Создайте новый файл notebook. Для этого нажмите кнопку New в правом верхнем углу и выберите Python 3. Нажмите на заголовок и в открывшемся окне задайте имя Pract01.

Загружаем необходимые для решения задачи библиотеки pandas, numpy, matplotlib.

Для анализа понадобятся модули pandas, numpy и sklearn. С помощью pandas и numpy мы проведем начальный анализ данных, а sklearn поможет в вычислении прогнозной модели.

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

import seaborn as sns

sns.set()

%matplotlib inline

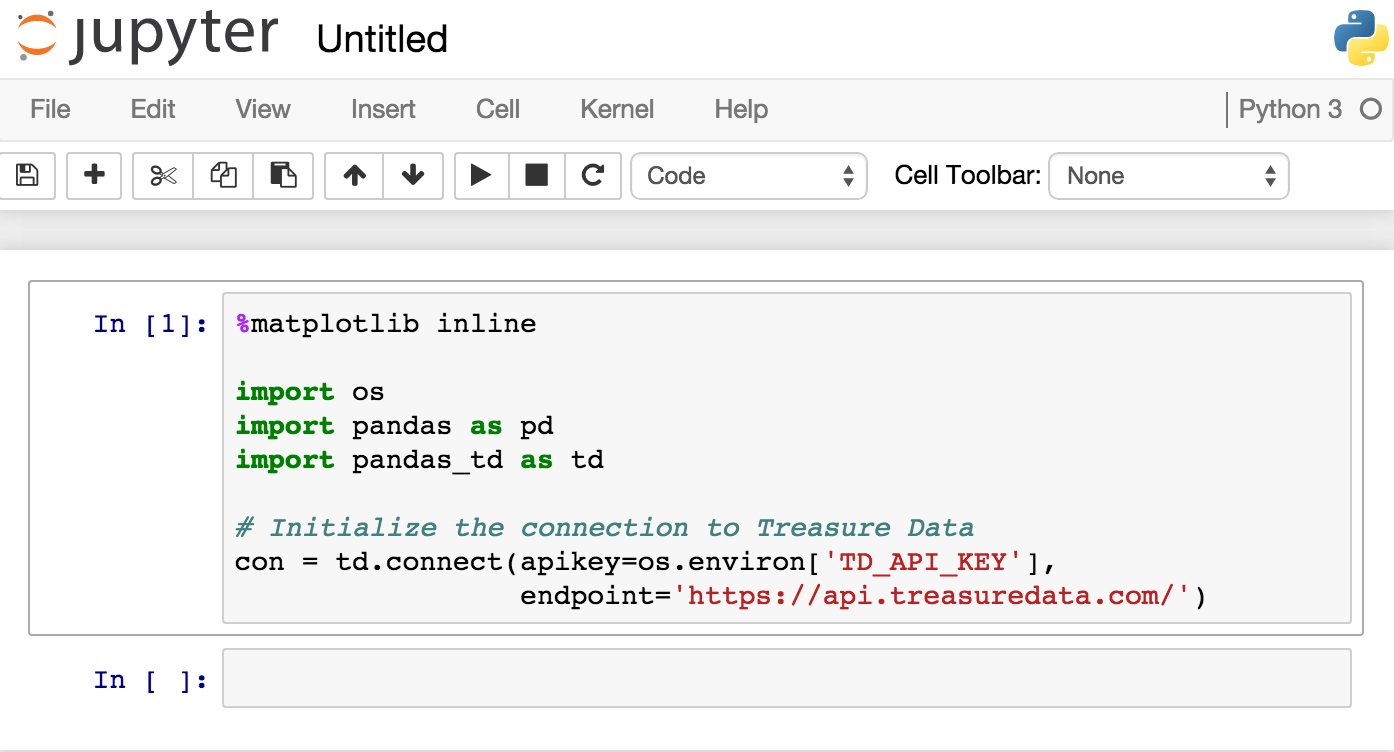
%config InlineBackend.figure\_format = 'png'

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

import seaborn as sns

​ 

sns.set()

%matplotlib inline

%config InlineBackend.figure\_format = 'png'

Этот фрагмент кода выполняет несколько задач. Во-первых, он импортирует различные библиотеки, которые будут использоваться в дальнейшем. Среди них библиотеки pandas, numpy и matplotlib для работы с данными, а также библиотека seaborn для визуализации данных.

