Московский авиационный институт

(Национальный исследовательский университет)

Лабораторная работа № 1

Тема: Данные и проблемы в них

По курсу: Машинное обучение

Студент: Бабичева А. Д.

Группа: М80-304Б-17

Дата:

Оценка:

Москва, 2020

Задание: необходимо сформировать два набора данных для приложений машинного обучения. Первый датасет должен представлять из себя табличный набор данных для задачи классификации. Второй датасет должен быть отличен от первого, и может представлять из себя набор изображений, корпус документов, другой табличный датасет или датасет из соревнования Kaggle, предназначенный для решения интересующей вас задачи машинного обучения. Необходимо провести анализ обоих наборов данных, поставить решаемую вами задачу, определить признаки необходимые для решения задачи, в случае необходимости заняться генерацией новых признаков, устранением проблем в данных, визуализировать распределение и зависимость целевого признака от выбранных признаков.

Наборы данных:

1. Adult Income – набор данных для задачи классификации. Включает в себя 2 выборки: тренировочную (train.csv) и тестовую (test.csv).

   Колонки:

   - sex (object) – пол
   - income (object) – заработок
   - race (object) – раса
   - relationship (object) – отношения
   - marital-status (object) – семейное положение
   - workclass (object) – рабочий класс
   - occupation (object) – сфера работы
   - education (object) – образование
   - education-num (int) – образовательный номер
   - native-country (object) – родная страна
   - age (int) – возраст
   - capital-loss (int) – потери капитала
   - hours-per-week (int) – кол-во рабочих часов в неделю
   - capital-gain (int) – прибыль
   - fnlwgt (int) – некоторый числовой параметр

Решаемая задача. Решается задача классификации: необходимо определить, какие люди имеют заработок больше 50 тысяч долларов в год, а какие меньше. Для тестовой выборки необходимо предсказать значения колонки income по заданным.

1. Ames Iowa Housing Data — набор данных для задачи регрессии. Включает в себя 2 выборки: тренировочную (train.csv) и тестовую (test.csv).

   Колонки:

   - Order - Observation number.
   - PID - Parcel identification number - can be used with city web site for parcel review.
   - MS SubClass - Identifies the type of dwelling involved in the sale.
   - MS Zoning - Identifies the general zoning classification of the sale.
   - Lot Frontage - Linear feet of street connected to property.
   - Lot Area - Lot size in square feet.
   - Street - Type of road access to property.
   - Alley - Type of alley access to property.
   - Lot Shape - General shape of property.
   - Land Contour - Flatness of the property.
   - Utilities - Type of utilities available.
   - Lot Config - Lot configuration.
   - Land Slope - Slope of property.
   - Neighborhood - Physical locations within Ames city limits (map available).
   - Condition 1 - Proximity to various conditions.
   - Condition 2 - Proximity to various conditions (if more than one is present).
   - Bldg Type - Type of dwelling.
   - House Style - Style of dwelling.
   - Overall Qual - Rates the overall material and finish of the house.
   - Overall Cond - Rates the overall condition of the house.
   - Year Built - Original construction date.

- Year Remod/Add - Remodel date (same as construction date if no remodeling or additions).
- Roof Style - Type of roof.
- Roof Matl - Roof material.
- Exterior 1st - Exterior covering on house.
- Exterior 2nd - Exterior covering on house (if more than one material).
- Mas Vnr Type - Masonry veneer type.
- Mas Vnr Area - Masonry veneer area in square feet.
- Exter Qual - Evaluates the quality of the material on the exterior.
- Exter Cond - Evaluates the present condition of the material on the exterior.
- Foundation - Type of foundation.
- Bsmt Qual - Evaluates the height of the basement.
- Bsmt Cond - Evaluates the general condition of the basement.
- Bsmt Exposure - Refers to walkout or garden level walls.
- BsmtFin Type 1 - Rating of basement finished area.
- BsmtFin SF 1 - Type 1 finished square feet.
- BsmtFin Type 2 - Rating of basement finished area (if multiple types).
- BsmtFin SF 2 - Type 2 finished square feet.
- Bsmt Unf SF - Unfinished square feet of basement area.
- Total Bsmt SF - Total square feet of basement area.
- Heating - Type of heating.
- Heating QC - Heating quality and condition.
- Central Air - Central air conditioning.
- Electrical - Electrical system.
- 1st Flr SF - First Floor square feet.
- 2nd Flr SF - Second floor square feet.
- Low Qual Fin SF - Low quality finished square feet (all floors).
- Gr Liv Area - Above grade (ground) living area square feet.
- Bsmt Full Bath - Basement full bathrooms.

- Bsmt Half Bath - Basement half bathrooms.
- Full Bath - Full bathrooms above grade.
- Half Bath - Half baths above grade.
- Bedroom AbvGr - Bedrooms above grade (does NOT include basement bedrooms).
- Kitchen AbvGr - Kitchens above grade.
- Kitchen Qual - Kitchen quality.
- TotRms AbvGrd - Total rooms above grade (does not include bathrooms).
- Functional - Home functionality (Assume typical unless deductions are warranted).
- Fireplaces - Number of fireplaces.
- Fireplace Qu - Fireplace quality.
- Garage Type - Garage location.
- Garage Yr Blt - Year garage was built.
- Garage Finish - Interior finish of the garage.
- Garage Cars - Size of garage in car capacity.
- Garage Area - Size of garage in square feet.
- Garage Qual - Garage quality.
- Garage Cond - Garage condition.
- Paved Drive - Paved driveway.
- Wood Deck SF - Wood deck area in square feet.
- Open Porch SF - Open porch area in square feet.
- Enclosed Porch - Enclosed porch area in square feet.
- 3Ssn Porch - Three season porch area in square feet.
- Screen Porch - Screen porch area in square feet.
- Pool Area - Pool area in square feet.
- Pool QC - Pool quality.
- Fence - Fence quality.
- Misc Feature - Miscellaneous feature not covered in other categories.

- Misc Val - $Value of miscellaneous feature.
- Mo Sold - Month Sold.
- Yr Sold - Year Sold.
- Sale Type - Type of sale.
- Sale Condition - Condition of sale.
- SalePrice - Sale Price (target variable for the Regression).

Решаемая задача. Решается задача регрессии: для тестовой выборки необходимо по заданным параметрам предсказать стоимость дома (колонка SalePrice).

Анализ и подготовка данных Adult Income

Проблемы данных:

1. Данные типа object
2. Колонка, значения которой необходимо предсказать, имеет тип данных object

```
train.dtypes
```

```
age                int64
workclass         object
fnlwgt             int64
education         object
education-num      int64
marital-status    object
occupation        object
relationship      object
race              object
sex               object
capital-gain       int64
capital-loss       int64
hours-per-week     int64
native-country    object
income            object
dtype: object
```

3. Затраты по памяти

```
train.info()
```

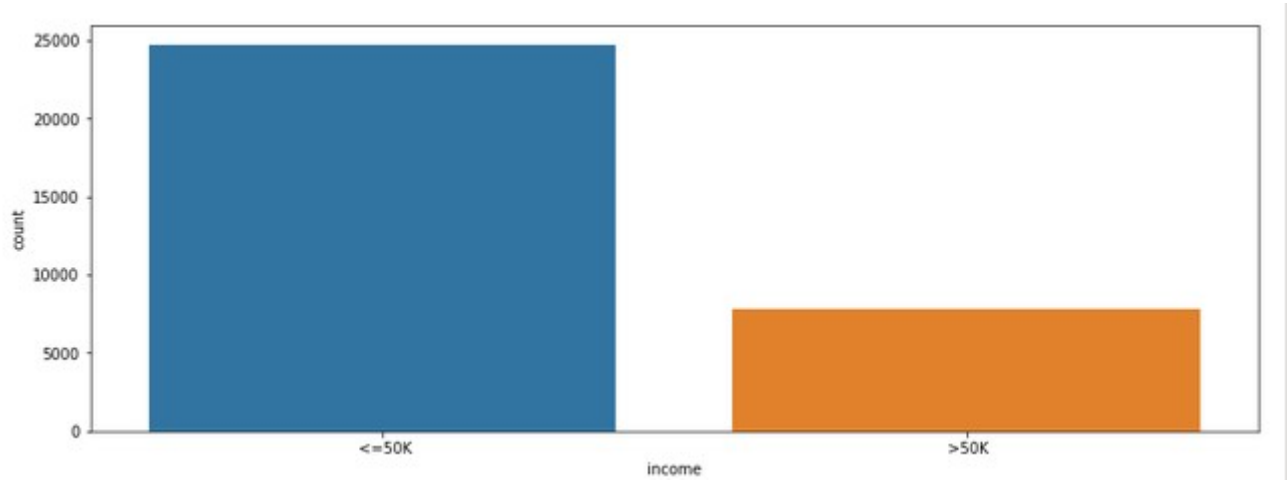
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 32561 entries, 0 to 32560
Data columns (total 15 columns):
age               32561 non-null int64
workclass         32561 non-null object
fnlwgt            32561 non-null int64
education         32561 non-null object
education-num     32561 non-null int64
marital-status    32561 non-null object
occupation        32561 non-null object
relationship      32561 non-null object
race              32561 non-null object
sex               32561 non-null object
capital-gain      32561 non-null int64
capital-loss      32561 non-null int64
hours-per-week    32561 non-null int64
native-country    32561 non-null object
income            32561 non-null object
dtypes: int64(6), object(9)
memory usage: 3.7+ MB
```

```
memory_analyse(train)
```

```
Average memory usage for float columns: 0.00 MB
Average memory usage for int columns: 0.21 MB
Average memory usage for object columns: 1.87 MB
```

4. Пропущенные данные, которые не являются типом None (в датасете представлены значением „ ?“

5. Несбалансированность данных



Анализ и подготовка

1. Затраты по памяти можно минимизировать путем приведения данных типа int64 к типу in32.

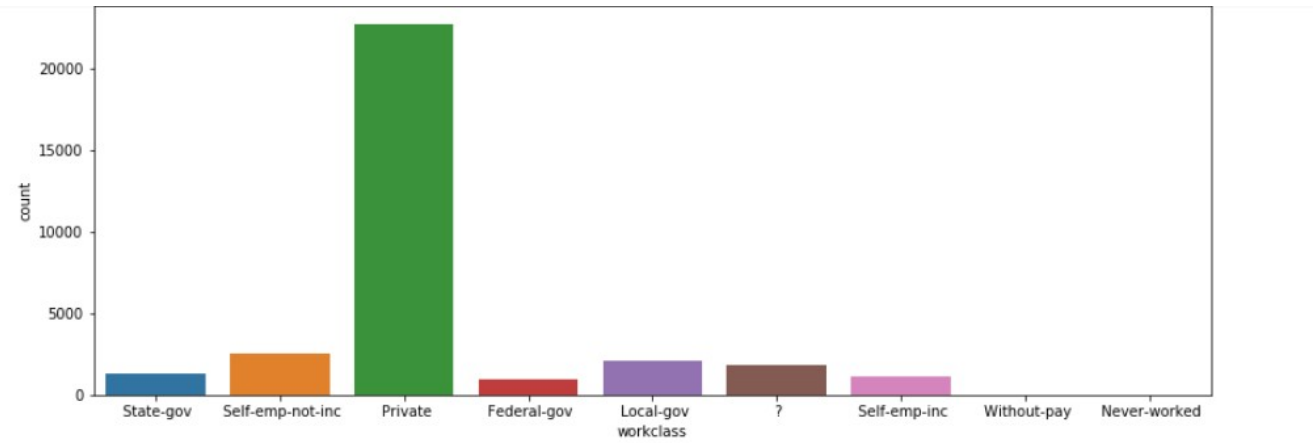
```
#минимизация затрат по памяти
for col in train.columns:
    if (train[col].dtype == int):
        test[col] = test[col].astype('int32')
        train[col] = train[col].astype('int32')
train.info()
```

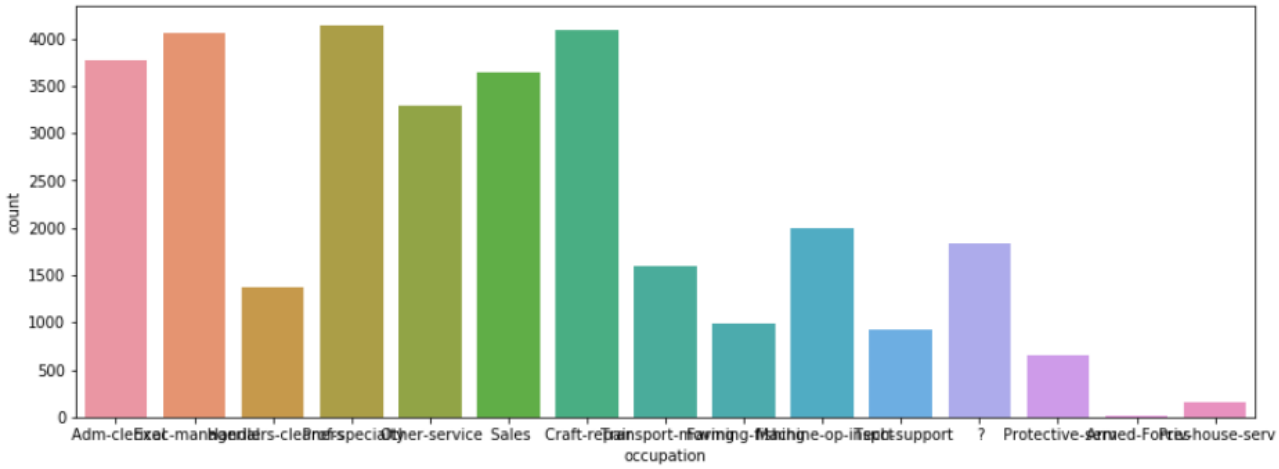
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 32561 entries, 0 to 32560
Data columns (total 15 columns):
age               32561 non-null int32
workclass         32561 non-null object
fnlwgt            32561 non-null int32
education         32561 non-null object
education-num     32561 non-null int32
marital-status    32561 non-null object
occupation        32561 non-null object
relationship      32561 non-null object
race              32561 non-null object
sex               32561 non-null object
capital-gain      32561 non-null int32
capital-loss      32561 non-null int32
hours-per-week    32561 non-null int32
native-country    32561 non-null object
income            32561 non-null object
dtypes: int32(6), object(9)
memory usage: 3.0+ MB
```

```
memory_analyse(train)
```

```
Average memory usage for float columns: 0.00 MB
Average memory usage for int columns: 0.00 MB
Average memory usage for object columns: 1.87 MB
```

2. Представлю графики распределения значений данных, имеющих пропущенные значения





Как видно, в первом и в третьем случае целесообразно будет осуществлять замену пропущенных данных самым часто встречающимся значением.

3. Для определения стратегии работы с данными типа object посмотрю, сколько уникальных значений в каждой колонке датасета.

	Column_Name	Type	Num_Unique
9	sex	object	2
14	income	object	2
8	race	object	5
7	relationship	object	6
5	marital-status	object	7
1	workclass	object	9
6	occupation	object	15
3	education	object	16
4	education-num	int32	16
13	native-country	object	42
0	age	int32	73
11	capital-loss	int32	92
12	hours-per-week	int32	94
10	capital-gain	int32	119
2	fnlwgt	int32	21648

По полученным результатам выбираю следующую стратегию: колонки с двумя уникальными значениями представить в виде бинарных признаков, колонки с количеством уникальных элементов до десяти включительно заменю эквивалентными колонками с бинарными признаками с помощью one-hot encoding, все остальные колонки типа object представлю типом category. Для колонок capital-loss и capital-gain проведу нормализацию.

Для удобства дальнейшего анализа сразу проведу преобразование над колонкой income.

```
train.income.replace([' >50K', ' <=50K'], [1, 0], inplace=True)
test.income.replace([' >50K.', ' <=50K.'], [1, 0], inplace=True)

train['income'] = train['income'].astype('int32')
test['income'] = test['income'].astype('int32')
```

4. Для корректной работы с пропущенными признаками приведу такие данные к типу None с помощью написанной функции.

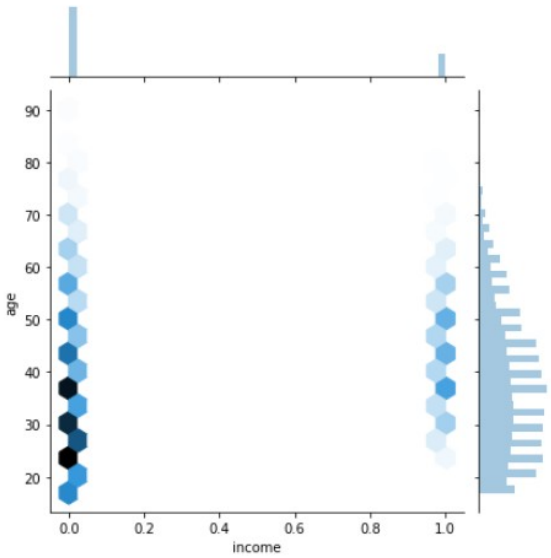
```
def make_NaN(df):
    for col in df.columns:
        df[col][df[col] == ' ?'] = None
```

```
make_NaN(train)
make_NaN(test)
```

5. Далее буду графически стараться выявлять зависимость между некоторыми признаками и income.

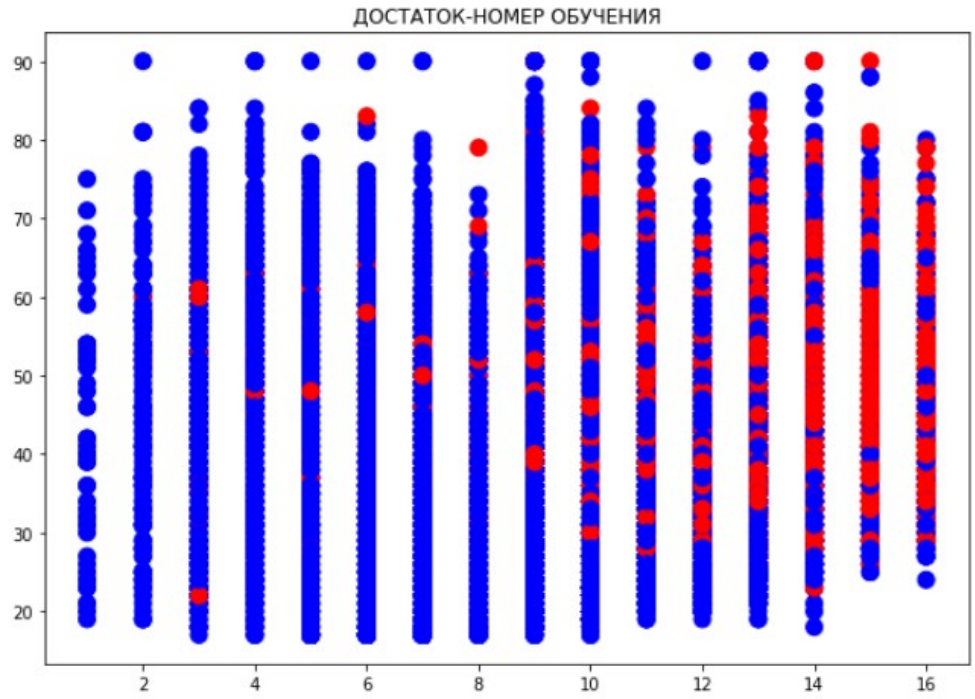
```
# ДОСТАТОК-ВОЗРАСТ
fig = plt.figure(figsize=(40, 40))
sns.jointplot(x='income', y='age', data=train, kind ='hex',gridsize=20)
```

```
<seaborn.axisgrid.JointGrid at 0x7f8dd4ae9438>

<Figure size 2880x2880 with 0 Axes>
```



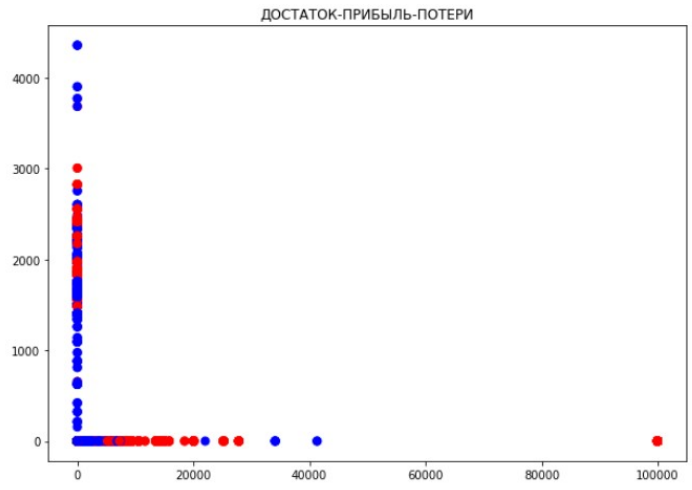
Как видно, большинство людей до 40 лет имеют заработок менее 50 тысяч долларов в год, поэтому возраст можно отнести к значимым признакам.

```
# ДОСТАТОК-НОМЕР ОБУЧЕНИЯ-ВОЗРАСТ
colors = ListedColormap(["blue", "red"])
# 0 - blue
# 1 - red
plt.figure(figsize=(10,7))
plt.title("ДОСТАТОК-НОМЕР ОБУЧЕНИЯ")
plt.scatter(train['education-num'], train['age'], c=train['income'], cmap=colors, s=100)
plt.show()
```



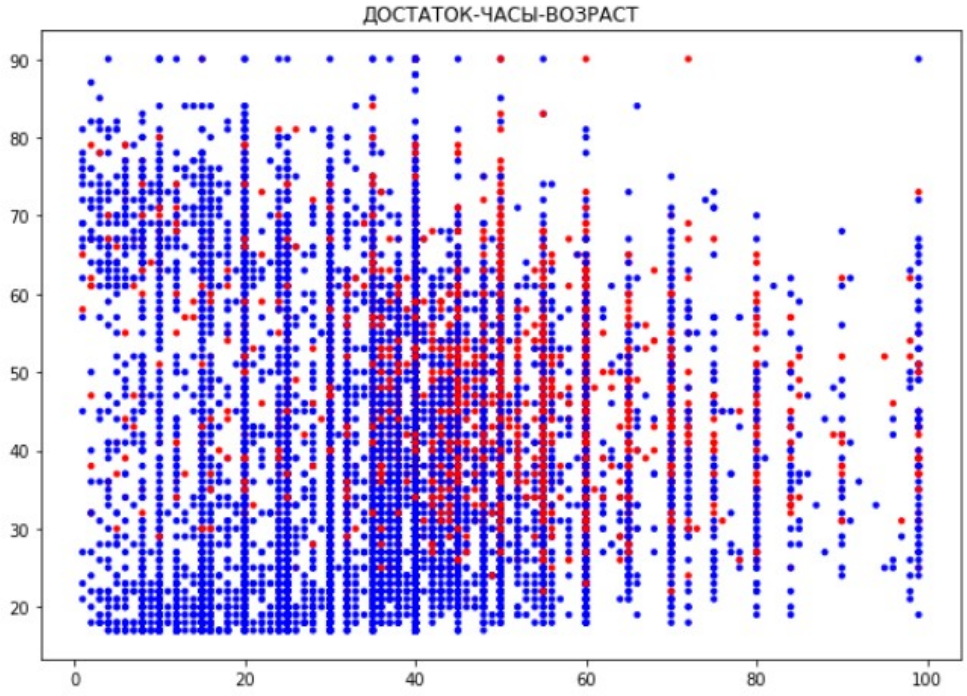
Большинство людей, имеющих большой заработок, относятся к 12 — 16 номеру обучения, поэтому данный признак тоже отношу к значимым.

```
# ДОСТАТОК-ПРИБЫЛЬ-ПОТЕРИ
colors = ListedColormap(["blue", "red"])
# 0 - blue
# 1 - red
plt.figure(figsize=(10,7))
plt.title("ДОСТАТОК-ПРИБЫЛЬ-ПОТЕРИ")
plt.scatter(train['capital-gain'], train['capital-loss'], c=train['income'], cmap=colors, s=50)
plt.show()
```



Как прибыль, так и денежные потери людей явно находятся в зависимости с заработком.

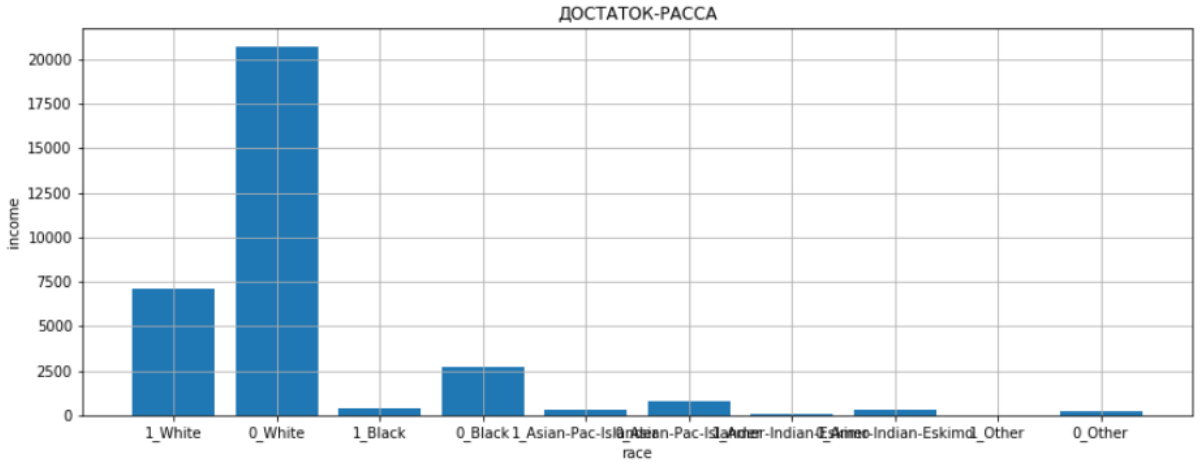
```
# ДОСТАТОК-ЧАСЫ-ВОЗРАСТ
colors = ListedColormap(["blue", "red"])
# 0 - blue
# 1 - red
plt.figure(figsize=(10,7))
plt.title("ДОСТАТОК-ЧАСЫ-ВОЗРАСТ")
plt.scatter(train['hours-per-week'], train['age'], c=train['income'], cmap=colors, s=10)
plt.show()
#те кто мало или много работают, скорее всего, мало не получают
```



Данный график показывает, что люди, имеющие большой заработок, преимущественно работают от 40 до 60 часов в неделю, поэтому этот признак тоже считаю значимым.

```
# ДОСТАТОК-РАССА

cols = [' White', ' Black', ' Asian-Pac-Islander', ' Amer-Indian-Eskimo', ' Other']
cls = ['1_White', '0_White', '1_Black', '0_Black', '1_Asian-Pac-Islander', '0_Asian-Pac-Islander', '1_Am

values = []

for c in cols:
    values.append(train[train.race == c][['income']].income.sum())
    values.append(train[(train.race == c) & (train.income == 0)][['income']].income.count())

fig = plt.figure(figsize=(14,5))
ax1 = fig.add_subplot()
ax1.set_xlabel('race')
ax1.set_ylabel('income')
ax1.set_title('ДОСТАТОК-РАССА')
ax1.grid()
ax1.bar(cls, values)
```

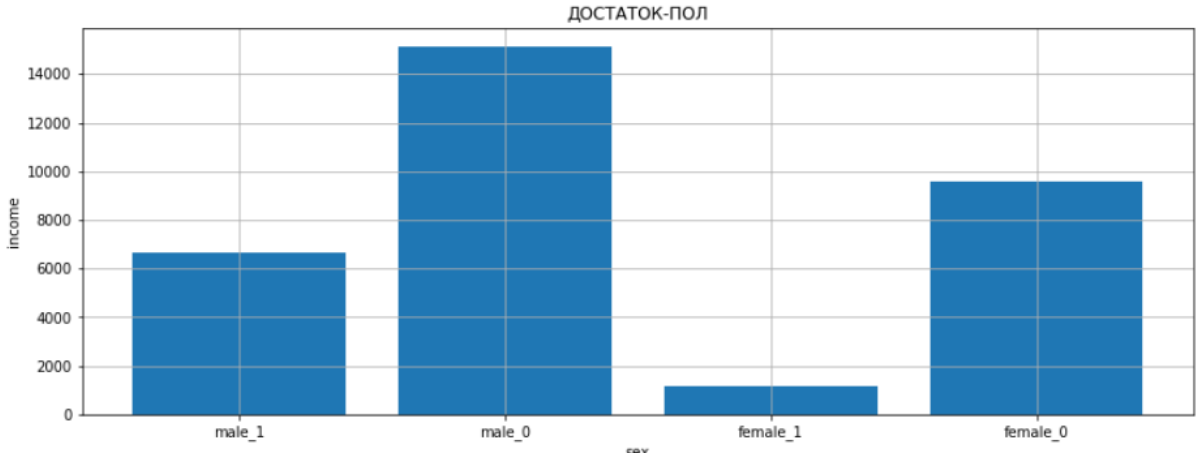
```
<BarContainer object of 10 artists>
```



Видно, что большиство людей с хорошим заработком относятся к белой расе, однако это может быть связано с несбалансированностью данных, но признак все равно буду считать значимым.

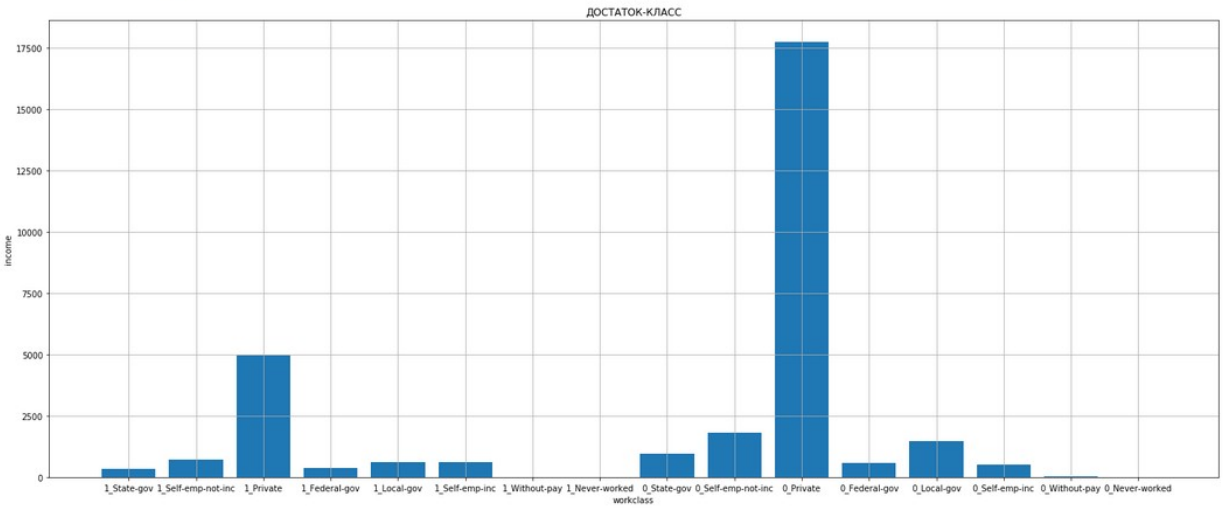
```
# ДОСТАТОК-ПОЛ

sex = ['male_1','male_0', 'female_1', 'female_0']

male_1 = train[train.sex == ' Male'][['income']].income.sum()
male_0 = train[(train.sex == ' Male') & (train.income == 0)][['sex']].count()
female_1 = train[train.sex == ' Female'][['income']].income.sum()
female_0 = train[(train.sex == ' Female') & (train.income == 0)][['sex']].count()

cls = [male_1, male_0, female_1, female_0]

fig = plt.figure(figsize=(14,5))
ax1 = fig.add_subplot()
ax1.set_xlabel('sex')
ax1.set_ylabel('income')
ax1.set_title('ДОСТАТОК-ПОЛ')
ax1.grid()
ax1.bar(sex, cls)
```

```
<BarContainer object of 4 artists>
```



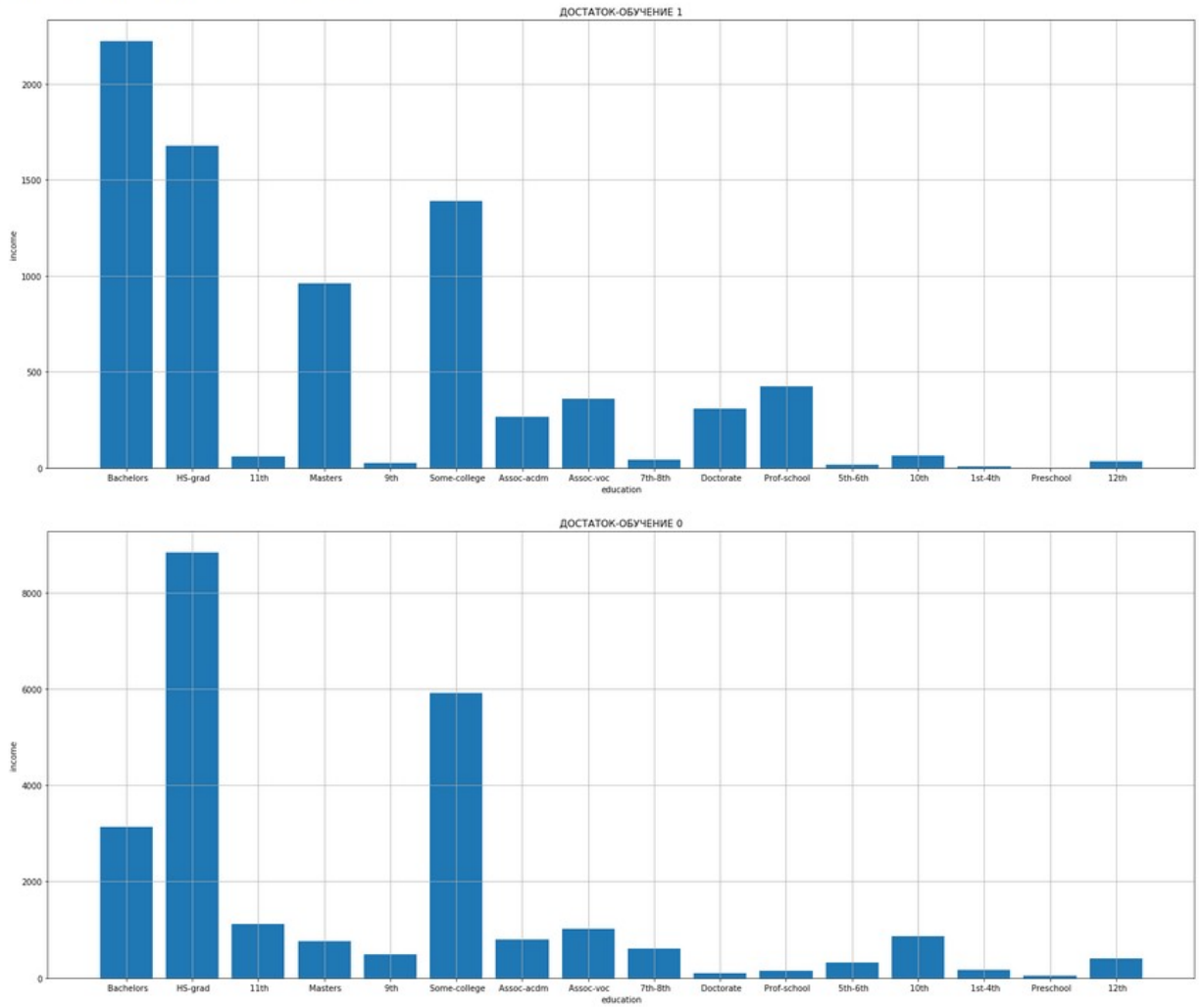
Здесь явно прослеживается, что треть мужчин имею большой заработок в то время, как лишь одиннадцатая часть женщин имеют такой же заработок, поэтому признак считаю значимым.

```
<BarContainer object of 16 artists>
```



По данному графику так же прослеживается некая закономерность, поэтому рабочий класс буду считать значимым признаком.

```
<BarContainer object of 16 artists>
```



По данному графику видно, что многие виды образования сильно влияют на заработок человека, поэтому признак значимый.

