Федеральное государственное бюджетное образовательное учреждение
высшего образования
«Воронежский государственный технический университет»

МЕТОДИЧЕСКИЕ РЕКОМЕНДАЦИИ

к лабораторным работам по курсу
«Методы машинного обучения»
для студентов направления
09.03.02 Информационные системы и технологии
профиля Информационные системы и технологии цифровизации

Составитель Минаева Ю.В.

Воронеж 2021

Лабораторная работа №1
Основы анализа данных на языке Python

Наука данных является обширной областью исследования с большим количеством областей, из которых анализ данных является неоспоримо один из наиболее важных из всех этих областей, и независимо от своего уровня мастерства в науке данных, она становится все более важной для понимания.

Что такое анализ данных?

Анализ данных - это обработка и преобразование большого количества неструктурированных или неорганизованных данных с целью генерирования ключевой информации об этих данных, которые могли бы помочь в принятии обоснованных решений.

Существуют различные инструменты, используемые для анализа данных, Python, Microsoft Excel, Tableau, SaS и т.Д., Но в этой статье мы сосредоточимся на том, как анализ данных выполняется в python. Более конкретно, как это делается с библиотекой Python под названием Pandas.

Что такое Pandas?

Pandas - это библиотека Python с открытым исходным кодом, используемая для манипулирования данными. Это быстрая и высокоэффективная библиотека с инструментами для загрузки нескольких видов данных в память. Его можно использовать для изменения формы, маркировки среза, индексации или даже группировки нескольких форм данных.

Структуры данных в Pandas

В Pandas есть 3 структуры данных, а именно:

Series

DataFrame

Panel

Лучший способ различить три из них - это видеть, что один содержит несколько стеков другого. Итак, DataFrame - это стек Series, а Panel - это стек DataFrame.

Series - это одномерный массив.

Стек из нескольких Series составляет двухмерный DataFrame

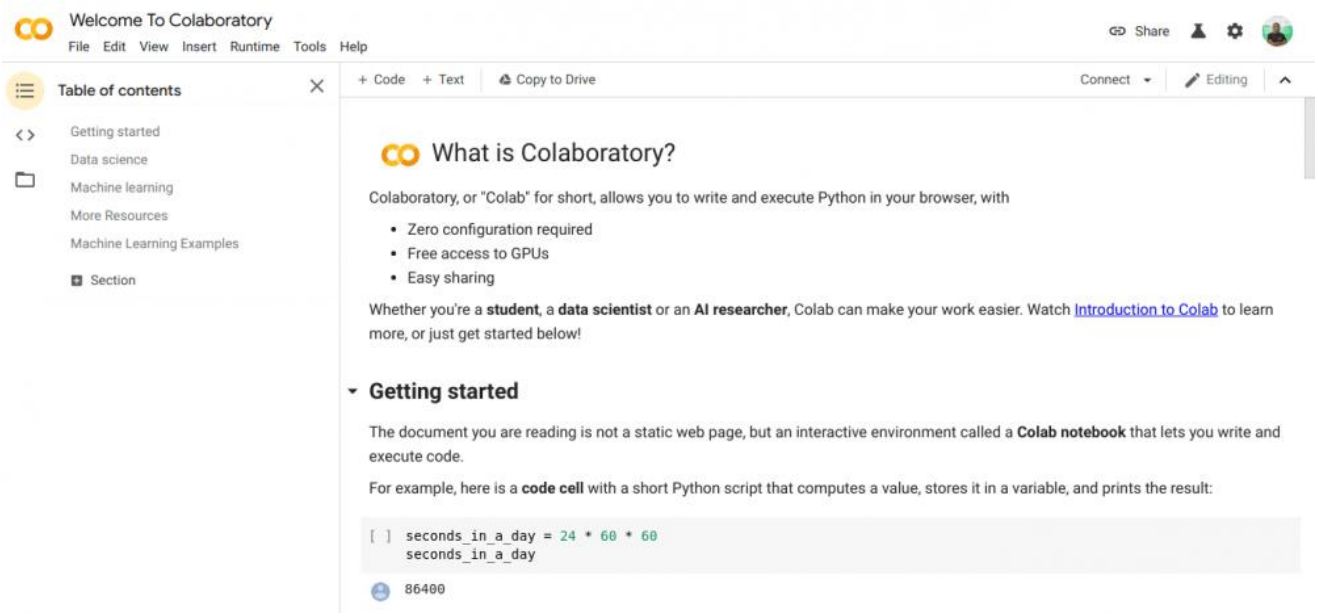
Стек из нескольких DataFrames образует трехмерный Panel

Структура данных, с которой мы будем работать больше всего, - это двухмерный DataFrame, который также может быть средством представления по умолчанию для некоторых наборов данных, с которыми мы можем столкнуться.

Анализ данных в Pandas

Для этой статьи какие-либо установки не требуются. Мы будем использовать инструмент под названием colaboratory, созданный Google. Это онлайн среда Python для анализа данных, машинного обучения и искусственного интеллекта. Это просто облачный Jupyter Notebook, который поставляется с предустановленным почти каждым пакетом Python, который вам понадобится как специалист по данным.

Теперь перейдите на сайт https://colab.research.google.com/notebooks/intro.ipynb. Вы должны увидеть картинку ниже.



В левом верхнем углу, выберите опцию «File» и нажмите «New notebook». Вы увидите новую страницу записной книжки Jupyter, загруженную в ваш браузер. Первое, что нам нужно сделать, это импортировать Pandas в нашу рабочую среду. Мы можем сделать это, с помощью строки:

import pandas as pd

Для этой статьи мы будем использовать набор данных о ценах на жилье для нашего анализа данных. Набор данных, который мы будем использовать, можно найти здесь. Первое, что мы хотели бы сделать, это загрузить этот набор данных в нашу среду.

Мы можем сделать это с помощью следующего кода в новой ячейке;

df = pd.read_csv('https://firebasestorage.googleapis.com/v0/b/ai6-portfolio-abeokuta.appspot.com/o/kc_house_data.csv?alt=media &token=6a5ab32c-3cac-42b3-b534-4dbd0e4bdbc0 ', sep=',')

read_csv Используется, чтобы прочитать файл CSV и мы прошли SEP свойство, чтобы показать, что файл CSV разделяются запятыми.

Также следует отметить, что наш загруженный CSV-файл хранится в переменной df.

Нам не нужно использовать функцию print() в Jupyter Notebook. Мы можем просто ввести имя переменной в нашей ячейке, и Jupyter Notebook распечатает его для нас.

Мы можем попробовать это, набрав df новую ячейку и запустив ее, она распечатает все данные в нашем наборе данных в виде DataFrame для нас.

