Шпаргалка по Pandas

1. Общие сведения

Pandas — мощная, быстрая и гибкая библиотека для анализа данных на Python. Она предназначена для обработки и анализа табличных данных. В основе `pandas` лежит `numpy`, который обеспечивает высокую производительность при выполнения матричных и векторных операций.

> **Скорость Numpy**
> Numpy реализует матричные преобразования с высокой эффективностью за счет использования непрерывных блоков памяти и нативного кода, что значительно ускоряет вычислительные операции.

Установка Pandas

Для установки `pandas` используйте следующую команду в командной строке:

```
pip install pandas
```

Основные возможности Pandas

- Группировка данных по различным признакам.
- Создание сводных таблиц.
- Удаление дубликатов и невалидных строк или столбцов.
- Фильтрация данных по условиям или уникальности.
- Применение агрегирующих функций: подсчет, сумма, среднее и т.д.
- Визуализация данных (совместимо с библиотеками Matplotlib и Seaborn).

2. Структуры данных в Pandas

Series

`Series` — одномерный массив, содержащий индексированные данные. Каждый элемент имеет индекс, который может быть числовым, строковым или заданным пользователем. Если индексы не заданы, они создаются автоматически.

```
import pandas as pd
```

```python
# Создание Series
s = pd.Series([1, 2, 3, 4], index=['a', 'b', 'c', 'd'])
print(s)
```

DataFrame

`DataFrame` — двумерная структура, аналогичная таблице. Каждая ячейка имеет адрес, состоящий из строки и столбца. Столбцы `DataFrame` представляют собой объекты `Series`.

```python
# Создание DataFrame
data = {
    'Имя': ['Алексей', 'Мария', 'Сергей'],
    'Возраст': [25, 30, 22],
    'Город': ['Москва', 'Санкт-Петербург', 'Екатеринбург']
}

df = pd.DataFrame(data)
print(df)
```

Работа с DataFrame

> **Датасет**
>
> Скачать датасет можно с репозитория урока

```python
# Загрузка данных из CSV файла
df = pd.read_csv('police_stops.csv')

# Просмотр первых нескольких строк и информации о датасете
print(df.head())
print(df.info())
```

Основные методы для работы с DataFrame:

- `.head()` и `.tail()` — вывод первых и последних строк.
- `display()` — полное отображение DataFrame в Jupyter Notebook.
- `.shape` — возвращает количество строк и столбцов.
- `.rename()` — переименование столбцов.
- `.describe()` — выдаёт статистические характеристики.

3. Базовые операции с DataFrame

Примеры операций

1. **Выбор столбцов**:

```
selected_columns = df[['stop_date', 'driver_gender', 'driver_race']]
```

2. **Фильтрация данных**:

```
searches = df[df['search_conducted'] == True]
```

3. **Сортировка данных**:

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

4. Обработка пропущенных значений и типов данных

Обнаружение пропусков

Используйте метод `.isnull()` для поиска пропущенных значений:

```
missing_values = df.isnull().sum()
```

Преобразование типов данных

Преобразование столбца в тип `datetime`:

```
df['stop_date'] = pd.to_datetime(df['stop_date'])
```

Вырезка столбцов

Метод `loc` позволяет вырезать столбцы по меткам:

```
df.loc[:, 'column_name_start':'column_name_end']
```

Метод `iloc` — вырезка по индексам:

```
df.iloc[0:100, 0:5]
```

Преобразование столбца в список

Для извлечения значений столбца в список используйте `tolist()`:

```
names_list = df['Имя'].tolist()
```

Очистка данных

Эта операция подготовит данные к построению на их основе нейронных сетей и моделей. Для этого существуют базовые функции:

- `drop_duplicates()` — удаляет дубликаты, то есть полностью идентичные строки. Этот метод позволяет избежать проблем, возникающих из-за повторяющихся записей, которые могут искажать результаты анализа.
- `fillna(value)` — заменяет пропуски `NaN` на указанное значение, например, на нули, среднее значение или медиану. Это особенно важно для количественных данных, чтобы избежать потерь информации.
- `dropna()` — удаляет строки с `NaN` из таблицы. Этот метод полезен, когда много пропусков и их невозможно корректно заполнить.

Пример кода:

```
# Проверка на наличие пропущенных значений
print(df.isnull().sum())

# Преобразование типа данных столбца с датой
df['stop_date'] = pd.to_datetime(df['stop_date'])

# Заполнение пропусков в возрасте водителя средним значением
df['driver_age'].fillna(df['driver_age'].mean(), inplace=True)

# Удаление дубликатов
df.drop_duplicates(inplace=True)

# Удаление строк с NaN в важном столбце "violation"
df.dropna(subset=['violation'], inplace=True)
```

5. Агрегация и группировка данных

Для анализа остановок по различным параметрам (пол, раса, тип нарушения) можно использовать ряд функций и визуализаций. Распределение остановок по различным характеристикам может дать понимание особенностей выборки и возможных предвзятостей.

