**Лекция 2.**

**Тема: Основные этапы работы с данными**

В академической и индустриальной практике укоренилась простая истина: качество данных определяет качество модели. Самый совершенный алгоритм, обученный на нерепрезентативных, зашумленных или некорректно подготовленных данных, продемонстрирует низкую обобщающую способность и приведет к ошибочным выводам. Таким образом, процесс обработки (предобработки) данных не является технической формальностью; это критически важный этап исследовательского цикла, направленный на преобразование "сырых" данных в чистый, структурированный и информативный набор признаков.

Основная цель – обеспечить выполнение ключевых предположений, заложенных в большинство алгоритмов машинного обучения, таких как:

Непротиворечивость и отсутствие артефактов в данных.

Релевантность признаков целевой переменной.

Соответствие масштаба и распределения требований конкретного алгоритма.

Пренебрежение этим действиям – наиболее распространенная причина неудачных проектов в Data Science.

## 1: Первичный анализ и разведочный анализ данных (Exploratory Data Analysis - EDA)

Цель: Получить первое интуитивное понимание набора данных, выявить очевидные закономерности, аномалии и проблемы.
Примеры будут продемонстрированы на наборе данных о пассажирах лайнера «Титаник» [1].

a) ***Загрузка данных из различных источников (CSV, JSON, SQL-базы).***
Методы: `.pandas.read_csv()`, `.read_sql()`, `.read_json()`
Пример:

```
import pandas as pd

data = pd.read_csv('titanic.csv')
```

b) ***Изучение структуры данных: .shape, .info(), .describe().***
Первичный осмотр с помощью методов `.head()`, `.tail()`, `.sample()` для понимания "внешнего вида" данных.

Проверка общей структуры:

`.shape` — количество строк и столбцов.

`.columns` — список названий признаков.

`.dtypes` — типы данных каждого признака

*Примечание:* Несоответствие типов данных (например, числовой столбец, прочитанный как object) — частая проблема, требующая исправления натекущем этапе.

**c) *Анализ типов данных и преобразование типов (например, строки в категории).***

Правильное определение типов данных — критически важный этап предобработки. Несоответствие типа данных семантике признака приводит к:

- Некорректным результатам анализа (например, среднее значение для ID)
- Проблемам с памятью и производительностью
- Ошибкам при построении моделей (алгоритмы требуют определенных типов данных)

Таблица 1. Основные типы данных в pandas

| Тип данных | Описание | Использование |
|---|---|---|
| **object** | Строки/смешанные типы | Текст, категории |
| **int64** | Целые числа | Целочисленные значения |
| **float64** | Числа с плавающей точкой | Непрерывные значения |
| **category** | Категориальный тип | Ограниченный набор значений |
| **datetime64** | Дата и время | Временные ряды |
| **bool** | Логический тип | Бинарные признаки |

Пример 1: Анализ текущих типов данных

```
import pandas as pd

data = pd.read_csv('titanic.csv')
print("Информация о типах данных:")
print(data.dtypes)
print("Детальная информация:")
data.info(memory_usage='deep')
```

Вывод:

```
Информация о типах данных:
pclass       float64
survived     float64
name          object
sex           object
age          float64
sibsp        float64
parch        float64
ticket        object
fare         float64
```

```
cabin         object
embarked      object
boat          object
body         float64
home.dest     object
dtype: object

Детальная информация:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1310 entries, 0 to 1309
Data columns (total 14 columns):
 #   Column     Non-Null Count  Dtype
---  ------     --------------  -----
 0   pclass     1309 non-null   float64
 1   survived   1309 non-null   float64
 2   name       1309 non-null   object
 3   sex        1309 non-null   object
 4   age        1046 non-null   float64
 5   sibsp      1309 non-null   float64
 6   parch      1309 non-null   float64
 7   ticket     1309 non-null   object
 8   fare       1308 non-null   float64
 9   cabin      295 non-null    object
 10  embarked   1307 non-null   object
 11  boat       486 non-null    object
 12  body       121 non-null    float64
 13  home.dest  745 non-null    object
dtypes: float64(7), object(7)
memory usage: 536.9 KB
```

Пример 2: Преобразование строк в категории

```
print(data['sex'].head(10))

print(f"Память до преобразования: {data['sex'] .memory_usage(deep=True)} bytes")
data['sex'] = data['sex'].astype('category')
print(f"Память после преобразования: {data['sex'].memory_usage(deep=True)}
bytes")
print(f"Уникальные категории: {data['sex'].cat.categories}")
```