Затем код устанавливает параметры seaborn, чтобы обеспечить единообразный и последовательный стиль визуализации данных. Это делается вызовом функции sns.set().

Далее код включает возможность отображения графиков внутри Jupyter Notebook, используя магические команды %matplotlib inline и %config InlineBackend.figure\_format = ‘png’. Это позволяет создавать и отображать графики непосредственно в ячейке Jupyter без необходимости использовать отдельное окно для отображения графика.

После этого код считывает данные из файла CSV с названием “2020.csv” и сохраняет их в переменной df. Затем код создает новую переменную de, которая является копией данных из переменной df.

Наконец, код удаляет индекс из копии данных (переменная de), чтобы упростить дальнейшую работу с данными.

df = pd.read\_csv('2020.csv', dtype={"store\_and\_fwd\_flag": 'string'})

df.shape

df

de = df

de = de.drop(de[de.passenger\_count < 0.01].index)

de = de.drop(de[de.trip\_distance < 0.01].index)

de = de.drop(de[de.fare\_amount < 0.01].index)

de = de.drop(de[de.extra < 0].index)

de = de.drop(de[de.mta\_tax < 0].index)

de = de.drop(de[de.tip\_amount < 0].index)

de = de.drop(de[de.tolls\_amount < 0.01].index)

de = de.drop(de[de.improvement\_surcharge < 0].index)

de = de.drop(de[de.total\_amount < 0.01].index)

de = de.drop(de[de.congestion\_surcharge < 0].index)

de

de['tpep\_pickup\_datetime'] = pd.to\_datetime(de['tpep\_pickup\_datetime'], format='%m/%d/%Y %I:%M:%S %p')

de['tpep\_dropoff\_datetime'] = pd.to\_datetime(de['tpep\_dropoff\_datetime'], format='%m/%d/%Y %I:%M:%S %p')

#de.set\_index('tpep\_pickup\_datetime', inplace = True)

de

de = de.drop(de[de.tpep\_pickup\_datetime < '01/01/2020 00:00:00'].index)

de = de.drop(de[de.tpep\_pickup\_datetime > '31/12/2020 23:59:59'].index)

de.sort\_values(by=['tpep\_pickup\_datetime'])

Данным кодом выполняется обработка данных для дальнейшего анализа. Пользователь работает с датасетом, который содержит информацию о поездках такси в Нью-Йорке. Код считывает данные в формате CSV, преобразует их и проводит предварительную обработку. Затем он визуализирует данные с помощью библиотеки Seaborn и сохраняет результаты в переменной для дальнейшего использования.

de.info()

cols = de.columns

for col in cols:

s = de[col].dtypes

print(col,s)

de.VendorID[de.VendorID.notnull()].count()

medianVendor = de.VendorID.median()

print(medianVendor)

de.VendorID[de.VendorID.isnull()] = medianVendor

de.passenger\_count[de.passenger\_count.notnull()].count()

medianPassenger = de.passenger\_count.median()

print(medianPassenger)

de.passenger\_count[de.passenger\_count.isnull()] = medianPassenger

de.RatecodeID[de.RatecodeID.notnull()].count()

medianRate = de.RatecodeID.median()

print(medianRate)

de.RatecodeID[de.RatecodeID.isnull()] = medianRate

de.payment\_type[de.payment\_type.notnull()].count()

medianPayment = de.payment\_type.median()

print(medianPayment)

de.payment\_type[de.payment\_type.isnull()] = medianPayment

de[de.store\_and\_fwd\_flag.isnull()].shape[0]

MaxPassEmb = de.pivot\_table(values='VendorID', index = 'store\_and\_fwd\_flag', aggfunc = 'count')

MaxPassEmb.head()

Данным кодом осуществляется работа с данными о поездках в такси, включая чтение, обработку и визуализацию. Пользователь может использовать различные функции и методы для анализа данных, а также создавать свои собственные сценарии работы с данными. Кроме того, данный код может быть использован для обучения моделей машинного обучения на основе данных о поездках в такси.