Но мы не всегда хотим видеть все данные, иногда просто хотим видеть первые несколько данных и имена их столбцов. Мы можем использовать df.head() функцию, чтобы напечатать первые пять столбцов и df.tail() распечатать последние пять. Вывод любого из двух будет выглядеть как таковой;

```
[ ] df.head()
```

	id	date	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfront	view	...	grade	sqft_above	sqft_basement	y
0	7129300520	20141013T000000	221900.0	3	1.00	1180	5650	1.0	0	0	...	7	1180	0	
1	6414100192	20141209T000000	538000.0	3	2.25	2570	7242	2.0	0	0	...	7	2170	400	
2	5631500400	20150225T000000	180000.0	2	1.00	770	10000	1.0	0	0	...	6	770	0	
3	2487200875	20141209T000000	604000.0	4	3.00	1960	5000	1.0	0	0	...	7	1050	910	
4	1954400510	20150218T000000	510000.0	3	2.00	1680	8080	1.0	0	0	...	8	1680	0	

5 rows × 21 columns

Если мы хотим проверить наличие связей между этими несколькими строками и столбцами данных, функция describe() поможет нам в этом.

Запуск df.describe() дает следующий вывод;

```
[ ] df.describe()
```

	id	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfront	view	condition	grade	sq
count	2.161300e+04	2.161300e+04	21613.000000	21613.000000	21613.000000	2.161300e+04	21613.000000	21613.000000	21613.000000	21613.000000	21613.000000	216
mean	4.580302e+09	5.400881e+05	3.370842	2.114757	2079.899736	1.510697e+04	1.494309	0.007542	0.234303	3.409430	7.656873	17
std	2.876566e+09	3.671272e+05	0.930062	0.770163	918.440897	4.142051e+04	0.539989	0.086517	0.766318	0.650743	1.175459	8
min	1.000102e+06	7.500000e+04	0.000000	0.000000	290.000000	5.200000e+02	1.000000	0.000000	0.000000	1.000000	1.000000	2
25%	2.123049e+09	3.219500e+05	3.000000	1.750000	1427.000000	5.040000e+03	1.000000	0.000000	0.000000	3.000000	7.000000	11
50%	3.904930e+09	4.500000e+05	3.000000	2.250000	1910.000000	7.618000e+03	1.500000	0.000000	0.000000	3.000000	7.000000	15
75%	7.308900e+09	6.450000e+05	4.000000	2.500000	2550.000000	1.068800e+04	2.000000	0.000000	0.000000	4.000000	8.000000	22
max	9.900000e+09	7.700000e+06	33.000000	8.000000	13540.000000	1.651359e+06	3.500000	1.000000	4.000000	5.000000	13.000000	94

Сразу видим, что функция describe() дает среднее, стандартное отклонение, минимальное и максимальное значения.

Также можем проверить форму нашего 2D DataFrame, чтобы узнать, сколько у него строк и столбцов. Можем сделать это, используя df.shape() который возвращает кортеж в формате (строки, столбцы).

Мы также можем проверить имена всех столбцов в нашем DataFrame, используя df.columns().

Что если мы хотим выбрать только один столбец и вернуть все данные в нем? Это сделано способом, похожим на прорезание словаря. Введите следующий код в новую ячейку и запустите его

df['price ']

Приведенный выше код возвращает столбец цены, мы можем пойти дальше, сохранив его в новой переменной

price = df['price']

Теперь мы можем выполнить любое другое действие, которое может быть выполнено в DataFrame с нашей ценовой переменной, поскольку оно является лишь подмножеством фактического DataFrame. Мы можем использовать такие функции, как df.head()df.shape() т.д.

Также можем выбрать несколько столбцов, передав список имен столбцов в df как таковой

data = df[['price ', 'bedrooms']]

Приведенный выше выбор столбцов с именами «цена» и «спальни», если мы введем в data.head() новую ячейку, у нас будет следующее

	price	bedrooms
0	221900.0	3
1	538000.0	3
2	180000.0	2
3	604000.0	4
4	510000.0	3

Вышеупомянутый способ нарезки столбцов возвращает все элементы строк в этом столбце, что если мы хотим вернуть подмножество строк и подмножество столбцов из нашего набора данных? Это можно сделать с помощью iloc индексации и аналогично спискам Python. Таким образом, мы можем сделать что-то вроде

df.iloc[50: , 3]

Который возвращает 3-й столбец от 50-го ряда до конца. Это довольно аккуратно и точно так же, как нарезка списков в Python.

Теперь давайте сделаем несколько действительно интересных вещей: в нашем наборе данных о ценах на жилье есть столбец, в котором указывается цена дома, а в другом столбце -

количество спален в конкретном доме. Цена на жилье является постоянной величиной, поэтому возможно, что у нас нет двух домов с одинаковой ценой. Но количество спален несколько дискретно, поэтому у нас может быть несколько домов с двумя, тремя, четырьмя спальнями и т.д.

Что если мы хотим получить все дома с одинаковым количеством спален и определить среднюю цену каждой отдельной спальни? Это относительно легко сделать в Pandas:

df.groupby('bedrooms ')['price '].mean()

Вышеупомянутые сначала группируют DataFrame по наборам данных с идентичным номером спальни, используя df.groupby() функцию, затем мы говорим, что мы даем нам только столбец спальни и используемmean() функцию, чтобы найти среднее значение каждого дома в наборе данных.

Что если мы хотим визуализировать вышесказанное? Мы хотели бы иметь возможность проверить, как меняется средняя цена каждого отдельного номера спальни? Нам просто нужно связать предыдущий код с plot() функцией:

df.groupby('bedrooms ')['price '].mean().plot()

У нас будет вывод, который выглядит таковым;

```
[19] df.groupby('bedrooms')['price'].mean().plot()
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fba95ea1160>
```

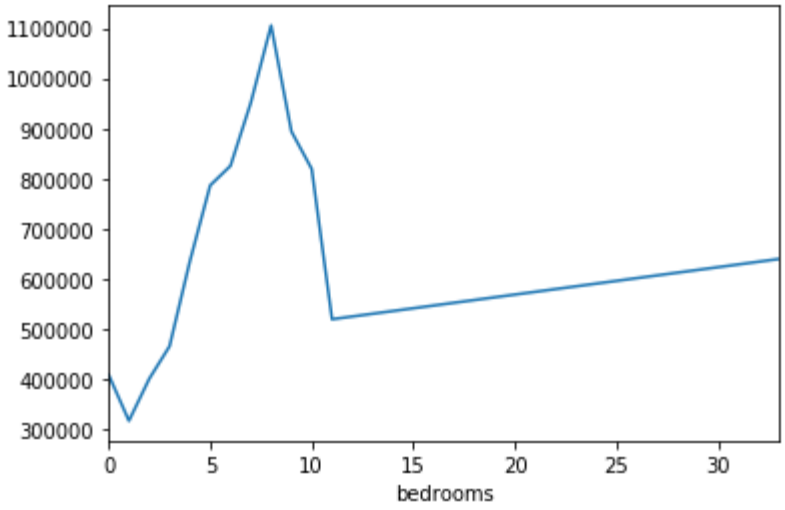


График показывает нам некоторые тенденции в данных. На горизонтальной оси у нас есть различное количество спален (обратите внимание, что более чем один дом может иметь X количество спален). На вертикальной оси мы имеем среднее значение цен в отношении соответствующего количества спален на горизонтальной ось. Теперь мы можем сразу заметить, что дома с 5-10 спальнями стоят намного дороже, чем дома с 3 спальнями. Также станет очевидным, что дома с 7 или 8 спальнями стоят намного больше, чем дома с 15, 20 или даже 30 комнатами.

Информация, подобная вышеприведенной, объясняет, почему анализ данных очень важен, мы можем извлечь полезную информацию из данных, которые не сразу или совсем невозможно заметить без анализа.

Отсутствующие данные

Давайте предположим, что я провожу опрос, который состоит из серии вопросов. Я делюсь ссылкой на опрос с тысячами людей, чтобы они могли высказать свое мнение. Моя конечная цель - провести анализ данных на этих данных, чтобы я мог получить некоторые ключевые выводы из этих данных.

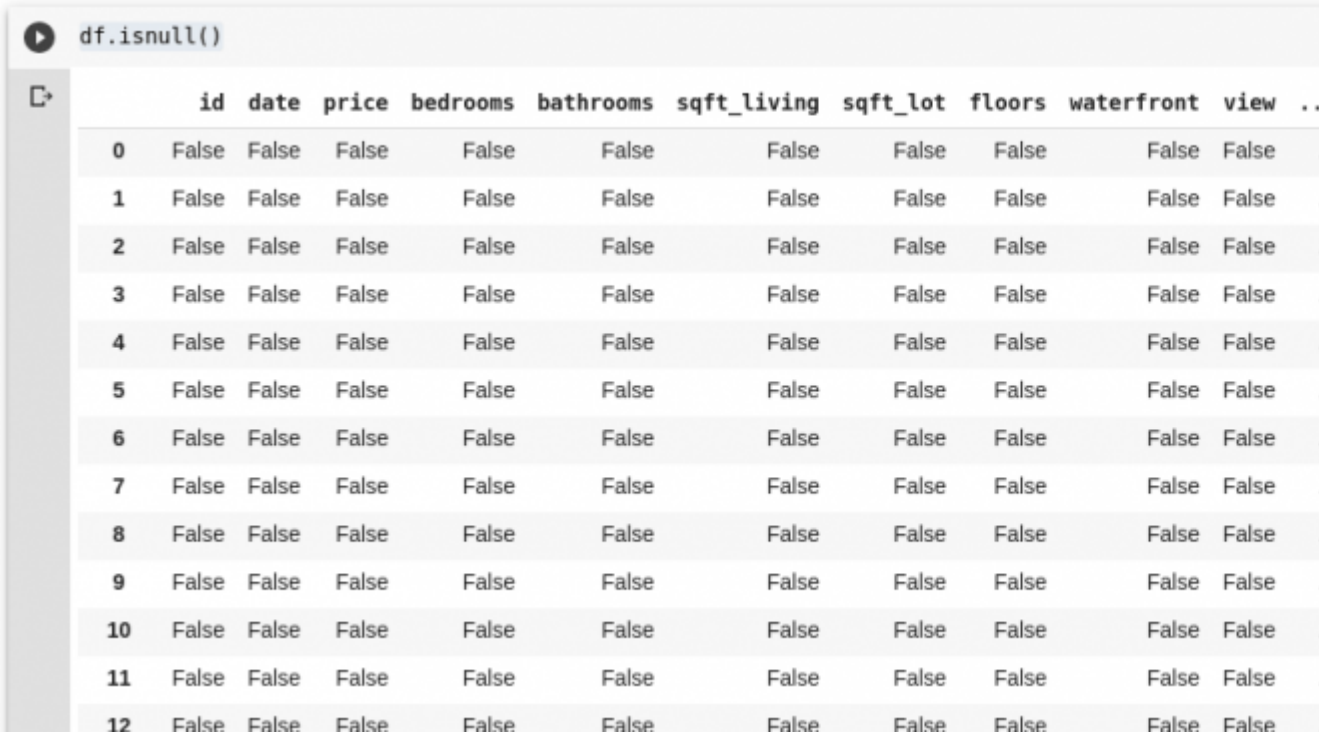
Теперь многое может пойти не так, некоторые геодезисты могут чувствовать себя неловко, отвечая на некоторые мои вопросы, и оставить это поле пустым. Многие люди могут сделать то же самое для нескольких частей моего опроса. Это не может считаться проблемой,

но представьте, что если бы я собирал числовые данные в своем опросе, а часть анализа требовала, чтобы я получил либо сумму, среднее значение, либо какую-то другую арифметическую операцию. Несколько пропущенных значений приведут к большому количеству неточностей в моем анализе, я должен найти способ найти и заменить эти пропущенные значения некоторыми значениями, которые могут быть их близкой заменой.

Pandas предоставляют нам функцию для поиска пропущенных значений в вызываемом DataFrame isnull().

df.isnull()

Оно возвращает DataFrame с логическими значениями, который сообщает нам, действительно ли изначально присутствующие данные отсутствовали. Вывод будет выглядеть таким:



```
df.isnull()
```

	id	date	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfront	view	..
0	False	False	False	False	False	False	False	False	False	False	
1	False	False	False	False	False	False	False	False	False	False	
2	False	False	False	False	False	False	False	False	False	False	
3	False	False	False	False	False	False	False	False	False	False	
4	False	False	False	False	False	False	False	False	False	False	
5	False	False	False	False	False	False	False	False	False	False	
6	False	False	False	False	False	False	False	False	False	False	
7	False	False	False	False	False	False	False	False	False	False	
8	False	False	False	False	False	False	False	False	False	False	
9	False	False	False	False	False	False	False	False	False	False	
10	False	False	False	False	False	False	False	False	False	False	
11	False	False	False	False	False	False	False	False	False	False	
12	False	False	False	False	False	False	False	False	False	False	

Нам нужен способ заменить все эти пропущенные значения, чаще всего выбор пропущенных значений можно принять за ноль. Иногда это может быть принято как среднее значение всех других данных или, возможно, среднее значение данных вокруг него, в зависимости от варианта использования анализируемых данных.

Чтобы заполнить все пропущенные значения в DataFrame, мы используем функцию fillna():

df.fillna(0)

Выше мы заполняем все пустые данные значением ноль. Это может быть любой другой номер, который мы указали.

Важность анализа не может быть переоценена, он помогает нам получить ответы прямо из наших данных! Многие утверждают, что анализ данных - это новая нефть для цифровой экономики.

-
- numpy
- matlibplot
- pandas
- sklearn

Задание: по примеру, рассмотренному в лабораторной работе, провести анализ данных для собственной выборки данных.

Лабораторная работа №2

Введение в машинное обучение на Python

Мы будем использовать датасет цветов ирисов Фишера. Этот датасет известен тем, что он используется практически всеми в качестве "hello world" примера в машинном обучении и статистике.

Набор данных содержит 150 наблюдений за цветами ириса. В датасете есть четыре колонки измерений цветов в сантиметрах. Пятая колонна является видом наблюдаемого цветка.

Все наблюдаемые цветы принадлежат к одному из трех видов. Узнать больше об этом датасете можно в Википедия.

На этом этапе мы загрузим данные из URL-адреса в CSV файл.