Вывод:

```
0    female
1      male
2    female
3      male
4    female
5      male
6    female
7      male
8    female
9      male
Name: sex, dtype: object
Память до преобразования: 70473 bytes
Память после преобразования: 1658 bytes
Уникальные категории: Index(['female', 'male'], dtype='object')
```

Пример 3: Обработка числовых данных в строковом формате

```
df = pd.DataFrame(data)

print("Уникальные значения до обработки:")
print(df['boat'].unique())

df['boat'] = pd.to_numeric(df['boat'], errors='coerce')
print("\nПосле преобразования:")
print(df['boat'].dtype)
print(df['boat'][0:10])
```

Вывод:

```
Уникальные значения до обработки:
['2' '11' nan '3' '10' 'D' '4' '9' '6' 'B' '8' 'A' '5' '7' 'C' '14' '5 9'
 '13' '1' '15' '5 7' '8 10' '12' '16' '13 15 B' 'C D' '15 16' '13 15']

После преобразования:
float64
0     2.0
1    11.0
2     NaN
3     NaN
4     NaN
5     3.0
6    10.0
7     NaN
8     NaN
9     NaN
Name: boat, dtype: float64
```

Пример 4: Работа с датами (преобразование к datetime)

```
dates = ['2023-01-15', '15/02/2023', 'March 3, 2023', '2023-04-20']
df = pd.DataFrame({'date_str': dates})

df['date'] = pd.to_datetime(df['date_str'], format='mixed')
print("Результат преобразования дат:")
print(df)
```

Вывод:

```
Результат преобразования дат:
        date_str       date
0     2025-01-15 2025-01-15
1     15/02/2025 2025-02-15
2  March 3, 2025 2025-03-03
3     2025-04-20 2025-04-20
```

Пример 5_1: Создание бинарных признаков

```
data_encoded = data['survived'], data['sex']
data['sex_encoded'] = data['sex'].map({'female': 0, 'male': 1})
# С помощью replace:
# data['sex_encoded'] = data['sex'].replace({'female': 0, 'male': 1})
print("После простого бинарного кодирования:")
print(data[['survived', 'sex', 'sex_encoded']])
```

Вывод:

```
После простого бинарного кодирования:
      survived     sex  sex_encoded
0          1.0  female          0.0
1          1.0    male          1.0
2          0.0  female          0.0
3          0.0    male          1.0
4          0.0  female          0.0
```

```
...         ...     ...           ...
1305        0.0  female           0.0
1306        0.0    male           1.0
1307        0.0    male           1.0
1308        0.0    male           1.0
1309        NaN     NaN           NaN
```

Пример 5_2: Создание бинарных признаков (LabelEncoder из sklearn (для интеграции в пайплайны)

```python
from sklearn.preprocessing import LabelEncoder

data_encoded = data['survived'], data['sex']
data = pd.DataFrame(data_encoded)

label_encoder = LabelEncoder()
data['sex_labelencoded'] = label_encoder.fit_transform(data['sex'])

print("Соответствие категорий числам:")
for i, class_ in enumerate(label_encoder.classes_):
    print(f"  {class_} -> {i}")

print("\nДанные после LabelEncoder:")
print(data[['survived', 'sex', 'sex_labelencoded']])
```

Вывод:

```
Соответствие категорий числам:
  female -> 0
  male -> 1
  nan -> 2

Данные после LabelEncoder:
      survived     sex  sex_labelencoded
0          1.0  female                 0
1          1.0    male                 1
2          0.0  female                 0
3          0.0    male                 1
4          0.0  female                 0
...        ...     ...               ...
1305       0.0  female                 0
1306       0.0    male                 1
1307       0.0    male                 1
1308       0.0    male                 1
1309       NaN     NaN                 2
```

Пример 6. Автоматизированный анализ типов данных

```python
def analyze_data_types(df):
    type_analysis = pd.DataFrame({
        'column': df.columns,
        'dtype': df.dtypes.values,
        'unique_count': [df[col].nunique() for col in df.columns],
        'null_count': df.isnull().sum().values,
    })

    recommend = []
    for _, row in type_analysis.iterrows():
        if row['dtype'] == 'object' and row['unique_count'] < 50:
            recommend.append('Преобразовать в category')
        elif row['dtype'] == 'float64' and
df[row['column']].dropna().apply(float.is_integer).all():
            recommend.append('Преобразовать в int64?')
        else:
            recommend.append('Тип оптимален')
```

```
    type_analysis['recommend'] = recommend
    return type_analysis

type_report = analyze_data_types(data)
print("Отчет по типам данных:")
print(type_report)
```