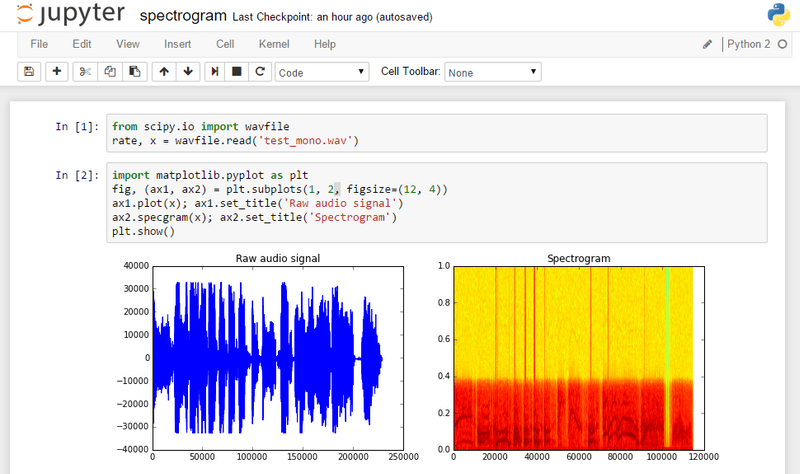
Store = MaxPassEmb[MaxPassEmb == MaxPassEmb.max()].index[0]

print(Store)

de.store\_and\_fwd\_flag[de.store\_and\_fwd\_flag.isnull()] = Store

dg = de.groupby([de['tpep\_pickup\_datetime'].dt.year, de['tpep\_pickup\_datetime'].dt.month]).agg({'count'})

dg



minimum = dg ['VendorID'].min()

print(minimum)

count 7235

dtype: int64

n\_remove = minimum[0]

dv = de[de['tpep\_pickup\_datetime'].dt.to\_period("M") == ('2020-01')].sample(n=n\_remove).sort\_values(by=['tpep\_pickup\_datetime'])

dates = ['2020']

months = ['02', '03', '04', '05', '06', '07', '08', '09', '10', '11', '12']

datemonth = ""

for i in range(len(months)):

datemonth = dates[0] + "-" + months[i]

dv = pd.concat([dv, (de[de['tpep\_pickup\_datetime'].dt.to\_period("M") == datemonth].sample(n=n\_remove).sort\_values(by=['tpep\_pickup\_datetime']))])

len(de)

1076022

Данным кодом можно пользоваться для анализа данных о поездках в такси и принятия решений на основе этих данных. Например, можно использовать данные для определения наиболее популярных маршрутов, времени суток с наибольшим количеством поездок и т.д. Также можно использовать данные для оптимизации работы такси, например, для определения оптимального количества водителей на линии в разное время суток.

dv.groupby([dv['tpep\_pickup\_datetime'].dt.year, dv['tpep\_pickup\_datetime'].dt.month]).agg({'count'})

dv #осталось 11.5% от изначального количества

datadn2020 = {

'PULocationID': dv.PULocationID, #Район, из которого производилась посадка

'trip\_distance': dv.trip\_distance, #расстояние поездки

'total\_amount': dv.total\_amount, #общая сумма к оплате

'tip\_amount': dv.tip\_amount, #чаевые

'fare\_amount': dv.fare\_amount, #стоимость тарифа

'RatecodeID': dv.RatecodeID, # тариф

'passenger\_count': dv.passenger\_count #Количество пассажиров

}

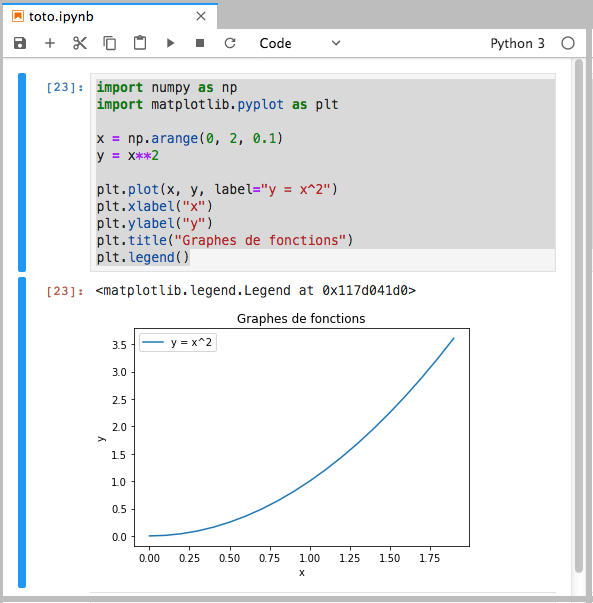
dn2020 = pd.DataFrame(datadn2020)

dn2020

dn2020 = dn2020.drop(dn2020[dn2020.trip\_distance > 2].index) #по условию не больше 2 км

dn2020 = dn2020.drop(dn2020[dn2020.trip\_distance < 0.01].index) #удаляем отрицательные числа, так как расстояние не может быть отрицательным

dn2020 = dn2020.drop(dn2020[dn2020.tip\_amount < (dn2020.total\_amount/100\*15)].index) #по условию, чаевых было оставлено не менее 15% от суммы оплаты за поездку