2.1 Импорт библиотек

Во-первых, давайте импортировать все модули, функции и объекты, которые мы планируем использовать в этом уроке.

```
# Загрузка библиотек

from pandas import read_csv
from pandas.plotting import scatter_matrix
from matplotlib import pyplot
from sklearn.model_selection import train_test_split
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import StratifiedKFold
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
```

```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
from sklearn.naive_bayes import GaussianNB
from sklearn.svm import SVC
```

Все должно загружаться без ошибок. Если у вас есть ошибка, остановитесь. Перед продолжением необходима рабочая среда SciPy. Посмотрите совет выше о настройке вашей среды.

2.2 Загрузка датасета

Мы можем загрузить данные непосредственно из репозитория машинного обучения UCI.

Мы используем модуль pandas для загрузки данных. Мы также будем использовать pandas чтобы исследовать данные как целей описательной статистики, так для визуализации данных.

Обратите внимание, что при загрузке данных мы указываем имена каждого столбца. Это поможет позже, когда мы будем исследовать данные.

```
# Загрузка датасета
url = "https://raw.githubusercontent.com/jbrownlee/Datasets/master/iris.csv"
names = ['sepal-length', 'sepal-width', 'petal-length', 'petal-width', 'class']
dataset = read_csv(url, names=names)
```

Датасет должен загрузиться без происшествий.

Если у вас есть проблемы с сетью, вы можете скачать файл iris.csv в рабочую директорию и загрузить его с помощью того же метода, изменив URL на локальное имя файла.

Лабораторная работа №3
Анализ датасета

Теперь пришло время взглянуть на данные более детально. На этом этапе мы погрузимся в анализ данные несколькими способами:

- Размерность датасета
- Просмотр среза данных
- Статистическая сводка атрибутов
- Разбивка данных по атрибуту класса.

Не волнуйтесь, каждый взгляд на данные является одной командой. Это полезные команды, которые можно использовать снова и снова в будущих проектах.

3.1 Размерность датасета

Мы можем получить быстрое представление о том, сколько экземпляров (строк) и сколько атрибутов (столбцов) содержится в датасете с помощью метода shape.

```
# shape
print(dataset.shape)
```

Вы должны увидеть 150 экземпляров и 5 атрибутов:

```
(150, 5)
```

3.2 Просмотр среза данных

Исследовании данных, стоит сразу в них заглянуть, для этого есть метод head()

```
# Срез данных head
print(dataset.head(20))
```

Это должно вывести первые 20 строк датасета.

	sepal-length	sepal-width	petal-length	petal-width	class
0	5.1	3.5	1.4	0.2	Iris-setosa
1	4.9	3.0	1.4	0.2	Iris-setosa
2	4.7	3.2	1.3	0.2	Iris-setosa
3	4.6	3.1	1.5	0.2	Iris-setosa
4	5.0	3.6	1.4	0.2	Iris-setosa

5	5.4	3.9	1.7	0.4	Iris-setosa
6	4.6	3.4	1.4	0.3	Iris-setosa
7	5.0	3.4	1.5	0.2	Iris-setosa
8	4.4	2.9	1.4	0.2	Iris-setosa
9	4.9	3.1	1.5	0.1	Iris-setosa
10	5.4	3.7	1.5	0.2	Iris-setosa
11	4.8	3.4	1.6	0.2	Iris-setosa
12	4.8	3.0	1.4	0.1	Iris-setosa
13	4.3	3.0	1.1	0.1	Iris-setosa
14	5.8	4.0	1.2	0.2	Iris-setosa
15	5.7	4.4	1.5	0.4	Iris-setosa
16	5.4	3.9	1.3	0.4	Iris-setosa
17	5.1	3.5	1.4	0.3	Iris-setosa
18	5.7	3.8	1.7	0.3	Iris-setosa
19	5.1	3.8	1.5	0.3	Iris-setosa

3.3 Статистическая сводка

Давайте взглянем теперь на статистическое резюме каждого атрибута. Статистическая сводка включает в себя количество экземпляров, их среднее, мин и макс значения, а также некоторые процентили.

```
# Стастические сводка методом describe
```

```
print(dataset.describe())
```

Мы видим, что все численные значения имеют одинаковую шкалу (сантиметры) и аналогичные диапазоны от 0 до 8 сантиметров.

```
       sepal-length  sepal-width  petal-length  petal-width
count    150.000000   150.000000    150.000000   150.000000
mean       5.843333     3.054000      3.758667     1.198667
std        0.828066     0.433594      1.764420     0.763161
min        4.300000     2.000000      1.000000     0.100000
25%        5.100000     2.800000      1.600000     0.300000
50%        5.800000     3.000000      4.350000     1.300000
75%        6.400000     3.300000      5.100000     1.800000
max        7.900000     4.400000      6.900000     2.500000
```

3.4 Распределение классов

Давайте теперь рассмотрим количество экземпляров (строк), которые принадлежат к каждому классу. Мы можем рассматривать это как абсолютный счет.

```
# Распределение по атрибуту class
print(dataset.groupby('class').size())
```

Мы видим, что каждый класс имеет одинаковое количество экземпляров (50 или 33% от датасета).

```
class
Iris-setosa        50
Iris-versicolor    50
```

```
Iris-virginica    50
dtype: int64
```

3.5 Резюме загрузки датасета

На будущее мы можем объединить все предыдущие элементы вместе в один скрипт..

```
# Загрузка библиотек
from pandas import read_csv

# Загрузка датасета
url = "https://raw.githubusercontent.com/jbrownlee/Datasets/master/iris.csv"
names = ['sepal-length', 'sepal-width', 'petal-length', 'petal-width', 'class']
dataset = read_csv(url, names=names)

# shape
print(dataset.shape)

# Срез данных head
print(dataset.head(20))

# Стастические сводка методом describe
print(dataset.describe())

# Распределение по атрибуту class
```

```
print(dataset.groupby('class').size())
```

Лабораторная работа №4

Визуализация данных

Теперь когда у нас есть базовое представление о данных, давайте расширим его с помощью визуализаций.

Мы рассмотрим два типа графиков:

- Одномерные (Univariate) графики, чтобы лучше понять каждый атрибут.
- Многомерные (Multivariate) графики, чтобы лучше понять взаимосвязь между атрибутами.

4.1 Одномерные графики

Начнем с некоторых одномерных графиков, то есть графики каждой отдельной переменной. Учитывая, что входные переменные являются числовыми, мы можем создавать диаграмма размаха (или "ящик с усами", по-английски "*box and whiskers diagram"*) каждого из них.

```
# Диаграмма размаха
```

```
dataset.plot(kind='box', subplots=True, layout=(2,2), sharex=False, sharey=False)
```

```
pyplot.show()
```

Это дает нам более четкое представление о распределении атрибутов на входе.