Вывод:

```
Отчет по типам данных:
       column    dtype  unique_count  null_count      recommend
0      pclass  float64             3           1   Преобразовать в int64?
1    survived  float64             2           1   Преобразовать в int64?
2        name   object          1307           1   Тип оптимален
3         sex   object             2           1   Преобразовать в category
4         age  float64            98         264   Тип оптимален
5       sibsp  float64             7           1   Преобразовать в int64?
6       parch  float64             8           1   Преобразовать в int64?
7      ticket   object           929           1   Тип оптимален
8        fare  float64           281           2   Тип оптимален
9       cabin   object           186        1015   Тип оптимален
10   embarked   object             3           3   Преобразовать в category
11       boat   object            27         824   Преобразовать в category
12       body  float64           121        1189   Преобразовать в int64?
13  home.dest   object           369         565   Тип оптимален
```

Пример 7. Использование `.groupby()`

```
survived = data[data['survived'] == 1]
not_survived = data[data['survived'] == 0]
data.groupby('pclass').survived.value_counts()
```

Вывод:

```
pclass     survived   count
1.0        1.0        200
           0.0        123
2.0        0.0        158
           1.0        119
3.0        0.0        528
           1.0        181
```

### *d) Анализ базовой статистики: среднее, медиана, дисперсия.*

Пример 8. Базовая статистика для числовых переменных

```
for col in numeric_columns:
    if col in df.columns:
        print(f"\n{col.upper()}:")
        print(f"  Среднее: {df[col].mean():.2f}")
        print(f"  Медиана: {df[col].median():.2f}")
        print(f"  Стандартное отклонение: {df[col].std():.2f}")
        print(f"  Дисперсия: {df[col].var():.2f}")
        print(f"  Минимум: {df[col].min():.2f}")
        print(f"  Максимум: {df[col].max():.2f}")
        print(f"  Количество пропусков: {df[col].isnull().sum()}")
```

Вывод:

```
PCLASS:
  Среднее: 2.29
  Медиана: 3.00
  Стандартное отклонение: 0.84
  Дисперсия: 0.70
  Минимум: 1.00
  Максимум: 3.00
```

```
  Количество пропусков: 1

SURVIVED:
  Среднее: 0.38
  Медиана: 0.00
  Стандартное отклонение: 0.49
  Дисперсия: 0.24
  Минимум: 0.00
  Максимум: 1.00
  Количество пропусков: 1
...
```

Пример 9. Корреляционная матрица

```
correlation_matrix = df[numeric_columns].corr()
print(correlation_matrix)
```

Вывод:

```
              pclass  survived       age     sibsp     parch      fare
pclass      1.000000 -0.312469 -0.408106  0.060832  0.018322 -0.558629
survived   -0.312469  1.000000 -0.055513 -0.027825  0.082660  0.244265
age        -0.408106 -0.055513  1.000000 -0.243699 -0.150917  0.178739
sibsp       0.060832 -0.027825 -0.243699  1.000000  0.373587  0.160238
parch       0.018322  0.082660 -0.150917  0.373587  1.000000  0.221539
fare       -0.558629  0.244265  0.178739  0.160238  0.221539  1.000000
```

*Примечание: Корреляционная матрица нужна для визуализации и количественной оценки связи между множеством переменных в наборе данных. Она позволяет выявить, какие переменные положительно, отрицательно или вообще не коррелируют друг с другом, что помогает в анализе, формировании гипотез и построении моделей.*

Пример 10. Статистика по классам пассажиров

```
for pclass in sorted(df['pclass'].unique()):
    class_data = df[df['pclass'] == pclass]
    print(f"\nКласс {pclass} (n={len(class_data)}):")
    print(f"  Выживаемость: {class_data['survived'].mean():.2%}")
    print(f"  Средний возраст: {class_data['age'].mean():.1f} лет")
    print(f"  Средняя цена билета: ${class_data['fare'].mean():.2f}")
```

Вывод:

```
Класс 1.0 (n=323):
  Выживаемость: 61.92%
  Средний возраст: 39.2 лет
  Средняя цена билета: $87.51

Класс 2.0 (n=277):
  Выживаемость: 42.96%
  Средний возраст: 29.5 лет
  Средняя цена билета: $21.18

Класс 3.0 (n=709):
  Выживаемость: 25.53%
  Средний возраст: 24.8 лет
  Средняя цена билета: $13.30

Класс nan (n=0):
  Выживаемость: nan%
  Средний возраст: nan лет
  Средняя цена билета: $nan
```

Результаты:

На основе анализа данных titanic.scv, основные статистические показатели:

1. Большинство пассажиров не выжили (выжило только 38%)
2. Социальная структура — преобладание пассажиров 3-го класса
3. Как очень дешевые (0$), так и экстремально дорогие билеты (512.33$). Это отражает резкое социальное неравенство на борту.
4. Демография — в основном взрослые одиночные путешественники (Средний возраст ≈ 30 лет, PARCH (родители/дети): Среднее = 0.39, медиана = 0 → Большинство без детей/родителей)
5. Проблема данных — возраст имеет много пропусков, что ограничивает анализ

## e) Визуализация EDA: гистограммы, box-plot (ящик с усами), scatter-plot, матрица корреляций.

Для отображения гистограмм можно использовать библиотеку matplotlib. Импорт:

```
import matplotlib.pyplot as plt
```

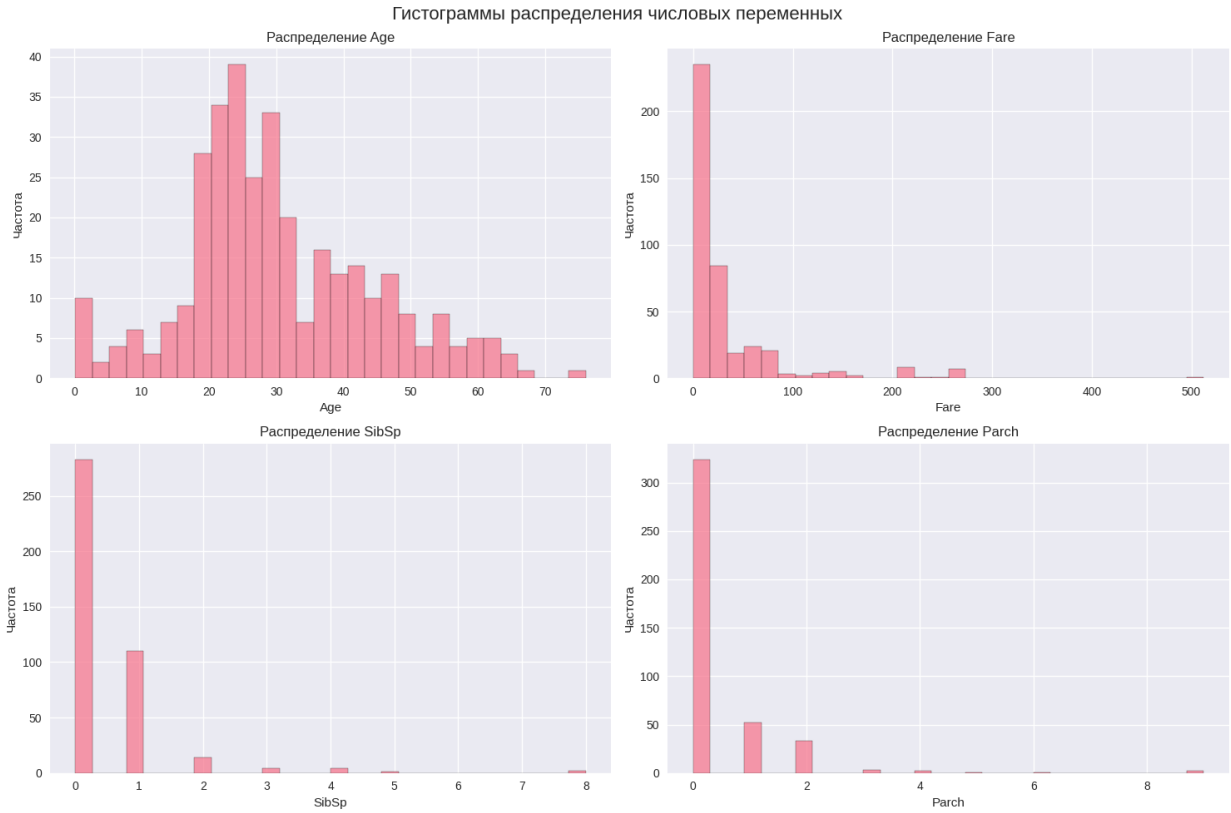
Далее выделяем столбцы, которые следует отразить в гистограмме:

Пример 11. Гистограммы распределения числовых переменных

```
numeric_cols = ['Age', 'Fare', 'SibSp', 'Parch']
fig, axes = plt.subplots(2, 2, figsize=(15, 10))
fig.suptitle('Гистограммы распределения числовых переменных', fontsize=16)

for i, col in enumerate(numeric_cols):
    row, col_idx = i // 2, i % 2
    axes[row, col_idx].hist(df[col].dropna(), bins=30, alpha=0.7,
edgecolor='black')
    axes[row, col_idx].set_title(f'Распределение {col}')
    axes[row, col_idx].set_xlabel(col)
    axes[row, col_idx].set_ylabel('Частота')
plt.tight_layout()
plt.show()
```

Вывод:

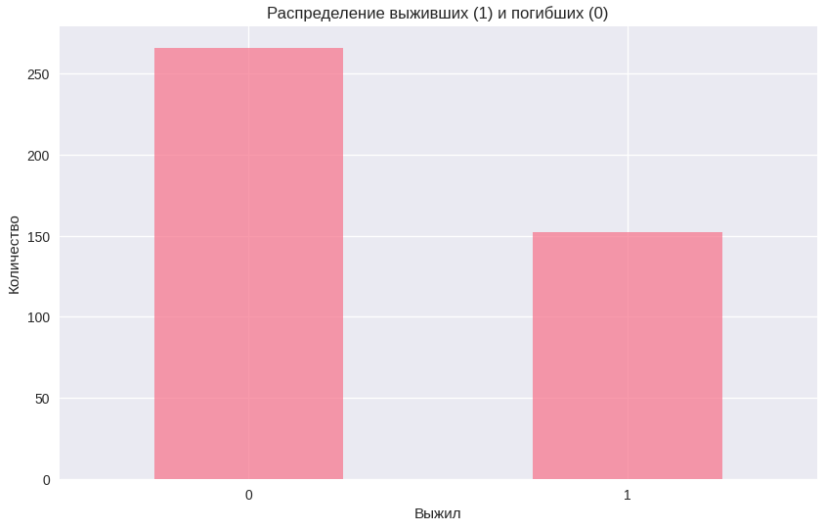



Аналогично построение гистограммы выживаемости по классам:

Пример 12. Гистограмма выживаемости по классам

```
plt.figure(figsize=(10, 6))
df['Survived'].value_counts().plot(kind='bar', alpha=0.7)
plt.title('Распределение выживших (1) и погибших (0)')
plt.xlabel('Выжил')
plt.ylabel('Количество')
plt.xticks(rotation=0)
plt.show()
```

Вывод:

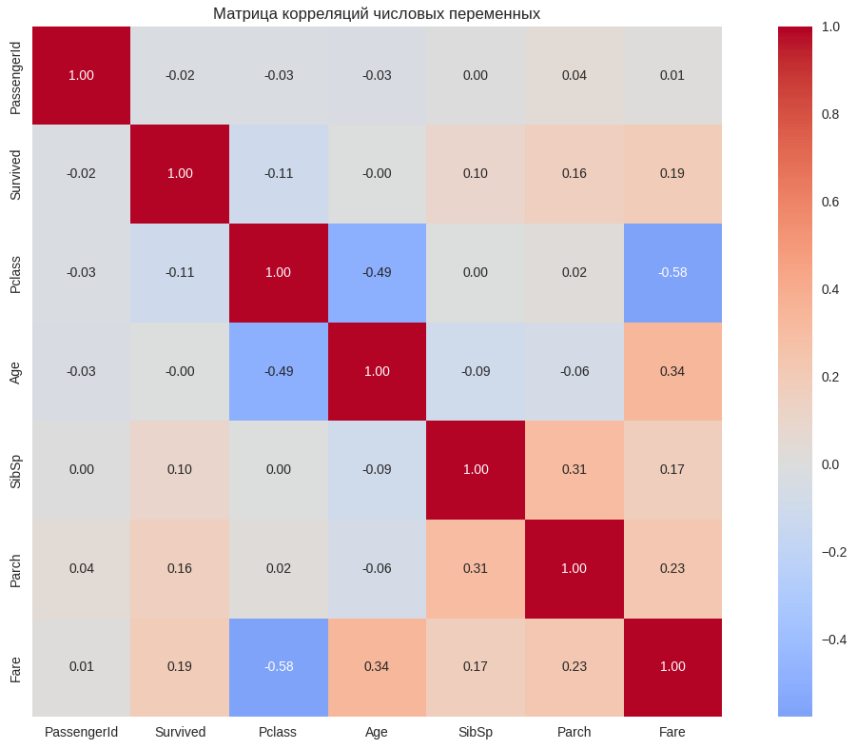



При создании тепловой карты корреляций выбираем только числовые колонки для корреляционного анализа и создаем матрицу корреляций с помощью метода `.corr()`

## Пример 13. Создание тепловой карты корреляций

```
numeric_df = df.select_dtypes(include=[np.number])
correlation_matrix = numeric_df.corr()
plt.figure(figsize=(12, 8))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm',
            center=0, square=True, fmt='.2f')
plt.title('Матрица корреляций числовых переменных')
plt.tight_layout()
plt.show()
```

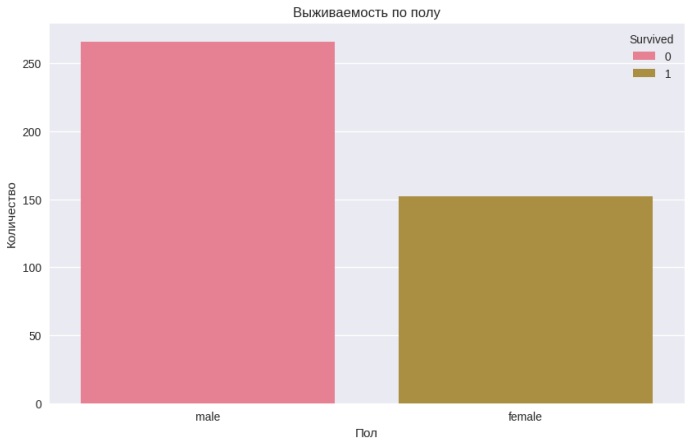
## Вывод:




## Пример 14. Выживаемость по полу

```
plt.figure(figsize=(10, 6))
sns.countplot(x='Sex', hue='Survived', data=df)
plt.title('Выживаемость по полу')
plt.xlabel('Пол')
plt.ylabel('Количество')
plt.show()
```

## Вывод:

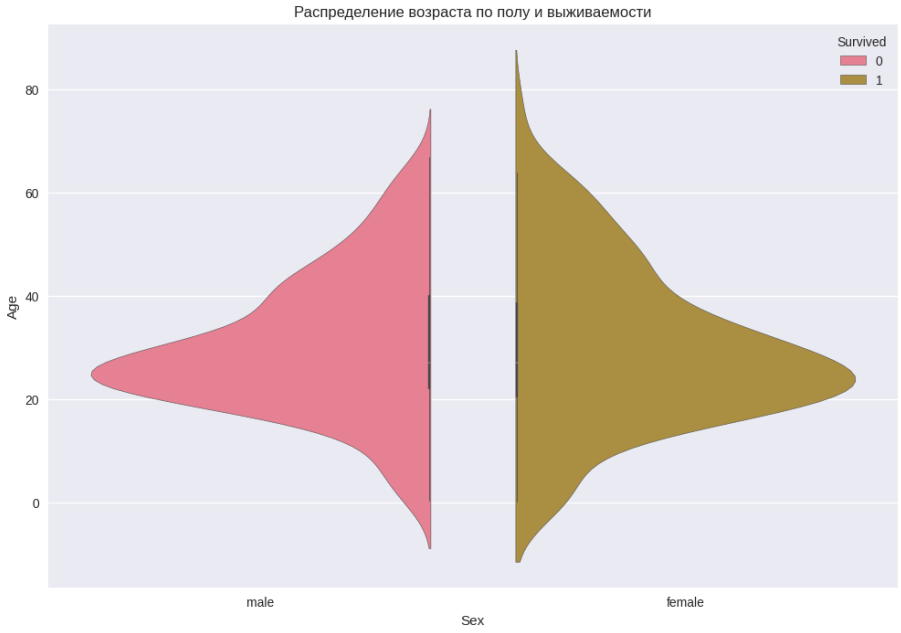


Метод .violinplot() отражает скрипичную диаграмму для более подробного представления о распределении, особенно при наличии нескольких режимов или асимметрии.

Пример 15. Анализ выживаемости по возрасту и полу

```
plt.figure(figsize=(12, 8))
sns.violinplot(x='Sex', y='Age', hue='Survived', data=df, split=True)
plt.title('Распределение возраста по полу и выживаемости')
plt.show()
```

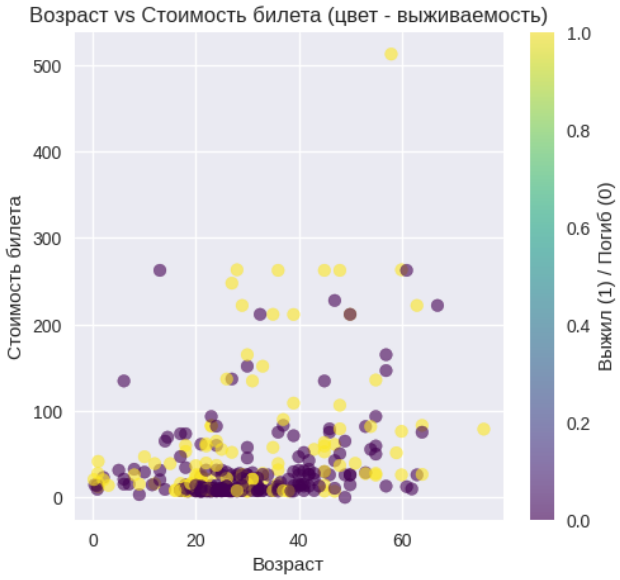
Вывод:




Пример 16. Анализ выживаемости по возрасту и стоимости билета

```
plt.figure(figsize=(12, 5))
plt.subplot(1, 2, 1)
scatter = plt.scatter(df['Age'], df['Fare'], c=df['Survived'],
        alpha=0.6, cmap='viridis')
plt.colorbar(scatter, label='Выжил (1) / Погиб (0)')
plt.xlabel('Возраст')
plt.ylabel('Стоимость билета')
plt.title('Возраст vs Стоимость билета (цвет - выживаемость)')
```

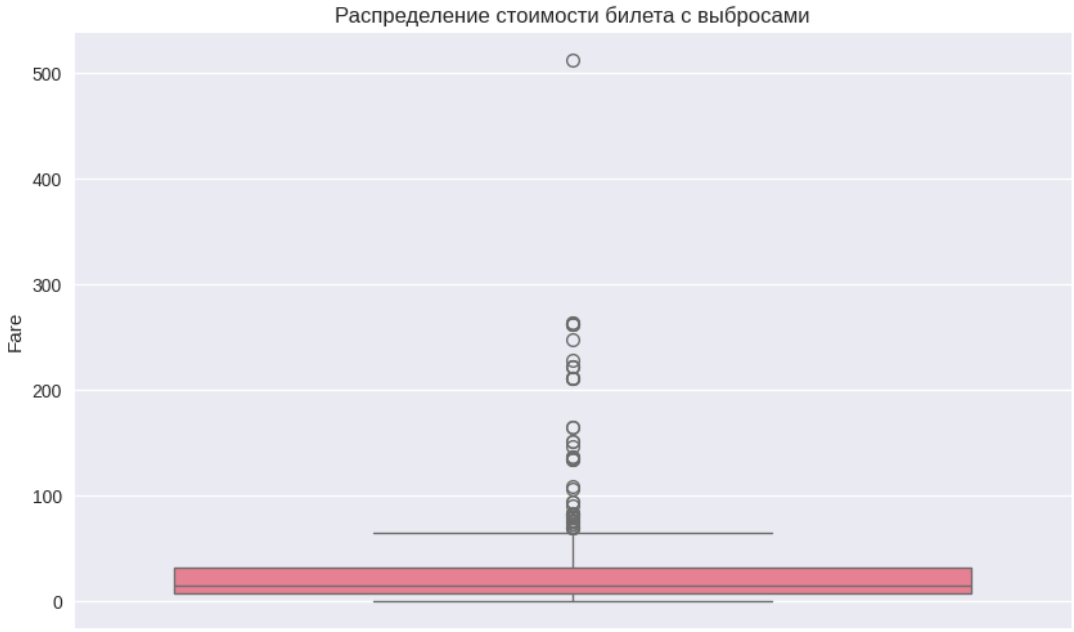
Вывод:



Пример 17. Распределение стоимости билета с выбросами с boxplot

```
plt.figure(figsize=(10, 6))
sns.boxplot(y='Fare', data=df)
plt.title('Распределение стоимости билета с выбросами')
plt.show()
```

Вывод:




## Обработка выбросов методом IQR

Интерквартильный размах (Interquartile range, IQR) - это статистическая мера, которая количественно определяет разброс средних 50% набора данных, т.е. одно из важных применений IQR — обнаружение выбросов. Выбросы — это точки данных, которые значительно выходят за рамки общей картины данных.

Общее правило заключается в определении выбросов как любых точек данных, которые находятся ниже Q1 – 1.5 * IQR или выше Q3 + 1.5 * IQR.

