create_datasets

September 29, 2021

1 Download and store data

This notebook contains information on downloading the Quandl Wiki stock prices and a few other sources that we use throughout the book.

1.1 Imports & Settings

[2]:
```
import warnings
warnings.filterwarnings('ignore')
```

[3]:
```
from pathlib import Path
import requests
from io import BytesIO
from zipfile import ZipFile, BadZipFile

import numpy as np
import pandas as pd
import pandas_datareader.data as web
from sklearn.datasets import fetch_openml

pd.set_option('display.expand_frame_repr', False)
```

1.2 Set Data Store path

Modify path if you would like to store the data elsewhere and change the notebooks accordingly

[93]:
```
DATA_STORE = Path('assets.h5')
```

1.3 Quandl Wiki Prices

Quandl makes available a dataset with stock prices, dividends and splits for 3000 US publicly-traded companies. Quandl decided to discontinue support in favor of its commercial offerings but the historical data are still useful to demonstrate the application of the machine learning solutions in the book, just ensure you implement your own algorithms on current data.

> As of April 11, 2018 this data feed is no longer actively supported by the Quandl community. We will continue to host this data feed on Quandl, but we do not recommend using it for investment or analysis.

1. Follow the instructions to create a free Quandl account
2. Download the entire WIKI/PRICES data
3. Extract the .zip file,
4. Move to this directory and rename to wiki_prices.csv
5. Run the below code to store in fast HDF format (see Chapter 02 on Market & Fundamental Data for details).

[8]:
```python
df = (pd.read_csv('wiki_prices.csv',
                 parse_dates=['date'],
                 index_col=['date', 'ticker'],
                 infer_datetime_format=True)
     .sort_index())

print(df.info(null_counts=True))
with pd.HDFStore(DATA_STORE) as store:
    store.put('quandl/wiki/prices', df)
```

```
<class 'pandas.core.frame.DataFrame'>
MultiIndex: 15389314 entries, (Timestamp('1962-01-02 00:00:00'), 'ARNC') to
(Timestamp('2018-03-27 00:00:00'), 'ZUMZ')
Data columns (total 12 columns):
 #   Column       Non-Null Count     Dtype
---  ------       --------------     -----
 0   open         15388776 non-null  float64
 1   high         15389259 non-null  float64
 2   low          15389259 non-null  float64
 3   close        15389313 non-null  float64
 4   volume       15389314 non-null  float64
 5   ex-dividend  15389314 non-null  float64
 6   split_ratio  15389313 non-null  float64
 7   adj_open     15388776 non-null  float64
 8   adj_high     15389259 non-null  float64
 9   adj_low      15389259 non-null  float64
 10  adj_close    15389313 non-null  float64
 11  adj_volume   15389314 non-null  float64
dtypes: float64(12)
memory usage: 1.4+ GB
None
```

1.3.1 Wiki Prices Metadata

As of writing, the following instructions no longer work because Quandl changed its API:

1. Follow the instructions to create a free Quandl account if you haven't done so yet
2. Find link to download wiki metadata under Companies](https://www.quandl.com/databases/WIKIP/documentation) or use the download link with your API_KEY: https://www.quandl.com/api/v3/databases/WIKI/metadata?api_key=
3. Extract the .zip file,

4. Move to this directory and rename to wiki_stocks.csv
5. Run the following code to store in fast HDF format

Instead, load the file `wiki_stocks.csv` as described and store in HDF5 format.

[5]:
```
df = pd.read_csv('wiki_stocks.csv')
# no longer needed
# df = pd.concat([df.loc[:, 'code'].str.strip(),
#                 df.loc[:, 'name'].str.split('(', expand=True)[0].str.strip().
 ↪to_frame('name')], axis=1)

print(df.info(null_counts=True))
with pd.HDFStore(DATA_STORE) as store:
    store.put('quandl/wiki/stocks', df)
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3199 entries, 0 to 3198
Data columns (total 2 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   code    3199 non-null   object
 1   name    3199 non-null   object
dtypes: object(2)
memory usage: 50.1+ KB
None
```

1.4 S&P 500 Prices

The following code downloads historical S&P 500 prices from FRED (only last 10 years of daily data is freely available)

[6]:
```
df = web.DataReader(name='SP500', data_source='fred', start=2009).squeeze().
 ↪to_frame('close')
print(df.info())
with pd.HDFStore(DATA_STORE) as store:
    store.put('sp500/fred', df)
```

```
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 2609 entries, 2010-06-16 to 2020-06-15
Data columns (total 1 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   close   2517 non-null   float64
dtypes: float64(1)
memory usage: 40.8 KB
None
```

Alternatively, download S&P500 data from stooq.com; at the time of writing the data was available since 1789. You can switch from Polish to English on the lower right-hand side.

We store the data from 1950-2020:

[9]:
```
sp500_stooq = (pd.read_csv('^spx_d.csv', index_col=0,
                     parse_dates=True).loc['1950':'2019'].rename(columns=str.
 ↪lower))
print(sp500_stooq.info())
```

```
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 17700 entries, 1950-01-03 to 2019-12-31
Data columns (total 5 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   open    17700 non-null  float64
 1   high    17700 non-null  float64
 2   low     17700 non-null  float64
 3   close   17700 non-null  float64
 4   volume  17700 non-null  float64
dtypes: float64(5)
memory usage: 829.7 KB
None
```

[10]:
```
with pd.HDFStore(DATA_STORE) as store:
    store.put('sp500/stooq', sp500_stooq)
```

1.4.1 S&P 500 Constituents

The following code downloads the current S&P 500 constituents from Wikipedia.

[13]:
```
url = 'https://en.wikipedia.org/wiki/List_of_S%26P_500_companies'
df = pd.read_html(url, header=0)[0]
```

[14]:
```
df.head()
```

[14]:
```
  Symbol             Security SEC filings             GICS Sector
GICS Sub Industry    Headquarters Location Date first added      CIK
Founded
0    MMM           3M Company     reports             Industrials
Industrial Conglomerates      St. Paul, Minnesota       1976-08-09    66740
1902
1    ABT  Abbott Laboratories     reports             Health Care
Health Care Equipment  North Chicago, Illinois       1964-03-31     1800
1888
2   ABBV          AbbVie Inc.     reports             Health Care
Pharmaceuticals  North Chicago, Illinois       2012-12-31  1551152  2013 (1888)
3   ABMD          ABIOMED Inc     reports             Health Care
Health Care Equipment   Danvers, Massachusetts       2018-05-31   815094
1981
4    ACN        Accenture plc     reports  Information Technology  IT Consulting
```

```
& Other Services          Dublin, Ireland       2011-07-06  1467373         1989
```

[15]:
```
df.columns = ['ticker', 'name', 'sec_filings', 'gics_sector',␣
↪'gics_sub_industry',
              'location', 'first_added', 'cik', 'founded']
df = df.drop('sec_filings', axis=1).set_index('ticker')
```

[16]:
```
print(df.info())
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 505 entries, MMM to ZTS
Data columns (total 7 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
 0   name               505 non-null    object
 1   gics_sector        505 non-null    object
 2   gics_sub_industry  505 non-null    object
 3   location           505 non-null    object
 4   first_added        408 non-null    object
 5   cik                505 non-null    int64
 6   founded            234 non-null    object
dtypes: int64(1), object(6)
memory usage: 31.6+ KB
None
```

[17]:
```
with pd.HDFStore(DATA_STORE) as store:
    store.put('sp500/stocks', df)
```

1.5 Metadata on US-traded companies

The following downloads several attributes for companies traded on NASDAQ, AMEX and NYSE

> Update: unfortunately, NASDAQ has disabled automatic downloads. However, you can still access and manually download the files at the below URL when you fill in the exchange names. So for AMEX, URL becomes `https://www.nasdaq.com/market-activity/stocks/screener?exchange=AMEX&letter=0&render=downl`

[12]:
```
# no longer works!
url = 'https://old.nasdaq.com/screening/companies-by-name.aspx?
↪letter=0&exchange={}&render=download'
exchanges = ['NASDAQ', 'AMEX', 'NYSE']
df = pd.concat([pd.read_csv(url.format(ex)) for ex in exchanges]).
↪dropna(how='all', axis=1)
df = df.rename(columns=str.lower).set_index('symbol').drop('summary quote',␣
↪axis=1)
df = df[~df.index.duplicated()]
print(df.info())
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 6988 entries, TXG to ZYME
Data columns (total 6 columns):
 #   Column     Non-Null Count  Dtype
---  ------     --------------  -----
 0   name       6988 non-null   object
 1   lastsale   6815 non-null   float64
 2   marketcap  5383 non-null   object
 3   ipoyear    3228 non-null   float64
 4   sector     5323 non-null   object
 5   industry   5323 non-null   object
dtypes: float64(2), object(4)
memory usage: 382.2+ KB
None
```

[13]:
```
df.head()
```

[13]:
```
                                          name  lastsale marketcap  ipoyear
sector                                                industry
symbol
TXG                         10x Genomics, Inc.   88.4200     $8.7B   2019.0
Capital Goods  Biotechnology: Laboratory Analytical Instruments
YI                                   111, Inc.    6.6200  $545.22M   2018.0
Health Care                          Medical/Nursing Services
PIH     1347 Property Insurance Holdings, Inc.    4.5443   $27.58M   2014.0
Finance                        Property-Casualty Insurers
PIHPP   1347 Property Insurance Holdings, Inc.   25.4202       NaN      NaN
Finance                        Property-Casualty Insurers
TURN                      180 Degree Capital Corp.    1.8300   $56.95M      NaN
Finance                        Finance/Investors Services
```

1.5.1 Convert market cap information to numerical format

Market cap is provided as strings so we need to convert it to numerical format.

