```
'''
grid_search_clf = GridSearchCV(model,params,cv=5)
grid_search_clf.fit(x_train,y_train)
grid_search_clf.best_params_
'''
```

Out[91]:
```
'\ngrid_search_clf = GridSearchCV(model,params,cv=5)\ngrid_search_clf.fit(x_train,y_
train)\ngrid_search_clf.best_params_\n'
```

In [87]:
```
model_3 = XGBClassifier(base_score=0.5, booster='gbtree', colsample_bylevel=1,
              colsample_bynode=1, colsample_bytree=0.3,
              learning_rate=0.03, max_delta_step=0, max_depth=7,
              min_child_weight=5, missing=1, n_estimators=75
                        , objective='binary:logistic', random_state=42,
              seed = 42)
```

In [88]:
```
model_3.fit(x_train,y_train)
y_pred_train = model_3.predict(x_train)
print (classification_report(y_train, y_pred_train))

y_pred = model_3.predict(x_test)
print (classification_report(y_test, y_pred))
```

```
[14:02:53] WARNING: ../src/learner.cc:1095: Starting in XGBoost 1.3.0, the default e
valuation metric used with the objective 'binary:logistic' was changed from 'error'
to 'logloss'. Explicitly set eval_metric if you'd like to restore the old behavior.
              precision    recall  f1-score   support

           0       0.74      0.99      0.84       629
           1       0.89      0.17      0.29       271

    accuracy                           0.74       900
   macro avg       0.81      0.58      0.57       900
weighted avg       0.78      0.74      0.68       900

              precision    recall  f1-score   support

           0       0.78      0.99      0.87        71
           1       0.90      0.31      0.46        29

    accuracy                           0.79       100
   macro avg       0.84      0.65      0.67       100
weighted avg       0.81      0.79      0.75       100
```

In [92]:
```
pd.DataFrame({'feature': X.columns,
              'importance': model_3.feature_importances_}).sort_values('importance',
```

Out[92]:

	feature	importance
9	Saving accounts_little	0.133653
11	Saving accounts_no_info_saving	0.121843
7	Housing_own	0.090768
2	Credit amount	0.073974
3	Duration	0.068251
6	Housing_free	0.054020
13	Saving accounts_rich	0.049668
19	Purpose_radio/TV	0.048668
8	Housing_rent	0.046233
12	Saving accounts_quite rich	0.039268
0	Age	0.037151
17	Purpose_education	0.037150

	feature	importance
10	Saving accounts_moderate	0.032281
15	Purpose_car	0.031298
14	Purpose_business	0.029599
1	Job	0.027683
5	Sex_male	0.027139
4	Sex_female	0.026476
18	Purpose_furniture/equipment	0.024877
16	Purpose_domestic appliances	0.000000
20	Purpose_repairs	0.000000
21	Purpose_vacation/others	0.000000

Вывод:

- **Общее заключение:**

По предоставленным данным хорошо можно проводить кластерный анализ, алгоритм k-means сходиться качественно. Иначе дело обстоит с классификацией: во-первых, по атрибутам сберегательного и расчетного счета много пропусков, и проблема в том, что они вносят весомый вклад (особенно в модель xgboost), при наличии датасета с большим кол-вом строк эта проблема была бы легко разрешима, но в наших данных 1000 строк, и при удалении пропусков остается 50% от данных. Во-вторых, исходя из логических соображений и учебных материалов, колонка job должна вносить хороший весовой вклад, но в каждой из нашей модели ее вес очень низкий, т.к большинство клиентов попадают во второй класс, поэтому можно пересмотреть ранжирование класса работы, или сделать другой предиктор - заработная плата. Таким образом для улучшения качетсва модели нужно наращивать датасет, и, возможно, поискать другие предикторы, для более качественной характеристики клиента.

In [ ]: