Appendix A: Time Series Examples

Contents

Introduction

This appendix provides the code necessary to download time series used in the lecture notes.

Imports

```
import re
import os
import zipfile
import requests
import numpy as np
import pandas as pd
import cryptocompare
from datetime import datetime
from urllib.request import urlretrieve
import statsmodels.api as sm
from datetime import datetime
import matplotlib.pyplot as plt
plt.style.use('ggplot')
plt.rcParams.update({'font.size': 16, 'axes.labelweight': 'bold', 'figure.figs
```

```
---------------------------------------------------------------------------
ModuleNotFoundError                       Traceback (most recent call last)
Cell In[1], line 7
      5 import numpy as np
      6 import pandas as pd
----> 7 import cryptocompare
      8 from datetime import datetime
      9 from urllib.request import urlretrieve

ModuleNotFoundError: No module named 'cryptocompare'
```

1. Beach width

Description

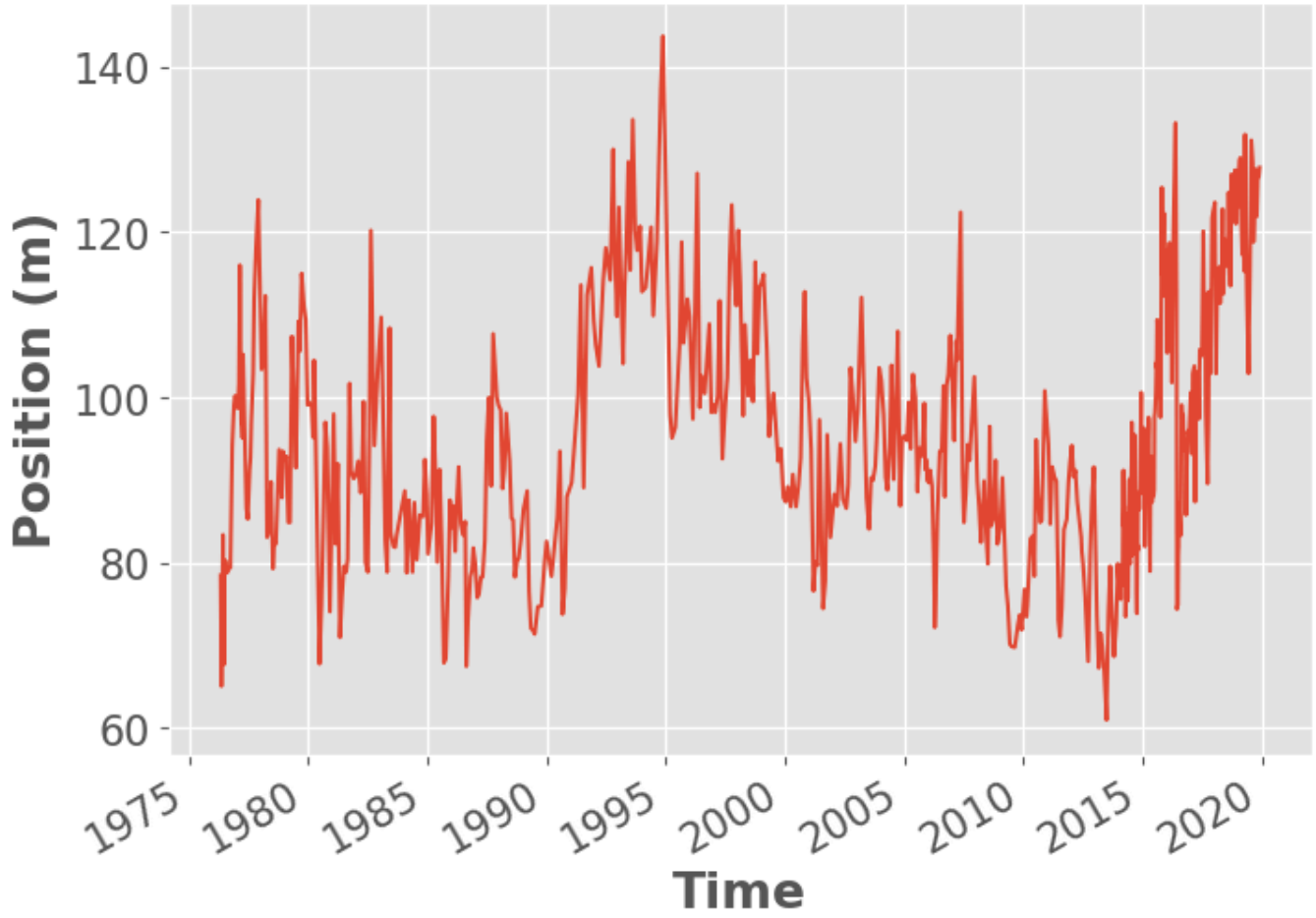
A beach survey program of one of the local beaches in Sydney, Australia, called Narrabeen Beach. The survey program started in the 1970's and has continued to the present day. All the data is available at: http://narrabeen.wrl.unsw.edu.au/explore_data/time_series/. The survey program aimed to measure the width of the beach every few weeks. There were five locations