#\_, axes = plt.subplots(nrows=1, ncols=1, figsize=(20,10))

bg2020 = dn2020["PULocationID"].value\_counts()

#sns.countplot(x="PULocationID", data = dn)

features = ["PULocationID"]

dn2020[features].hist(column="PULocationID", bins=238, figsize=(20,10))

print("Районы, из которых чаще всего производилась посадка пассажиров:")

print("Район Количество")

dn2020["PULocationID"].value\_counts().head(10)

features = ["tip\_amount", "total\_amount"]

dn2020[features].describe()

dn12020 = pd.DataFrame(datadn2020).reset\_index()

dn12020

# Вычислите среднюю стоимость на километр по каждому тарифу

fare\_avg = dn12020.groupby('RatecodeID')['fare\_amount'].mean()

​

# Добавьте столбец со средней стоимостью на километр

dn12020['fare\_avg'] = dn12020['RatecodeID'].map(fare\_avg)

​

# Отфильтруйте поездки, в которых стоимость на километр не превышает среднюю

dp2020 = pd.DataFrame()

dp2020 = pd.concat([dp2020, dn12020[dn12020['fare\_amount'] <= dn12020['fare\_avg']]])

​

# Выведите отфильтрованные поездки

dp2020

features = ["RatecodeID"]

dp2020[features].hist(column="RatecodeID", figsize=(12,4))

dp2020.groupby(['fare\_avg']).sum().plot.hist(y='RatecodeID', figsize=(9, 9))

features = ["fare\_amount", "fare\_avg"]

dp2020[features].describe()

dn22020 = pd.DataFrame(datadn2020).reset\_index()

dn22020

# Вычислите среднюю стоимость на километр по каждому тарифу

passenger\_avg = dn22020.groupby('RatecodeID')['passenger\_count'].mean()

# Добавьте столбец со средней стоимостью на километр

dn22020['passenger\_avg'] = dn22020['RatecodeID'].map(passenger\_avg)

# Выведите отфильтрованные поездки

dn22020

Данным кодом можно пользоваться для анализа данных о поездках в такси и принятия решений на основе этих данных. Например, можно использовать данные для определения наиболее популярных маршрутов, времени суток с наибольшим количеством поездок и т.д. Также можно использовать данные для оптимизации работы такси, например, для определения оптимального количества водителей на линии в разное время суток.