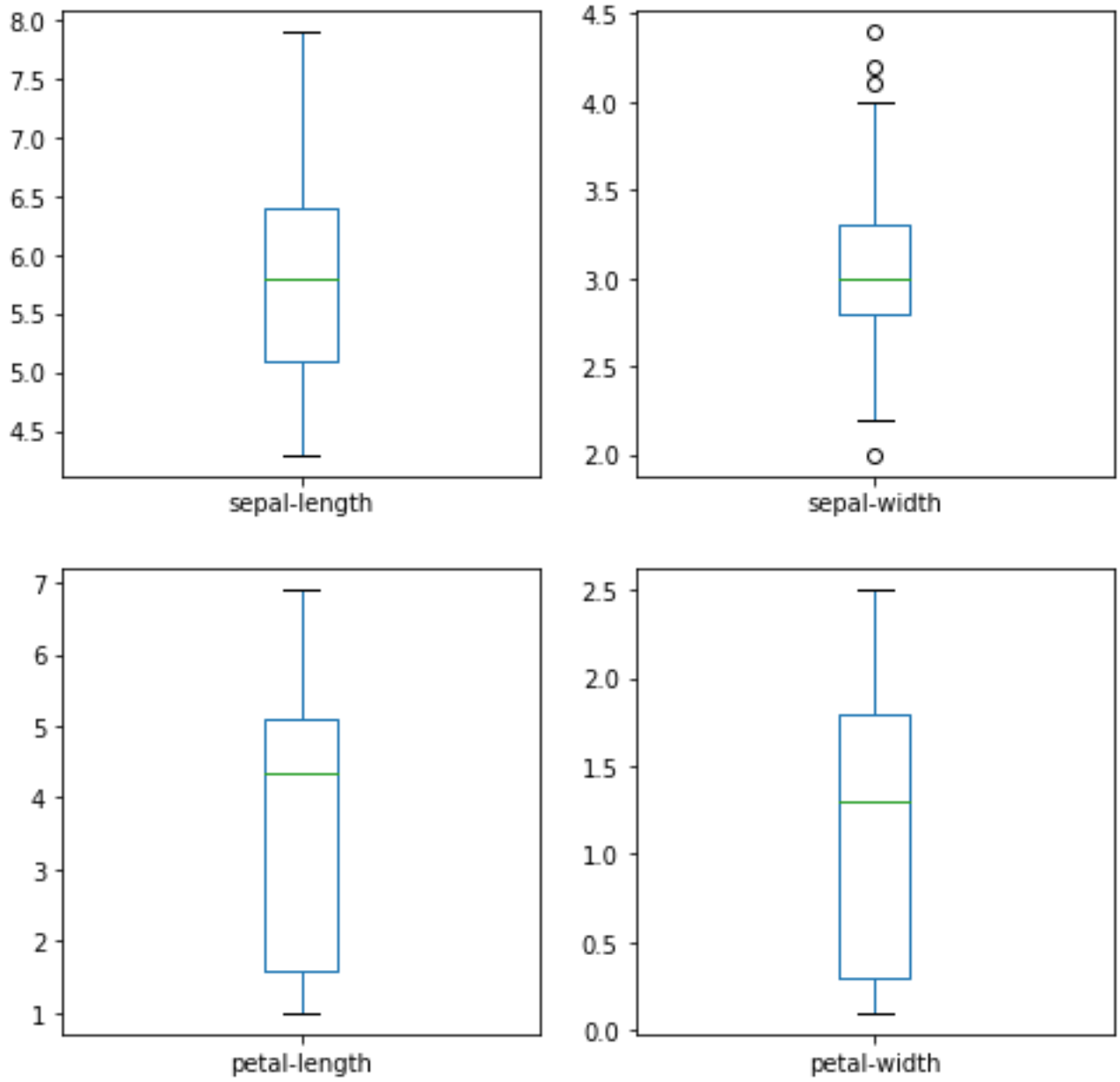
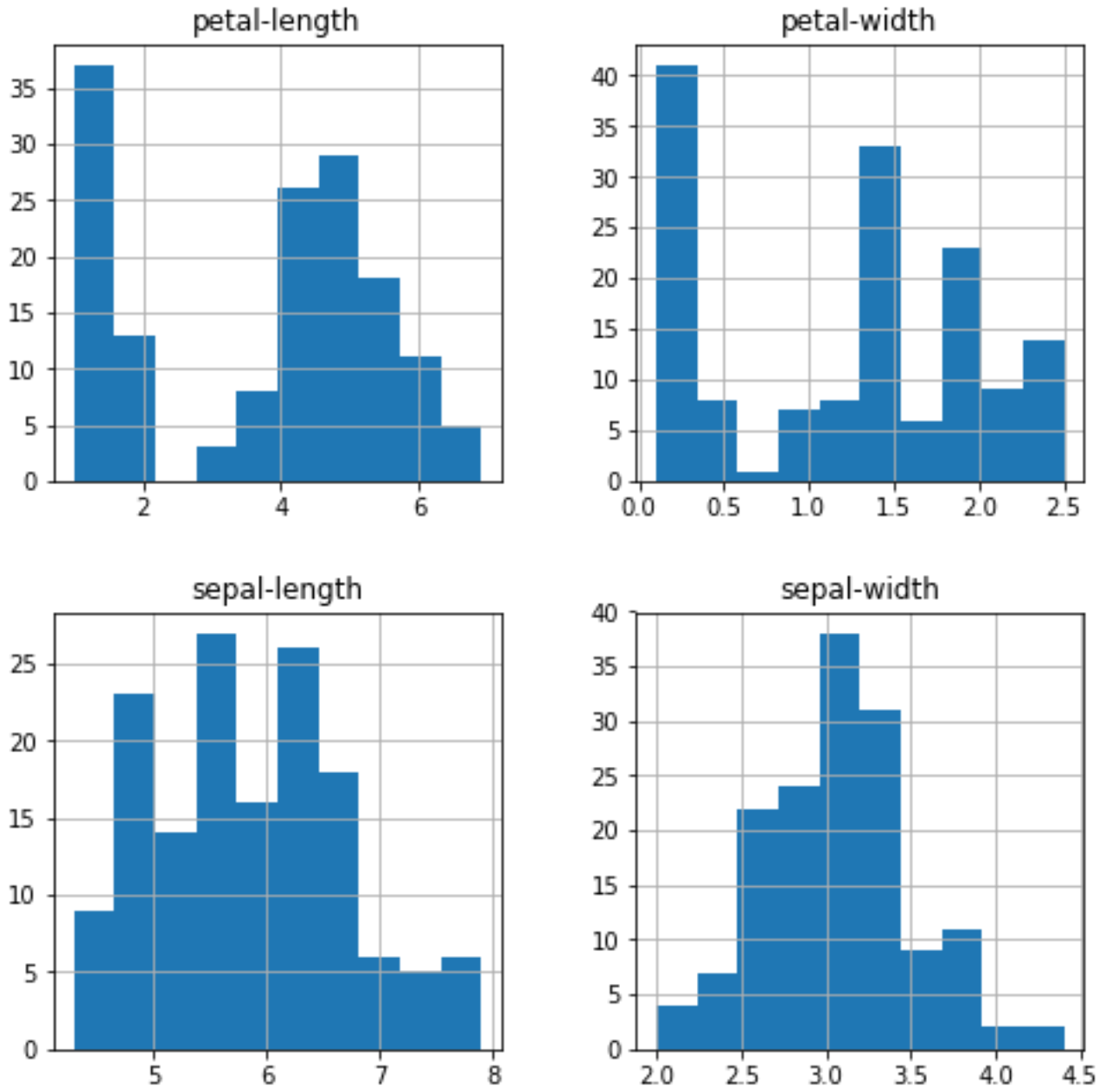


Диаграмма размаха атрибутов входных данных

Мы также можем создать гистограмму входных данных каждой переменной, чтобы получить представление о распределении.

```
# Гистограмма распределения атрибутов датасета
dataset.hist()
pyplot.show()
```

Из графиков видно, что две из входных переменных имеют около гауссово (нормальное) распределение. Это полезно отметить, поскольку мы можем использовать алгоритмы, которые могут использовать это предположение.