Пример 18. Обработка выбросов методом IQR

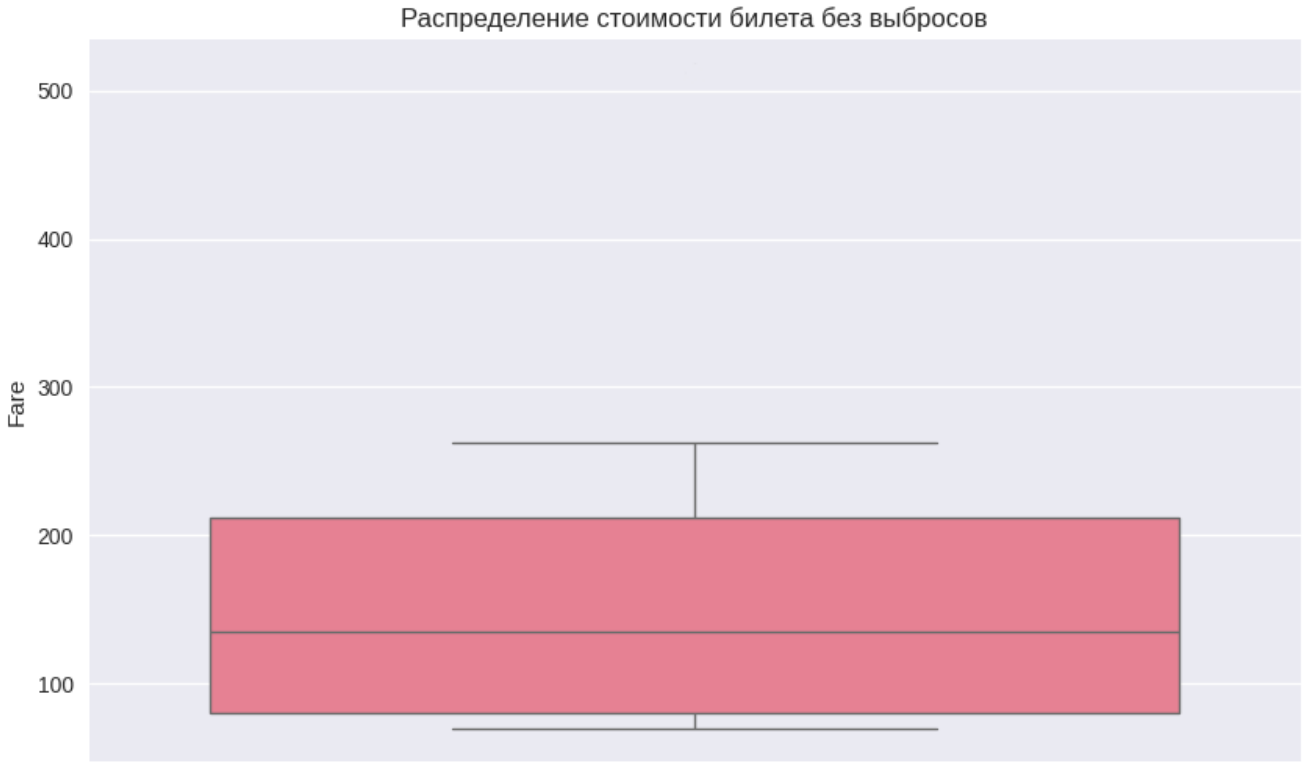
```
Q1 = df['Fare'].quantile(0.25)
Q3 = df['Fare'].quantile(0.75)
IQR = Q3 - Q1
outliers = df[(df['Fare'] < (Q1 - 1.5 * IQR)) | (df['Fare'] > (Q3 + 1.5 * IQR))]
print(f"\nКоличество выбросов в стоимости билета: {len(outliers)}")
print(f"Процент выбросов: {len(outliers)/len(df)*100:.2f}%")
```

Вывод:

```
Количество выбросов в стоимости билета: 55

Процент выбросов: 13.16%
```

```
plt.figure(figsize=(10, 6))
sns.boxplot(y='Fare', data=outliers)
plt.title('Распределение стоимости билета без выбросов')
plt.show()
```




## Заключение

В рамках данного модуля изучен комплексный исследовательский анализ данных (EDA). На примере данных "Titanic" анализ проведёт с помощью гистограмм распределения, матриц корреляции, визуализации и других методов. В процессе работы применены на практике ключевые методы визуального анализа данных.

Освоенные техники EDA являются фундаментальными для специалистов по анализу данных и могут быть применены к широкому спектру задач в различных областях. Полученные навыки позволяют быстро оценивать качество и структуру данных, выявлять аномалии и выбросы, обнаруживать скрытые паттерны и взаимосвязи и формулировать гипотезы для дальнейшего анализа.

Данный модуль закладывает основу для перехода к более сложным методам статистического анализа и машинного обучения, поскольку качественный EDA является критически важным первым этапом любого проекта.

**Литература**


1. Титаник (набор данных) [Электронный ресурс] // Kaggle. — URL: https://www.kaggle.com/datasets/berkinoktay/titanik-csv/data (дата обращения: 21.08.2025).
2. Акимов А. А., Валитов Д. Р., Кубряк А. И. Предварительная обработка данных для машинного обучения //Научное обозрение. Технические науки. – 2022. – №. 2. – С. 26-31.
3. Devert A. Matplotlib Plotting Cookbook. – Packt Publishing Ltd, 2014.