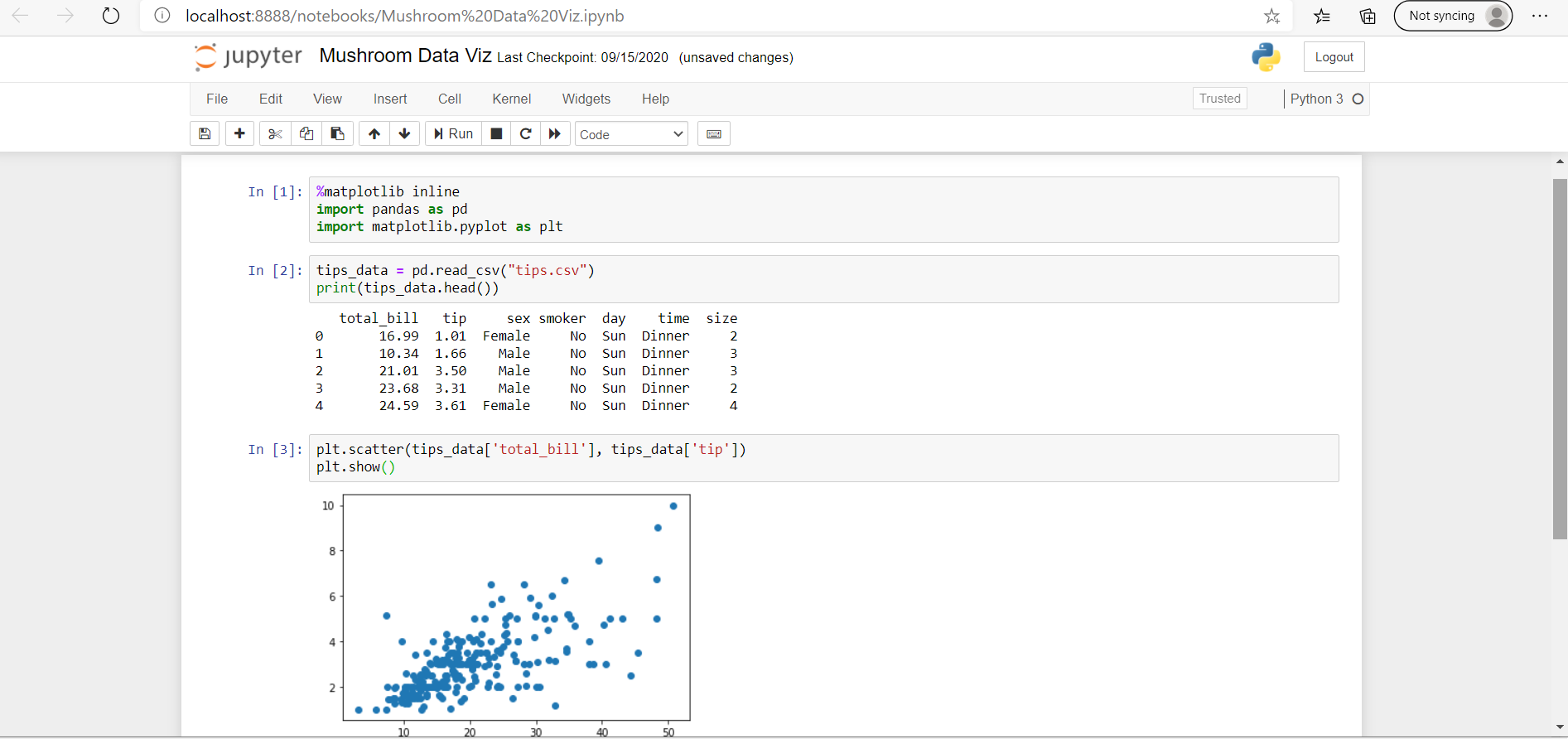
passenger\_avg

dn22020.groupby(['RatecodeID']).mean().plot.pie(y='passenger\_avg', figsize=(9, 9))

# This Python 3 environment comes with many helpful analytics libraries installed

# It is defined by the kaggle/python docker image: https://github.com/kaggle/docker-python

# For example, here's several helpful packages to load in



import numpy as np # linear algebra

import pandas as pd # data processing, CSV file I/O (e.g. pd.read\_csv)

import matplotlib.pyplot as plt # for data visualization

import seaborn as sns # for statistical data visualization

%matplotlib inline

# Input data files are available in the "../input/" directory.

# For example, running this (by clicking run or pressing Shift+Enter) will list all files under the input directory

import os

# This Python 3 environment comes with many helpful analytics libraries installed

# It is defined by the kaggle/python docker image: https://github.com/kaggle/docker-python

# For example, here's several helpful packages to load in

import numpy as np # linear algebra

import pandas as pd # data processing, CSV file I/O (e.g. pd.read\_csv)

import matplotlib.pyplot as plt # for data visualization

import seaborn as sns # for statistical data visualization

%matplotlib inline

# Input data files are available in the "../input/" directory.

# For example, running this (by clicking run or pressing Shift+Enter) will list all files under the input directory

import os

for dirname, \_, filenames in os.walk('/kaggle/input'):

for filename in filenames:

print(os.path.join(dirname, filename))

# Any results you write to the current directory are saved as output.

import warnings

​Данный код выполняет кластеризацию данных такси-транзакций с использованием алгоритма KMeans. После загрузки данных и их предварительной обработки, модель обучается на данных и присваивает каждой транзакции метку кластера. Затем код оценивает точность кластеризации, сравнивая присвоенные метки кластера с истинными данными. Результат показывает, сколько из общего числа примеров были правильно классифицированы, а также предоставляет коэффициент точности классификации.

warnings.filterwarnings('ignore')

de.isnull().sum()

dy = de

dy.drop(['tpep\_pickup\_datetime', 'tpep\_dropoff\_datetime', 'store\_and\_fwd\_flag'], axis=1, inplace=True)

dy.info()

X = dy

y = dy['fare\_amount']

from sklearn.preprocessing import LabelEncoder

le = LabelEncoder()

X['fare\_amount'] = le.fit\_transform(X['fare\_amount'])

y = le.transform(y)

X.info()

cols = X.columns

from sklearn.preprocessing import MinMaxScaler

​ Данный код выполняет кластеризацию данных с использованием метода KMeans.