Гистограммы входных данных атрибутов датасета

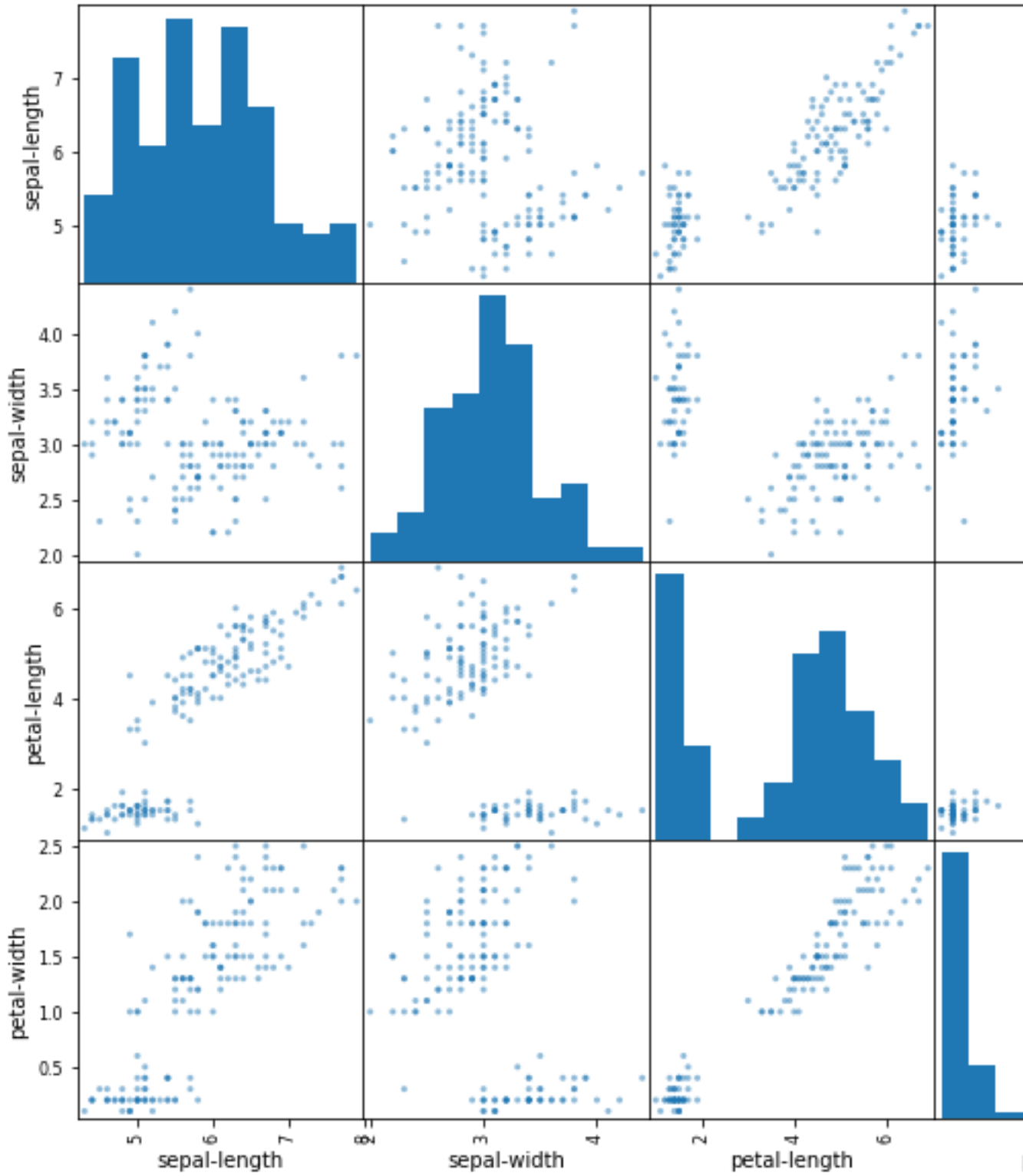
4.2 Многомерные графики

Теперь мы можем посмотреть на взаимодействия между переменными.

Во-первых, давайте посмотрим на диаграммы рассеяния всех пар атрибутов. Это может быть полезно для выявления структурированных взаимосвязей между входными переменными.

```
#Матрица диаграмм рассеяния
scatter_matrix(dataset)
pyplot.show()
```

Обратите внимание на диагональ некоторых пар атрибутов. Это говорит о высокой корреляции и предсказуемой взаимосвязи.



4.3 Резюме визуализации данных

Для справки мы можем связать все предыдущие элементы вместе в один скрипт. Полный пример приведен ниже.

```
# Загрузка библиотек
```

```
from pandas import read_csv
```

```
# Загрузка датасета
url = "https://raw.githubusercontent.com/jbrownlee/Datasets/master/iris.csv"
names = ['sepal-length', 'sepal-width', 'petal-length', 'petal-width', 'class']
dataset = read_csv(url, names=names)

# Диаграмма размаха
dataset.plot(kind='box', subplots=True, layout=(2,2), sharex=False, sharey=False)
pyplot.show()

# Гистограмма распределения атрибутов датасета
dataset.hist()
pyplot.show()

#Матрица диаграмм рассеяния
scatter_matrix(dataset)
pyplot.show()
```

Лабораторная работа №5

Оценка алгоритмов

Теперь пришло время создать некоторые модели данных и оценить их точность на контрольных данных.

Вот что мы собираемся охватить в этом шаге:

- Отделим обучающая выборка от контрольной (тестовой) выборке.
- Настройка 10-кратной кросс-валидации
- Построим несколько различных моделей для прогнозирования класса цветка из измерений цветов
- Выберем лучшую модель.

5.1 Создание контрольной выборки

Мы должны знать, что модель, которую мы создали, хороша.

Позже мы будем использовать статистические методы для оценки точности моделей. Мы также хотим получить более конкретную оценку точности наилучшей модели на контрольных данных, оценив ее по фактических контрольным данным.

То есть, мы собираемся удержать некоторые данные, на которых алгоритмы не будут обучаться, и мы будем использовать эти данные, чтобы получить второе и независимое представление о том, насколько точной может быть лучшая модель на самом деле.