that we would measure along the beach, from location 1 at the north end of the beach, to location 5 at the south end. You can check out the website for more information.

Method

The survey data is available on the above website, but there isn't an easy-to-use API available to query the data. Instead we are going to use the `requests` library to interact with the website and parse the response with regex (the `re` library).

```
def get_beach_data(beach_profile):
    """Scrape beach data from http://narrabeen.wrl.unsw.edu.au"""
    if beach_profile not in range(1, 6):
        raise ValueError("beach_profile should be an integer 1, 2, 3, 4, or 5.
    url = "http://narrabeen.wrl.unsw.edu.au/explore_data/time_series/"
    with requests.session() as s:
        cookies = dict(s.get(url).cookies.items())
        data = {
            "csrfmiddlewaretoken": cookies["csrftoken"],
            "profile": str(beach_profile),
            "startDate": "27/04/1976",
            "datatype": "WIDTH",
        }
        response = s.post(url, cookies=cookies, data=data, verify=False)
        return re.findall(r'new Date\("(.*)"\)\.getTime\(\), (\d+.\d)', respon

PROFILE = 1
time, width = zip(*get_beach_data(PROFILE))
df = (pd.DataFrame(width, index=pd.to_datetime(time), dtype=float)
        .rename(columns={0: "Shoreline"})
        .rename_axis(index="Time")
     )
# For some reason, this value is missing from the series
missing_value = pd.DataFrame({"Shoreline": [122.5]},
                             index=[pd.Timestamp(year=2017, month=12, day=15)]
df = pd.concat([df, missing_value]).sort_index()
df.to_csv("data/beach.csv")
df[:"2016"].to_csv("data/beach_train.csv")
df["2017":].to_csv("data/beach_valid.csv")
df.plot.line(xlabel="Time", ylabel="Position (m)", legend=False);
```



2. Cryptocurrencies

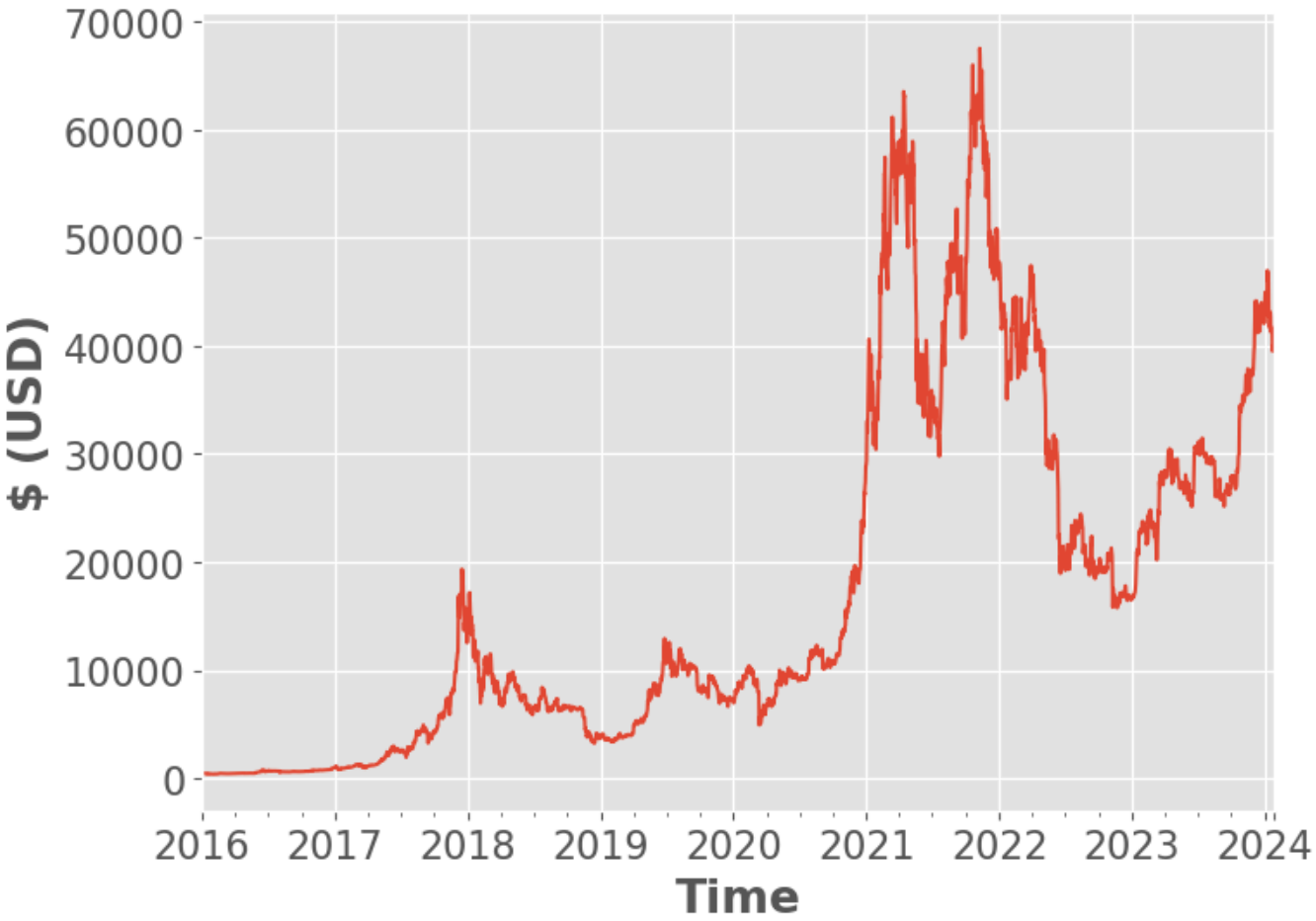
Description

Cryptocurrency is a digital currency that you may have heard a bit about in the news recently. You can read more about it here. We'll be extracting historical prices for two cryptocurrencies here: Bitcoin and Dogecoin.

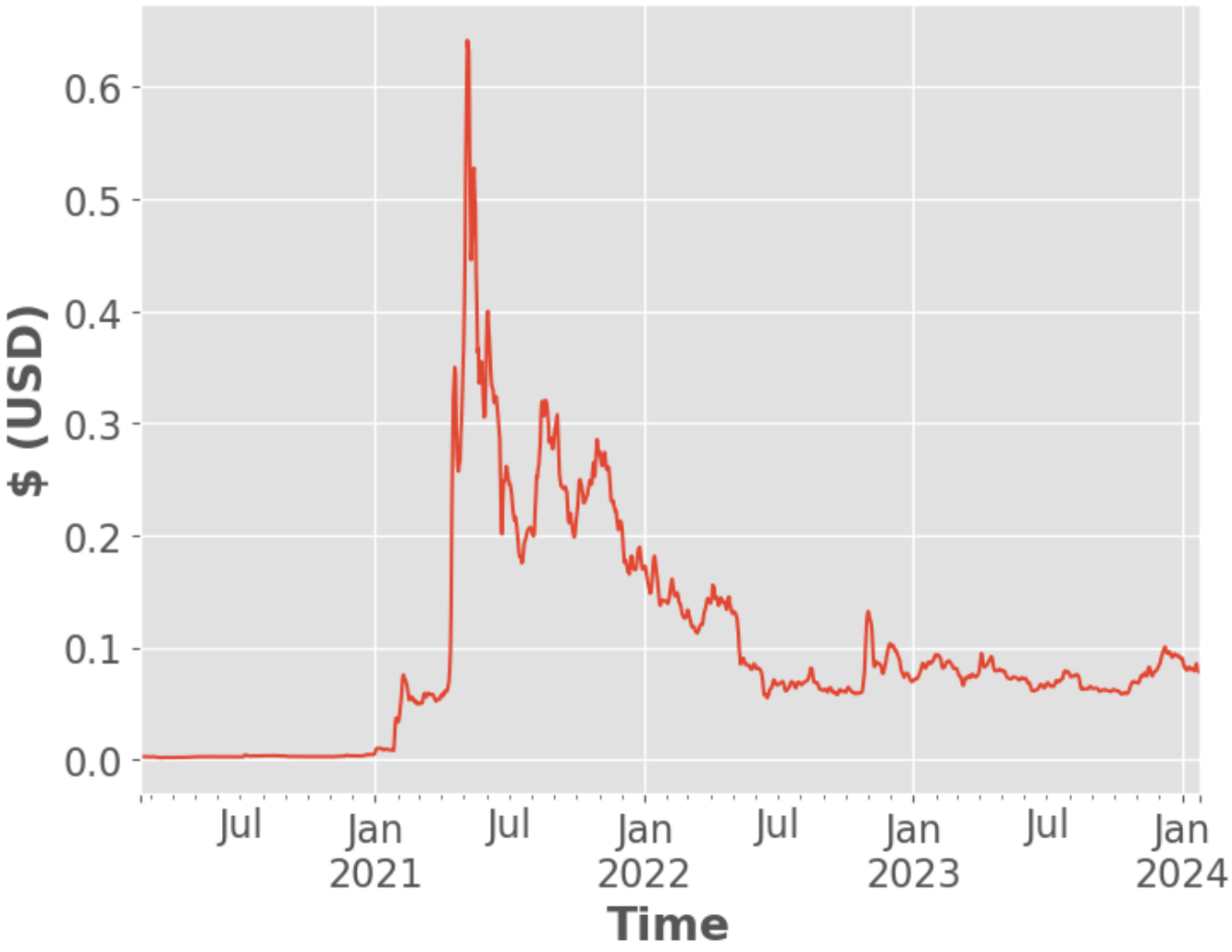
Method

I'll be using the cryptocompare package to access cryptocurrency data from CryptoCompare.com. We can only access 2000 records at a time, so I'll use a simple loop to get full historical records of daily price.

```
# Bitcoin
times = []
prices = []
timestamp = datetime.now()

for i in range(3):  # 3 is enough to get historical prices for now, a bit lazy
    records = cryptocompare.get_historical_price_day("BTC", currency="USD", to
    time = [r["time"] for r in records]
    price = [r["close"] for r in records]
    times += time
    prices += price
    timestamp = datetime.fromtimestamp(min(times))

df = (pd.DataFrame(prices, index=pd.to_datetime(times, unit="s", origin="unix"
        .rename(columns={0:"Price"})
        .sort_index()["2016":]
        .rename_axis(index="Time")
        .resample("1D").mean()
     )
# df.to_csv("data/bitcoin.csv")
df.plot.line(xlabel="Time", ylabel="$ (USD)", legend=False);
```



```
# Dogecoin
times = []
prices = []
timestamp = datetime.now()

for i in range(1):
    records = cryptocompare.get_historical_price_day("DOGE", currency="USD", t
    time = [r["time"] for r in records]
    price = [r["close"] for r in records]
    times += time
    prices += price
    timestamp = datetime.fromtimestamp(min(times))

df = (pd.DataFrame(prices, index=pd.to_datetime(times, unit="s", origin="unix"
        .rename(columns={0:"Price"})
        .rolling(3, center=True).mean()  # moving average to smooth things out
        .sort_index()["2020":]
        .rename_axis(index="Time")
        .resample("1D").mean()
        .dropna()
     )
# df.to_csv("data/dogecoin.csv")
df.plot.line(xlabel="Time", ylabel="$ (USD)", legend=False);
```