6. Далее займусь отбором наиболее значимых численных признаков с помощью корреляции.

```
train.corr()
# age, education-num, capital-gain, capital-loss, hours-per-week
```

	age	fnlwgt	education-num	capital-gain	capital-loss	hours-per-week	income
age	1.000000	-0.076646	0.036527	0.077674	0.057775	0.068756	0.234037
fnlwgt	-0.076646	1.000000	-0.043195	0.000432	-0.010252	-0.018768	-0.009463
education-num	0.036527	-0.043195	1.000000	0.122630	0.079923	0.148123	0.335154
capital-gain	0.077674	0.000432	0.122630	1.000000	-0.031615	0.078409	0.223329
capital-loss	0.057775	-0.010252	0.079923	-0.031615	1.000000	0.054256	0.150526
hours-per-week	0.068756	-0.018768	0.148123	0.078409	0.054256	1.000000	0.229689
income	0.234037	-0.009463	0.335154	0.223329	0.150526	0.229689	1.000000

Видно, что меньше всего income коррелирует с fnlwgt, поэтому он не войдет в список значимых признаков.

7. Далее я подумаю, что делать с пропущенными значениями в колонке occupation, так как в ней мы наблюдаем практически равномерное распределение между наиболее часто встречающимися признаками.

   Для начала с помощью написанной функции посмотрю на количество None.

```
def find_NaN(df):
    nans = df.loc[:, df.isnull().any()].copy()
    print(nans.columns)
    nans = nans.isna().sum()
    print(nans)
```

```
find_NaN(train)
```

```
Index(['workclass', 'occupation', 'native-country'], dtype='object')
workclass         1836
occupation        1843
native-country     583
dtype: int64
```

   Далее я пыталась сопоставить данные этой колонки с данными какой-либо другой, чтобы определить, в каких случая я могу заменить пропущенное значение тем или иным значением.

```
# occupation
for col in train.columns:
    if train[col].dtype == object:
        print(pd.crosstab(train[col], train['occupation']))
        print('\n\n\n')
```

   Однако, и тут все признаки имели примерно одинаковое распределение (подробности в ноутбуке), поэтому на свой страх и риск было принято такое решение: так как датасет имеет более 32 тысяч наименований, количество пропущенных элементов относительно невелико, поэтому буду заменять их наиболее часто встречающимися.

8. Подготовка данных сводилась к выделению обучающей выборки, ее таргета и тестовых данных, далее устранению пропущенных данных и преобразованию датасетов по ранее принятой стратегии.

```
def delete_NaN(df):
    #cols = ['workclass', 'occupation', 'native-country']
    df['workclass'][df['workclass'].isnull()] = df.describe(include=['object']).loc['top', 'workclass']
    df['occupation'][df['occupation'].isnull()] = df.describe(include=['object']).loc['top', 'occupation']
    df['native-country'][df['native-country'].isnull()] = df.describe(include=['object']).loc['top', 'native-countr

def one_hot(df, col):
    dfs = pd.get_dummies(df[col], prefix=col)
    df_ = pd.concat([df, dfs], axis=1)
    df_ = df_.drop(col, axis=1)
    return df_

def transform_df(df):
    unique_counts = pd.DataFrame.from_records([(col, df[col].dtype, df[col].nunique()) for col in df.columns],
                          columns=['Column_Name', 'Type', 'Num_Unique']).sort_values(by=['Num_Unique'])
    df_ = df.copy()
    for col in df.columns:
        if (col == 'capital-gain') | (col == 'capital-loss'):
            df_[col] = (df_[col] - df_[col].mean()) / df_[col].std()
        elif (unique_counts[unique_counts['Column_Name'] == col][['Num_Unique']].values[0][0] == 2) & (df_[col].dty
            values = df_[col].unique()
            df_[col].replace(values, [1, 0], inplace=True)
        elif (unique_counts[unique_counts['Column_Name'] == col][['Num_Unique']].values[0][0] < 10) & (df_[col].dty
            df_ = one_hot(df_, col).copy()
        elif  (unique_counts[unique_counts['Column_Name'] == col][['Num_Unique']].values[0][0] >= 10) & (df_[col].d
            df_[col] = df_[col].astype('category')
    return df_
```

```
unnessesary = ['fnlwgt']
```

```
X = train.copy()
delete_NaN(X)
X = train.drop(unnessesary, axis=1)
X = X.drop('income', axis=1)
X = transform_df(X).copy()
X.head()
```

	age	education	education-num	occupation	sex	capital-gain	capital-loss	hours-per-week	native-country	workclass_Federal-gov	workclass_Local-gov	workclass_Never-worked	workclass_Private	workclass_Self-emp-inc	workcl Self- no
0	39	Bachelors	13	Adm-clerical	1	0.148451	-0.216656	40	United-States	0	0	0	0	0	
1	50	Bachelors	13	Exec-managerial	1	-0.145918	-0.216656	13	United-States	0	0	0	0	0	
2	38	HS-grad	9	Handlers-cleaners	1	-0.145918	-0.216656	40	United-States	0	0	0	1	0	
3	53	11th	7	Handlers-cleaners	1	-0.145918	-0.216656	40	United-States	0	0	0	1	0	
4	28	Bachelors	13	Prof-specialty	0	-0.145918	-0.216656	40	Cuba	0	0	0	1	0	

```
y = train['income']
test_ = test.drop('income', axis=1)
delete_NaN(test_)
test_ = train.drop(unnessesary, axis=1)
test_ = transform_df(test_).copy()
target = test['income']
test_.head()
```

	age	education	education-num	occupation	sex	capital-gain	capital-loss	hours-per-week	native-country	income	workclass_Federal-gov	workclass_Local-gov	workclass_Never-worked	workclass_Private	workclass_Self-emp-inc
0	39	Bachelors	13	Adm-clerical	1	0.148451	-0.216656	40	United-States	0	0	0	0	0	0
1	50	Bachelors	13	Exec-managerial	1	-0.145918	-0.216656	13	United-States	0	0	0	0	0	0
2	38	HS-grad	9	Handlers-cleaners	1	-0.145918	-0.216656	40	United-States	0	0	0	0	1	0
3	53	11th	7	Handlers-cleaners	1	-0.145918	-0.216656	40	United-States	0	0	0	0	1	0
4	28	Bachelors	13	Prof-specialty	0	-0.145918	-0.216656	40	Cuba	0	0	0	0	1	0