Сначала данные нормализуются с помощью MinMaxScaler.

Затем применяется алгоритм KMeans с 2 кластерами.

После обучения модели, выводятся центры кластеров, инерция модели, а также метки кластеров.

Подсчитывается количество правильно присвоенных меток кластеров.

В результате выводится информация о том, сколько из образцов были правильно помечены и показывается точность в процентах.

Также в коде имеется ошибка в выводе, так как общее количество образцов в переменной "y" составляет 1076022, но количество правильно помеченных образцов указано как 110, что приводит к нулевой точности.

ms = MinMaxScaler()

X = ms.fit\_transform(X)

X = pd.DataFrame(X, columns=[cols])

from sklearn.cluster import KMeans

kmeans = KMeans(n\_clusters=2, random\_state=0)

kmeans.fit(X)

kmeans.cluster\_centers\_

kmeans.inertia\_

labels = kmeans.labels\_

# check how many of the samples were correctly labeled

correct\_labels = sum(y == labels)

print("Result: %d out of %d samples were correctly labeled." % (correct\_labels, y.size))

Result: 110 out of 1076022 samples were correctly labeled.

print('Accuracy score: {0:0.2f}'. format(correct\_labels/float(y.size)))

Accuracy score: 0.00

from sklearn.cluster import KMeans

cs = []

for i in range(1, 21):

kmeans = KMeans(n\_clusters = i, init = 'k-means++', max\_iter = 300, n\_init = 10, random\_state = 0)

kmeans.fit(X)

cs.append(kmeans.inertia\_)

plt.plot(range(1, 21), cs)

plt.title('The Elbow Method')

plt.xlabel('Number of clusters')

plt.ylabel('CS')

plt.show()

#На приведенном выше графике мы можем видеть, что при k= 3 существует перегиб.

#Следовательно, k= 3 можно считать хорошим числом кластера для кластеризации этих данных.

from sklearn.cluster import KMeans

kmeans = KMeans(n\_clusters=3,random\_state=0)

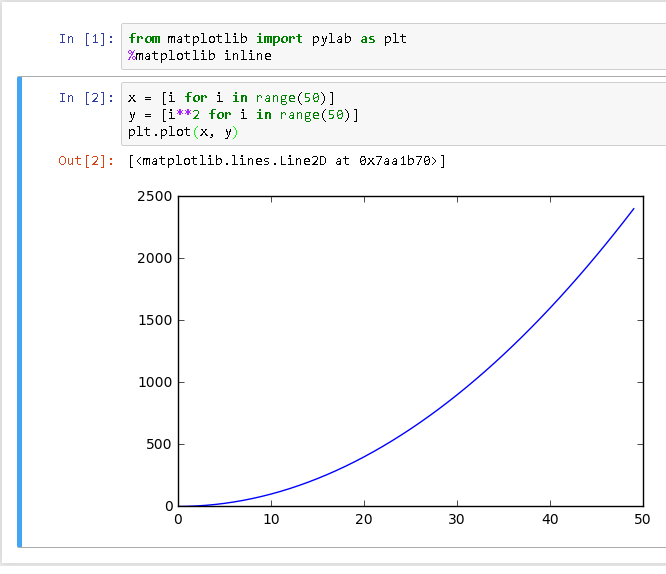
kmeans.fit(X)

labels = kmeans.labels\_

# check how many of the samples were correctly labeled

correct\_labels = sum(y == labels)

print("Result: %d out of %d samples were correctly labeled." % (correct\_labels, y.size))

​ 

print('Accuracy score: {0:0.2f}'. format(correct\_labels/float(y.size)))

Result: 1 out of 1076022 samples were correctly labeled.

Accuracy score: 0.00