[14]:
```
mcap = df[['marketcap']].dropna()
mcap['suffix'] = mcap.marketcap.str[-1]
mcap.suffix.value_counts()
```

[14]:
```
M    3148
B    2235
Name: suffix, dtype: int64
```

Keep only values with value units:

[15]:
```
mcap = mcap[mcap.suffix.str.endswith(('B', 'M'))]
mcap.marketcap = pd.to_numeric(mcap.marketcap.str[1:-1])
mcaps = {'M': 1e6, 'B': 1e9}
```

```
for symbol, factor in mcaps.items():
    mcap.loc[mcap.suffix == symbol, 'marketcap'] *= factor
mcap.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 5383 entries, TXG to ZYME
Data columns (total 2 columns):
 #   Column     Non-Null Count  Dtype
---  ------     --------------  -----
 0   marketcap  5383 non-null   float64
 1   suffix     5383 non-null   object
dtypes: float64(1), object(1)
memory usage: 286.2+ KB
```

[16]:
```
df['marketcap'] = mcap.marketcap
df.marketcap.describe(percentiles=np.arange(.1, 1, .1).round(1)).apply(lambda x:
 ↪ f'{int(x):,d}')
```

[16]:
```
count                5,383
mean         8,058,312,556
std         46,063,490,648
min              1,680,000
10%             41,436,000
20%            104,184,000
30%            192,888,000
40%            335,156,000
50%            587,760,000
60%          1,120,000,000
70%          2,140,000,000
80%          4,480,000,000
90%         13,602,000,000
max      1,486,630,000,000
Name: marketcap, dtype: object
```

1.5.2 Store result

The file `us_equities_meta_data.csv` contains a version of the data used for many of the examples. Load using

```
df = pd.read_csv('us_equities_meta_data.csv')
```

and proceed to store in HDF5 format.

[5]:
```
df = pd.read_csv('us_equities_meta_data.csv')
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 6834 entries, 0 to 6833
Data columns (total 7 columns):
```

```
 #   Column     Non-Null Count  Dtype
---  ------     --------------  -----
 0   ticker     6834 non-null   object
 1   name       6834 non-null   object
 2   lastsale   6718 non-null   float64
 3   marketcap  5766 non-null   float64
 4   ipoyear    3038 non-null   float64
 5   sector     5288 non-null   object
 6   industry   5288 non-null   object
dtypes: float64(3), object(4)
memory usage: 373.9+ KB
```

[7]:
```
with pd.HDFStore(DATA_STORE) as store:
    store.put('us_equities/stocks', df.set_index('ticker'))
```

1.6 MNIST Data

[36]:
```
mnist = fetch_openml('mnist_784', version=1)
```

[37]:
```
print(mnist.DESCR)
```

```
**Author**: Yann LeCun, Corinna Cortes, Christopher J.C. Burges
**Source**: [MNIST Website](http://yann.lecun.com/exdb/mnist/) - Date unknown
**Please cite**:

The MNIST database of handwritten digits with 784 features, raw data available
at: http://yann.lecun.com/exdb/mnist/. It can be split in a training set of the
first 60,000 examples, and a test set of 10,000 examples

It is a subset of a larger set available from NIST. The digits have been size-
normalized and centered in a fixed-size image. It is a good database for people
who want to try learning techniques and pattern recognition methods on real-
world data while spending minimal efforts on preprocessing and formatting. The
original black and white (bilevel) images from NIST were size normalized to fit
in a 20x20 pixel box while preserving their aspect ratio. The resulting images
contain grey levels as a result of the anti-aliasing technique used by the
normalization algorithm. the images were centered in a 28x28 image by computing
the center of mass of the pixels, and translating the image so as to position
this point at the center of the 28x28 field.

With some classification methods (particularly template-based methods, such as
SVM and K-nearest neighbors), the error rate improves when the digits are
centered by bounding box rather than center of mass. If you do this kind of pre-
processing, you should report it in your publications. The MNIST database was
constructed from NIST's NIST originally designated SD-3 as their training set
and SD-1 as their test set. However, SD-3 is much cleaner and easier to
recognize than SD-1. The reason for this can be found on the fact that SD-3 was
```

```
collected among Census Bureau employees, while SD-1 was collected among high-
school students. Drawing sensible conclusions from learning experiments requires
that the result be independent of the choice of training set and test among the
complete set of samples. Therefore it was necessary to build a new database by
mixing NIST's datasets.

The MNIST training set is composed of 30,000 patterns from SD-3 and 30,000
patterns from SD-1. Our test set was composed of 5,000 patterns from SD-3 and
5,000 patterns from SD-1. The 60,000 pattern training set contained examples
from approximately 250 writers. We made sure that the sets of writers of the
training set and test set were disjoint. SD-1 contains 58,527 digit images
written by 500 different writers. In contrast to SD-3, where blocks of data from
each writer appeared in sequence, the data in SD-1 is scrambled. Writer
identities for SD-1 is available and we used this information to unscramble the
writers. We then split SD-1 in two: characters written by the first 250 writers
went into our new training set. The remaining 250 writers were placed in our
test set. Thus we had two sets with nearly 30,000 examples each. The new
training set was completed with enough examples from SD-3, starting at pattern #
0, to make a full set of 60,000 training patterns. Similarly, the new test set
was completed with SD-3 examples starting at pattern # 35,000 to make a full set
with 60,000 test patterns. Only a subset of 10,000 test images (5,000 from SD-1
and 5,000 from SD-3) is available on this site. The full 60,000 sample training
set is available.

Downloaded from openml.org.
```