Разделим загруженный датасет на два:

- 80% данных мы будем использовать для обучения, оценки и выбора лучшей среди наших моделей
- 20% данных, что мы будем использовать в качестве контрольного теста качества полученных моделей

```
# Разделение датасета на обучающую и контрольную выборки
array = dataset.values

# Выбор первых 4-х столбцов
X = array[:,0:4]

# Выбор 5-го столбца
y = array[:,4]

# Разделение X и y на обучающую и контрольную выборки
```

```
X_train, X_validation, Y_train, Y_validation = train_test_split(X, y, test_size=0.20,
random_state=1)
```

Теперь у вас есть обучающиеся данные в X_train и Y_train для подготовки моделей и контрольная выборка X_validation и Y_validation, которые мы можем использовать позже.

Обратите внимание, что мы использовали срез в Python для выбора столбцов в массиве NumPy.

5.2 Тестирование проверки

Для оценки точности модели мы будем использовать стратифицированную 10-кратную кросс-валидацию.

Это разделит наш датасет на 10 частей, обучающийся на 9 частях и 1 тестовой проверке и будет повторять обучение на всех комбинаций из выборок train-test.

Стратифицированная означает, что каждый прогон по выборке данных будет стремиться иметь такое же распределение примера по классу, как это существует во всем наборе обучаемых данных.

Для получения дополнительной информации о том как работает метод k-fold кросс-валидации можно посмотреть по ссылке.

Мы устанавливаем случайное затравку через random_state аргумент на фиксированное число, чтобы гарантировать, что каждый алгоритм оценивается на тех же выборках обучающихся данных. Конкретные случайные затравки не имеет значения.

Мы используем метрику «acccuracy» для оценки моделей.

Это соотношение числа правильно предсказанных экземпляров, разделенных на общее количество экземпляров в наборе данных, умноженном на 100, чтобы дать процент (например, точность 95%). Мы будем использовать оценочную переменную, когда мы будем запускать сборку и оценивать каждую модель.

5.3 Строим модели машинного обучения

Мы не знаем, какие алгоритмы будет хороши для этой задачи или какие конфигурации их использовать.

Все что увидела выше, что некоторые классы частично линейно зависят в некоторых измерениях, поэтому в целом ожидаем хорошие результаты.

Давайте протестируем 6 различных алгоритмов:

- Логистическая регрессия или логит-модель (LR)
- Линейный дискриминантный анализ (LDA)
- Метод k-ближайших соседей (KNN)
- Классификация и регрессия с помощью деревьев (CART)
- Наивный байесовский классификатор (NB)
- Метод опорных векторов (SVM)

Здесь приведена смесь простых линейных (LR и LDA), нелинейных (KNN, CART, NB и SVM) алгоритмов.

Давайте построим и оценим наши модели:

```
# Загружаем алгоритмы модели
```

```
models = []
```

```
models.append(('LR', LogisticRegression(solver='liblinear', multi_class='ovr')))
models.append(('LDA', LinearDiscriminantAnalysis()))
models.append(('KNN', KNeighborsClassifier()))
models.append(('CART', DecisionTreeClassifier()))
models.append(('NB', GaussianNB()))
models.append(('SVM', SVC(gamma='auto')))

# оцениваем модель на каждой итерации
results = []
names = []

for name, model in models:
        kfold = StratifiedKFold(n_splits=10, random_state=1, shuffle=True)
        cv_results = cross_val_score(model, X_train, Y_train, cv=kfold,
scoring='accuracy')
        results.append(cv_results)
        names.append(name)
        print('%s: %f (%f)' % (name, cv_results.mean(), cv_results.std()))
```

5.4 Выбираем модель

Теперь у нас есть 6 моделей и оценки точности каждой из них. Мы должны сравнить модели друг с другом и выбрать наиболее точные.

Запустив приведенный выше пример, мы получим следующие необработанные результаты:

```
LR: 0.955909 (0.044337)
LDA: 0.975641 (0.037246)
KNN: 0.950524 (0.040563)
CART: 0.951166 (0.052812)
NB: 0.951166 (0.052812)
SVM: 0.983333 (0.033333)
```

Обратите внимание, что ваши результаты могут немного варьироваться, учитывая стохастический характер алгоритмов обучения.

Как интерпретировать полученные значения качества моделей?

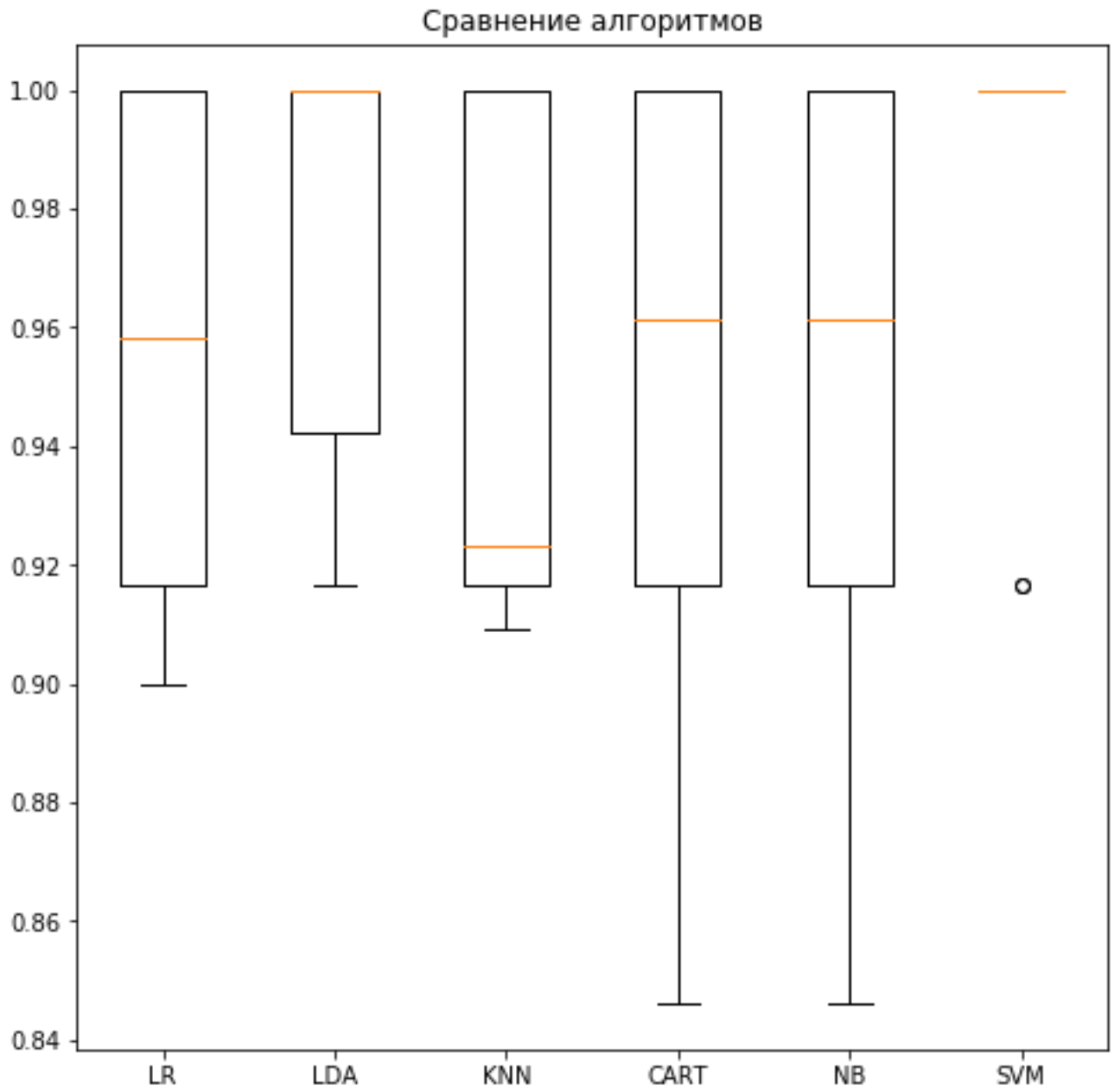
В этом случае, мы видим, что это выглядит как метод опорных векторов (SVM) имеет самый большой расчет точности около 0,98 или 98%.