Пример кода:

```
import matplotlib.pyplot as plt
import seaborn as sns

# Группировка и подсчет остановок по расе водителя
race_counts = df.groupby('driver_race')
['stop_date'].count().sort_values(ascending=False)

plt.figure(figsize=(12, 6))
sns.countplot(data=df, x='driver_gender')
plt.title('Распределение по полу водителей')
plt.xlabel('Пол')
plt.ylabel('Количество остановок')
plt.show()

# Анализ проведения обысков по полу водителя
search_by_gender = df.groupby('driver_gender')['search_conducted'].mean()

# Топ-5 наиболее частых нарушений
top_violations = df['violation'].value_counts().head()

print("Подсчет остановок по расе:")
print(race_counts)
print("\nПроцент обысков по полу водителя:")
print(search_by_gender)
print("\nТоп-5 нарушений:")
print(top_violations)
```

Дополнительные методы агрегации и группировки:

- **Агрегирование по нескольким параметрам**:

```
aggregation = df.groupby(['driver_gender',
'driver_race']).agg({'stop_date': 'count', 'search_conducted': 'mean'})
print(aggregation)
```

- **Создание сводной таблицы**:

```
pivot_table = pd.pivot_table(df, values='stop_date',
index='driver_gender', columns='driver_race', aggfunc='count',
fill_value=0)
print(pivot_table)
```

- **Визуализация агрегированных данных**:

```
plt.figure(figsize=(12, 8))
sns.heatmap(pivot_table, annot=True, cmap='YlGnBu')
plt.title('Количество остановок по полу и расе водителей')
plt.ylabel('Пол')
plt.xlabel('Раса')
plt.show()
```

6. Визуализация данных (20 минут)

Визуализация данных очень важна для выявления тенденций, паттернов и аномалий в наборах данных. Ниже приведены примеры кода, которые помогут вам создать различные графики для анализа остановок и обысков.

Гистограмма возраста водителей

Это график покажет распределение возрастов водителей.

```
import matplotlib.pyplot as plt
import seaborn as sns

# Гистограмма возраста водителей
plt.figure(figsize=(10, 6))
sns.histplot(data=df, x='driver_age', bins=30, kde=True)
plt.title('Распределение возраста водителей')
plt.xlabel('Возраст водителей')
plt.ylabel('Частота')
plt.show()
```

Столбчатая диаграмма остановок по расе

Эта диаграмма зрительно отобразит количество остановок по расовому признаку водителей.

```
# Столбчатая диаграмма остановок по расе
plt.figure(figsize=(12, 6))
sns.countplot(data=df, x='driver_race',
order=df['driver_race'].value_counts().index)
plt.title('Количество остановок по расе водителя')
plt.xlabel('Раса водителя')
plt.ylabel('Количество остановок')
plt.xticks(rotation=45)
plt.show()
```

Точечная диаграмма: связь между возрастом и продолжительностью остановки

Перед построением важно преобразовать данные о продолжительности остановки в числовой формат.

```
# Преобразование данных о продолжительности остановки в числовой формат
duration_map = {
    "0-15 Min": 15,
    "16-30 Min": 30,
    "30+ Min": 60,
    "2": 2,
    "1": 1
}
df['stop_duration'] = df['stop_duration'].map(duration_map)

plt.figure(figsize=(10, 6))
sns.scatterplot(data=df, x='driver_age', y='stop_duration')
plt.title('Зависимость продолжительности остановки от возраста водителя')
plt.xlabel('Возраст водителя')
plt.ylabel('Продолжительность остановки (мин.)')
plt.show()
```

Круговая диаграмма для распределения рас водителей

Эта диаграмма покажет доли рас водителей в остановках.

```
# Круговая диаграмма для распределения рас водителей
race_distribution = df['driver_race'].value_counts()
plt.figure(figsize=(8, 8))
plt.pie(race_distribution, labels=race_distribution.index, autopct='%1.1f%%',
startangle=90)
plt.title('Распределение рас водителей в остановках')
```

```
plt.axis('equal')  # Для круглой диаграммы
plt.show()
```

Зависимость остановок от месяцев

Этот график демонстрирует количество остановок по месяцам, что позволяет наблюдать тенденции во времени.

```
# Зависимость остановок от месяцев
plt.figure(figsize=(12, 6))
df['year_month'] = df['stop_date'].dt.to_period('M')
monthly_stats = df.groupby('year_month').size()
monthly_stats.plot(kind='bar')
plt.title('Количество остановок по месяцам')
plt.xlabel('Месяц')
plt.ylabel('Количество остановок')
plt.xticks(rotation=45)
plt.show()
```

7. Статистический анализ (15 минут)

Статистический анализ позволяет выявить важные характеристики данных, а также провести проверку гипотез.

Расчет базовых статистик

Для получения основных статистических характеристик количественных значений.

```
# Базовые статистики для числовых столбцов
print(df[['driver_age', 'stop_duration']].describe())
```

Анализ корреляций

Корреляционный анализ позволяет выявить взаимосвязь между различными переменными.

```
# Корреляционный анализ
correlation = df.corr()

plt.figure(figsize=(10, 6))
sns.heatmap(correlation, annot=True, fmt='.2f', cmap='coolwarm', square=True)
```

```
plt.title('Корреляция между переменными')
plt.show()
```

Проверка гипотез

Применим подход к проверке гипотезы о зависимости обыска от расы.

```
from scipy.stats import chi2_contingency

# Проверка зависимости между обысками и расой водителей
contingency_table = pd.crosstab(df['driver_race'], df['search_conducted'])
chi2_stat, p_val, dof, expected = chi2_contingency(contingency_table)

print(f'Chi-square statistic: {chi2_stat}')
print(f'p-value: {p_val}')
```

Дополнительные источники

- Документация Pandas
- Курс по Pandas на Coursera
- Примеры использования Pandas на GitHub

Эти примеры и источники помогут вам более глубоко понять и использовать анализ данных с помощью Python и Pandas. Работая с такими инструментами, вы сможете извлечь важные инсайты из вашего набора данных и визуализировать их для лучшего восприятия.