[38]:
```python
mnist.keys()
```

[38]:
```
dict_keys(['data', 'target', 'frame', 'categories', 'feature_names',
'target_names', 'DESCR', 'details', 'url'])
```

[39]:
```python
mnist_path = Path('mnist')
if not mnist_path.exists():
    mnist_path.mkdir()
```

[40]:
```python
np.save(mnist_path / 'data', mnist.data.astype(np.uint8))
np.save(mnist_path / 'labels', mnist.target.astype(np.uint8))
```

1.7 Fashion MNIST Image Data

We will use the Fashion MNIST image data created by Zalando Research for some demonstrations.

[12]:
```python
fashion_mnist = fetch_openml(name='Fashion-MNIST')
```

[13]:
```python
print(fashion_mnist.DESCR)
```

```
**Author**: Han Xiao, Kashif Rasul, Roland Vollgraf
**Source**: [Zalando Research](https://github.com/zalandoresearch/fashion-mnist)
```

```
**Please cite**: Han Xiao and Kashif Rasul and Roland Vollgraf, Fashion-MNIST: a
Novel Image Dataset for Benchmarking Machine Learning Algorithms, arXiv,
cs.LG/1708.07747

Fashion-MNIST is a dataset of Zalando's article images, consisting of a training
set of 60,000 examples and a test set of 10,000 examples. Each example is a
28x28 grayscale image, associated with a label from 10 classes. Fashion-MNIST is
intended to serve as a direct drop-in replacement for the original MNIST dataset
for benchmarking machine learning algorithms. It shares the same image size and
structure of training and testing splits.

Raw data available at: https://github.com/zalandoresearch/fashion-mnist

### Target classes
Each training and test example is assigned to one of the following labels:
Label  Description
0  T-shirt/top
1  Trouser
2  Pullover
3  Dress
4  Coat
5  Sandal
6  Shirt
7  Sneaker
8  Bag
9  Ankle boot

Downloaded from openml.org.
```

[33]:
```python
label_dict = {0: 'T-shirt/top',
              1: 'Trouser',
              2: 'Pullover',
              3: 'Dress',
              4: 'Coat',
              5: 'Sandal',
              6: 'Shirt',
              7: 'Sneaker',
              8: 'Bag',
              9: 'Ankle boot'}
```

[34]:
```python
fashion_path = Path('fashion_mnist')
if not fashion_path.exists():
    fashion_path.mkdir()
```

[35]:
```python
pd.Series(label_dict).to_csv(fashion_path / 'label_dict.csv', index=False,␣
 ↪header=None)
```

[31]:
```
np.save(fashion_path / 'data', fashion_mnist.data.astype(np.uint8))
np.save(fashion_path / 'labels', fashion_mnist.target.astype(np.uint8))
```

1.8 Bond Price Indexes

The following code downloads several bond indexes from the Federal Reserve Economic Data service (FRED)

[27]:
```
securities = {'BAMLCC0A0CMTRIV'   : 'US Corp Master TRI',
              'BAMLHYH0A0HYM2TRIV': 'US High Yield TRI',
              'BAMLEMCBPITRIV'    : 'Emerging Markets Corporate Plus TRI',
              'GOLDAMGBD228NLBM'  : 'Gold (London, USD)',
              'DGS10'             : '10-Year Treasury CMR',
              }

df = web.DataReader(name=list(securities.keys()), data_source='fred',␣
 ↪start=2000)
df = df.rename(columns=securities).dropna(how='all').resample('B').mean()

with pd.HDFStore(DATA_STORE) as store:
    store.put('fred/assets', df)
```