Мы также можем создать график результатов оценки модели и сравнить расхождение средней точность каждой модели. Существует разбор показателей точности для каждого алгоритма, потому что каждый алгоритм был оценен 10 раз (в рамках 10-кратной кросс-валидации).

Хороший способ сравнить результаты для каждого алгоритма заключается в создании диаграмме размаха атрибутов выходных данных и их усов для каждого распределения и сравнения распределений.

```
# Сравниванием алгоритмы
pyplot.boxplot(results, labels=names)
pyplot.title('Algorithm Comparison')
pyplot.show()
```

Мы видим, что ящики и усы участков в верхней части диапазона достигают 100% точности, а некоторые находятся в районе 85% точности.



5.5 Резюме оценки алгоритмов

Для справки мы можем связать все предыдущие элементы вместе в один скрипт. Полный пример приведен ниже.

```
# Загрузка библиотек
from pandas import read_csv
from pandas.plotting import scatter_matrix
from matplotlib import pyplot
```

```
from sklearn.model_selection import train_test_split
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import StratifiedKFold
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
from sklearn.naive_bayes import GaussianNB
from sklearn.svm import SVC

# Загрузка датасета
url = "https://raw.githubusercontent.com/jbrownlee/Datasets/master/iris.csv"
names = ['sepal-length', 'sepal-width', 'petal-length', 'petal-width', 'class']
dataset = read_csv(url, names=names)

# Разделение датасета на обучающую и контрольную выборки
```

```
array = dataset.values

# Выбор первых 4-х столбцов
X = array[:,0:4]

# Выбор 5-го столбца
y = array[:,4]

# Разделение X и y на обучающую и контрольную выборки
X_train, X_validation, Y_train, Y_validation = train_test_split(X, y, test_size=0.20,
random_state=1)

# Загружаем алгоритмы моделей
models = []
models.append(('LR', LogisticRegression(solver='liblinear', multi_class='ovr')))
models.append(('LDA', LinearDiscriminantAnalysis()))
models.append(('KNN', KNeighborsClassifier()))
models.append(('CART', DecisionTreeClassifier()))
models.append(('NB', GaussianNB()))
models.append(('SVM', SVC(gamma='auto')))
```

```
# оцениваем модель на каждой итерации
results = []
names = []
for name, model in models:
        kfold = StratifiedKFold(n_splits=10, random_state=1, shuffle=True)
        cv_results = cross_val_score(model, X_train, Y_train, cv=kfold,
scoring='accuracy')
        results.append(cv_results)
        names.append(name)
        print('%s: %f (%f)' % (name, cv_results.mean(), cv_results.std()))
# Сравниванием алгоритмы
pyplot.boxplot(results, labels=names)
pyplot.title('Algorithm Comparison')
pyplot.show()
```

Лабораторная работа №6

Прогнозирование данных

Прежде чем что-либо прогнозировать необходимо выбрать алгоритм для прогнозирования. По результатам оценки моделей предыдущего раздела мы выбрали модель SVM как наиболее точную. Мы будем использовать эту модель в качестве нашей конечной модели.

Теперь мы хотим получить представление о точности модели на нашей контрольной выборке данных.

Это даст нам независимую окончательную проверку точности лучшей модели. Полезно сохранить контрольную выборку для случаев когда была допущена ошибки в процессе обучения, такая как переобучение или утечка данных. Обе эти проблемы могут привести к чрезмерно оптимистичному результату.

6.1 Создаем прогноз

Мы можем протестировать модель на всей выборке обучаемых данных и сделать прогноз на контрольной выборке.

```
# Создаем прогноз на контрольной выборке
model = SVC(gamma='auto')
model.fit(X_train, Y_train)
predictions = model.predict(X_validation)
```

6.2 Оцениваем прогноз

Мы можем оценить прогноз, сравнив его с ожидаемым результатом контрольной выборки, а затем вычислить точность классификации, а также матрицу ошибок и отчет о классификации.

```
# Оцениваем прогноз
print(accuracy_score(Y_validation, predictions))
print(confusion_matrix(Y_validation, predictions))
print(classification_report(Y_validation, predictions))
```

Мы видим, что точность 0,966 или около 96% на контрольной выборке.

Матрица ошибок дает представление об одной допущенной ошибке (сумма недиагональных значений).

Наконец, отчет о классификации предусматривает разбивку каждого класса по точности (precision), полнота (recall), f1-оценка, показывающим отличные результаты (при этом контрольная выборка была небольшая, всего 30 значений).

```
0.9666666666666667
[[11  0  0]
 [ 0 12  1]
 [ 0  0  6]]
                 precision    recall  f1-score   support

    Iris-setosa       1.00      1.00      1.00        11
Iris-versicolor       1.00      0.92      0.96        13
 Iris-virginica       0.86      1.00      0.92         6

    avg / total       0.97      0.97      0.97        30
```

6.3 Резюме прогнозирование данных

Для справки мы можем связать все предыдущие элементы вместе в один скрипт. Полный пример приведен ниже.

```
# Загрузка библиотек
from pandas import read_csv
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score
```

```
# Загрузка датасета
url = "https://raw.githubusercontent.com/jbrownlee/Datasets/master/iris.csv"
names = ['sepal-length', 'sepal-width', 'petal-length', 'petal-width', 'class']
dataset = read_csv(url, names=names)

# Разделение датасета на обучающую и контрольную выборки
array = dataset.values

# Выбор первых 4-х столбцов
X = array[:,0:4]

# Выбор 5-го столбца
y = array[:,4]

# Разделение X и y на обучающую и контрольную выборки
X_train, X_validation, Y_train, Y_validation = train_test_split(X, y, test_size=0.20,
random_state=1)

# Создаем прогноз на контрольной выборке
model = SVC(gamma='auto')
model.fit(X_train, Y_train)
```

```
predictions = model.predict(X_validation)

# Оцениваем прогноз
print(accuracy_score(Y_validation, predictions))
print(confusion_matrix(Y_validation, predictions))
print(classification_report(Y_validation, predictions))
```