Анализ и подготовка данных Ames Iowa Housing Data

Проблемы данных:

1. Данные типа object
2. Затраты по памяти

```
memory_analyse(train)
```

```
Average memory usage for float columns: 0.02 MB
Average memory usage for int columns: 0.02 MB
Average memory usage for object columns: 0.12 MB
```

3. Пропущенные данные

Анализ и подготовка

1. Для начала уменьшу затраты по памяти.

```
for col in train.columns:
    if train[col].dtype == float:
        test[col] = test[col].astype('float32')
        train[col] = train[col].astype('float32')
    if (train[col].dtype == int) & (col != 'PID') & (col != 'SalePrice'):
        test[col] = test[col].astype('int32')
        train[col] = train[col].astype('int32')
```

```
memory_analyse(train)
```

```
Average memory usage for float columns: 0.00 MB
Average memory usage for int columns: 0.01 MB
Average memory usage for object columns: 0.12 MB
```

2. Далее просмотрю информацию по пропущенным данным.

```
train.shape
```

```
(2197, 82)
```

```
find_NaN(train)
```

```
Index(['Lot Frontage', 'Alley', 'Mas Vnr Type', 'Mas Vnr Area', 'Bsmt Qual',
       'Bsmt Cond', 'Bsmt Exposure', 'BsmtFin Type 1', 'BsmtFin SF 1',
       'BsmtFin Type 2', 'BsmtFin SF 2', 'Bsmt Unf SF', 'Total Bsmt SF',
       'Electrical', 'Bsmt Full Bath', 'Bsmt Half Bath', 'Fireplace Qu',
       'Garage Type', 'Garage Yr Blt', 'Garage Finish', 'Garage Cars',
       'Garage Area', 'Garage Qual', 'Garage Cond', 'Pool QC', 'Fence',
       'Misc Feature'],
      dtype='object')
Lot Frontage       362
Alley             2054
Mas Vnr Type        22
Mas Vnr Area        22
Bsmt Qual           67
Bsmt Cond           67
Bsmt Exposure       69
BsmtFin Type 1      67
BsmtFin SF 1         1
BsmtFin Type 2      68
BsmtFin SF 2         1
Bsmt Unf SF          1
Total Bsmt SF        1
Electrical           1
Bsmt Full Bath       1
Bsmt Half Bath       1
Fireplace Qu      1066
Garage Type        120
Garage Yr Blt      122
Garage Finish      122
Garage Cars          1
Garage Area          1
Garage Qual        122
Garage Cond        122
Pool QC           2185
Fence             1778
Misc Feature      2117
dtype: int64
```

Вижу, что есть колонки, в которых порядка половины данных пропущено, поэтому сразу отношу их к неинформативным признакам.

```
unnesessary = ['Alley', 'Pool QC', 'Fence', 'Misc Feature', 'Fireplace Qu']
nesessary = train.columns.drop(unnesessary)
```

3. Для дальнейшего определения значимых признаков вычислю корреляцию между SalePrice и остальными численными данными, задам порог в 0.5.

```python
def corr(df, threshold=0.5):
    s1 = pd.Series(df['SalePrice'])
    corr_cols = []
    uncorr_cols = []
    for col in df.columns:
        if df[col].dtype != object:
            s2 = pd.Series(df[col])
            #print(s2)
            c = s1.corr(s2 )
            if abs(c) >= threshold:
                print("corr {0} with price: {1}".format(col, c))
                corr_cols.append(col)
            else:
                uncorr_cols.append(col)
    return corr_cols, uncorr_cols
```

```python
most_corr_cols, not_corr = corr(train)
unnesessary = unnesessary + not_corr
nesessary = train.columns.drop(unnesessary)
unnesessary
```

```
corr Overall Qual with price: 0.7999283301254436
corr Year Built with price: 0.5599749453883133
corr Year Remod/Add with price: 0.5313409904270119
corr Mas Vnr Area with price: 0.5137578573388994
corr Total Bsmt SF with price: 0.629263322130809
corr 1st Flr SF with price: 0.6191638606412259
corr Gr Liv Area with price: 0.6996746308182467
corr Full Bath with price: 0.5478145643173974
corr Garage Yr Blt with price: 0.5185544734947679
corr Garage Cars with price: 0.6440907424901234
corr Garage Area with price: 0.6374274217626136
corr SalePrice with price: 0.9999999999999998

['Alley',
 'Pool QC',
 'Fence',
 'Misc Feature',
 'Fireplace Qu',
 'Order',
 'PID',
 'MS SubClass',
 'Lot Frontage',
 'Lot Area',
 'Overall Cond',
 'BsmtFin SF 1',
 'BsmtFin SF 2',
 'Bsmt Unf SF',
 '2nd Flr SF',
 'Low Qual Fin SF',
 'Bsmt Full Bath',
 'Bsmt Half Bath',
 'Half Bath',
 'Bedroom AbvGr',
 'Kitchen AbvGr',
 'TotRms AbvGrd',
 'Fireplaces',
 'Wood Deck SF',
 'Open Porch SF',
 'Enclosed Porch',
 '3Ssn Porch',
 'Screen Porch',
 'Pool Area',
 'Misc Val',
 'Mo Sold',
 'Yr Sold']
```

4. Стратегия по изменению данных будет подобна той, что использовалась на предыдущем наборе данных, исключением будет то, что нормализировать буду все численные колонки.

5. Для определения значений для заполнения пропусков я отстроила графики распределения значений для всех колонок, а так же сделала кросстаблицы с различными признаками (подробнее в ноутбуке, так как очень много колонок и информации).

   Результаты:

   Колонки, заполняемые *медианами*: Mas Vnr Area, Garage Yr Blt.

   Колонки, заполняемые *наиболее часто встречающимся значением*: Bsmt Cond, Bsmt Exposure, BsmtFin Type 2, Electrical, Garage Qual, Garage Cond.

   Колонки, заполняемые *средним значением*: Total Bsmt SF, Garage Area.

   Garage Finish: заполняю «Unf».

   Bsmt Qual: при Garage Finish == Unf заполняю «TA», иначе «Gd»

   Garage Type: при Garage Cars != 1 заполняю 'Attchd' , при Garage Cars != 1 и Garage Qual != TA — 'Detchd', при Garage Cars == 1 и Garage Qual == TA и Garage Finish == Unf — 'Detchd', во всех остальных случаях — 'Attchd'.

   BsmtFin Type 1: при BsmtFin SF 1 <= 352.75 заполняю «Unf», иначе «GLQ».

   Mas Vnr Type: при Mas Vnr Area <= 50 заполняю «None», иначе 'BrkFace'.

   Garage Cars: заполняю значением 2.

6. Провожу преобразование данных.

```python
def delete_NaN(df):
    cols_med = ['Mas Vnr Area', 'Garage Yr Blt']
    cols_top = ['Bsmt Cond', 'Bsmt Exposure', 'BsmtFin Type 2', 'Electrical', 'Garage Qual', 'Garage Cond']
    cols_mean = ['Total Bsmt SF', 'Garage Area']

    df['Garage Cars'][df['Garage Cars'].isnull()] = 2

    for col in cols_med:
        df[col][df[col].isnull()] = df[col].median()
    for col in cols_top:
        df[col][df[col].isnull()] = df.describe(include=['object']).loc['top', col]
    for col in cols_mean:
        df[col][df[col].isnull()] = df[col].mean()

    df['Mas Vnr Type'][(df['Mas Vnr Area'] <= 50) & df['Mas Vnr Type'].isnull()] = 'None'
    df['Mas Vnr Type'][(df['Mas Vnr Area'] > 50) & df['Mas Vnr Type'].isnull()] = 'BrkFace'

    df['BsmtFin Type 1'][(df['BsmtFin SF 1'] <= 352.75) & df['BsmtFin Type 1'].isnull()] = 'Unf'
    df['BsmtFin Type 1'][(df['BsmtFin SF 1'] > 352.75) & df['BsmtFin Type 1'].isnull()] = 'GLQ'

    """
    RFn = df['Garage Finish'][df['Garage Yr Blt'] > 1955].value_counts()['RFn']
    Unf = df['Garage Finish'][df['Garage Yr Blt'] > 1955].value_counts()['Unf']
    Fin = df['Garage Finish'][df['Garage Yr Blt'] > 1955].value_counts()['Fin']
    j = df.columns.get_loc('Garage Finish')
    count = df['Garage Finish'][df['Garage Yr Blt'] > 1955].count()
    df['Garage Finish'][(df['Garage Yr Blt'] == 2207) & df['Garage Finish'].isnull()] = 'Unf'
    df['Garage Finish'][(df['Garage Yr Blt'] <= 1955) & df['Garage Finish'].isnull()] = 'Unf'
    for i in range(df.shape[0]):
        if (df.iloc[i,j] == None):
            r = random.uniform(0, 1)
            if (r < RFn / count):
                df.iloc[i,j] = 'RFn'
            elif (r >= RFn / count) & (r < Unf / count):
                df.iloc[i,j] = 'Unf'
            else:
                df.iloc[i,j] = 'Fin'
    """

    df['Garage Finish'][df['Garage Finish'].isnull()] = 'Unf'

    df['Bsmt Qual'][(df['Garage Finish'] == 'Unf') & df['Bsmt Qual'].isnull()] = 'TA'
    df['Bsmt Qual'][(df['Garage Finish'] != 'Unf') & df['Bsmt Qual'].isnull()] = 'Gd'

    df['Garage Type'][(df['Garage Cars'] != 1) & df['Garage Type'].isnull()] = 'Attchd'
    df['Garage Type'][(df['Garage Cars'] == 1) & (df['Garage Qual'] != 'TA') & df['Garage Type'].isnull()] = 'Detchd
    df['Garage Type'][(df['Garage Cars'] == 1) & (df['Garage Qual'] == 'TA') & (df['Garage Finish'] == 'Unf')] = 'De
    df['Garage Type'][(df['Garage Cars'] == 1) & (df['Garage Qual'] == 'TA') & (df['Garage Finish'] != 'Unf') & df['
```

```python
def one_hot(df, col):
    dfs = pd.get_dummies(df[col], prefix=col)
    df_ = pd.concat([df, dfs], axis=1)
    df_ = df_.drop(col, axis=1)
    return df_

def transform_df(df):
    unique_counts = pd.DataFrame.from_records([(col, df[col].dtype, df[col].nunique()) for col in df.columns],
                          columns=['Column_Name', 'Type', 'Num_Unique']).sort_values(by=['Num_Unique'])
    df_ = df.copy()
    for col in df.columns:
        if (df_[col].dtype == 'int32') | (df_[col].dtype == 'float32'):
            df_[col] = (df_[col] - df_[col].mean()) / df_[col].std()
        elif (df_[col].dtype == object) & (unique_counts[unique_counts['Column_Name'] == col][['Num_Unique']].values
            values = df_[col].unique()
            df_[col].replace(values, [1, 0], inplace=True)
        elif (df_[col].dtype == object) & (unique_counts[unique_counts['Column_Name'] == col][['Num_Unique']].values
            df_ = one_hot(df_, col).copy()
        elif (df_[col].dtype == object) & (unique_counts[unique_counts['Column_Name'] == col][['Num_Unique']].values
            df_[col] = df_[col].astype('category')
    return df_
```

```
X = train.copy()
delete_NaN(X)
X = train.drop(unnesessary, axis=1)
X = X.drop('SalePrice', axis=1)
X = transform_df(X).copy()
X.head()
```

	Street	Neighborhood	Overall Qual	Year Built	Year Remod/Add	Exterior 1st	Exterior 2nd	Mas Vnr Area	Total Bsmt SF	Central Air	1st Flr SF	Gr Liv Area	Full Bath	Garage Yr Blt	Gara Ca
0	1	SawyerW	0.617984	1.148406	1.063196	VinylSd	VinylSd	-0.568429	0.341658	1	0.116160	-0.566331	-1.070984	1.091488	0.2860
1	1	SawyerW	0.617984	0.590115	0.297582	HdBoard	HdBoard	0.697131	2.400642	1	2.572184	1.344406	0.736199	0.431389	1.5973
2	1	Timber	-0.081506	-0.460786	-1.281498	HdBoard	HdBoard	2.357503	-0.532315	1	0.432580	-0.320163	-1.070984	-0.811150	0.2860
3	1	NridgHt	2.016964	1.181247	1.111047	VinylSd	VinylSd	2.135759	0.852023	1	0.713842	-0.101346	0.736199	1.130317	0.2860
4	1	Gilbert	0.617984	1.049884	0.919643	VinylSd	VinylSd	-0.568429	-1.485143	1	-1.074183	-0.261551	0.736199	0.974999	0.2860

```
y = train['SalePrice']
test_ = test.copy()
delete_NaN(test_)
test_ = train.drop(unnesessary, axis=1)
test_ = transform_df(test_).copy()
test_.head()
```

	Street	Neighborhood	Overall Qual	Year Built	Year Remod/Add	Exterior 1st	Exterior 2nd	Mas Vnr Area	Total Bsmt SF	Central Air	1st Flr SF	Gr Liv Area	Full Bath	Garage Yr Blt	Gara Ca
0	1	SawyerW	0.617984	1.148406	1.063196	VinylSd	VinylSd	-0.568429	0.341658	1	0.116160	-0.566331	-1.070984	1.091488	0.2860
1	1	SawyerW	0.617984	0.590115	0.297582	HdBoard	HdBoard	0.697131	2.400642	1	2.572184	1.344406	0.736199	0.431389	1.5973
2	1	Timber	-0.081506	-0.460786	-1.281498	HdBoard	HdBoard	2.357503	-0.532315	1	0.432580	-0.320163	-1.070984	-0.811150	0.2860
3	1	NridgHt	2.016964	1.181247	1.111047	VinylSd	VinylSd	2.135759	0.852023	1	0.713842	-0.101346	0.736199	1.130317	0.2860
4	1	Gilbert	0.617984	1.049884	0.919643	VinylSd	VinylSd	-0.568429	-1.485143	1	-1.074183	-0.261551	0.736199	0.974999	0.2860