3. Global CO2 concentration

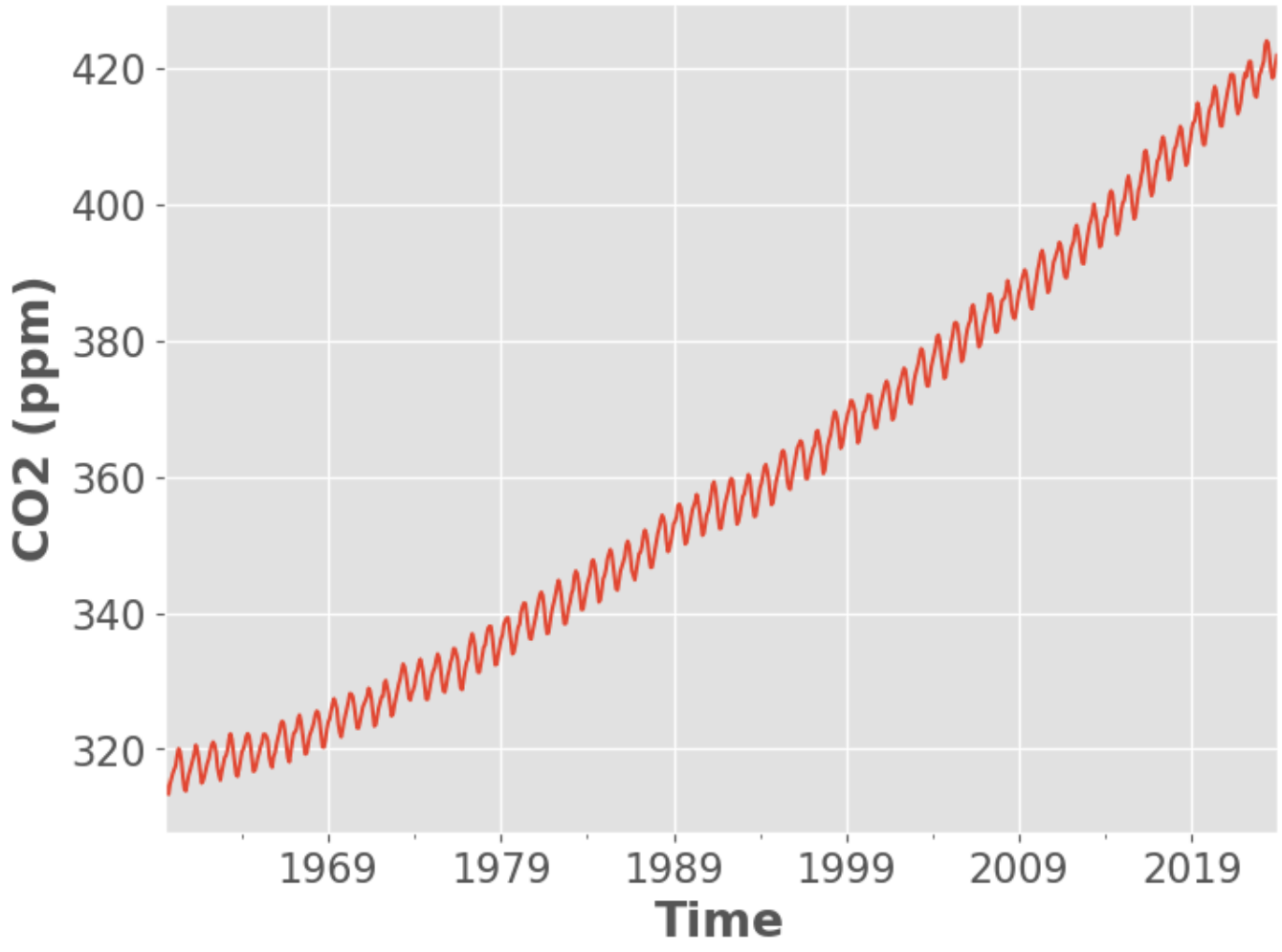
Description

Measurements of atmospheric CO2 have been taken at Mauna Loa Observatory, Hawaii since 1958. You can read more about the observatory and the data collection program here.

Method

Fortunately, the data is readily available online at the linked url above and we can read it in directly with Pandas.

```
url = "https://www.esrl.noaa.gov/gmd/webdata/ccgg/trends/co2/co2_mm_mlo.csv"
df = pd.read_csv(url, usecols=[0, 1, 3], names=["Year", "Month", "CO2"], heade
                 parse_dates={"Time" : [0, 1]}, skiprows=57)
df = df.resample("1M").mean()
# df.to_csv("data/co2.csv")
df.plot.line(xlabel="Time", ylabel="CO2 (ppm)", legend=False);
```



4. Global Temperature Anomalies

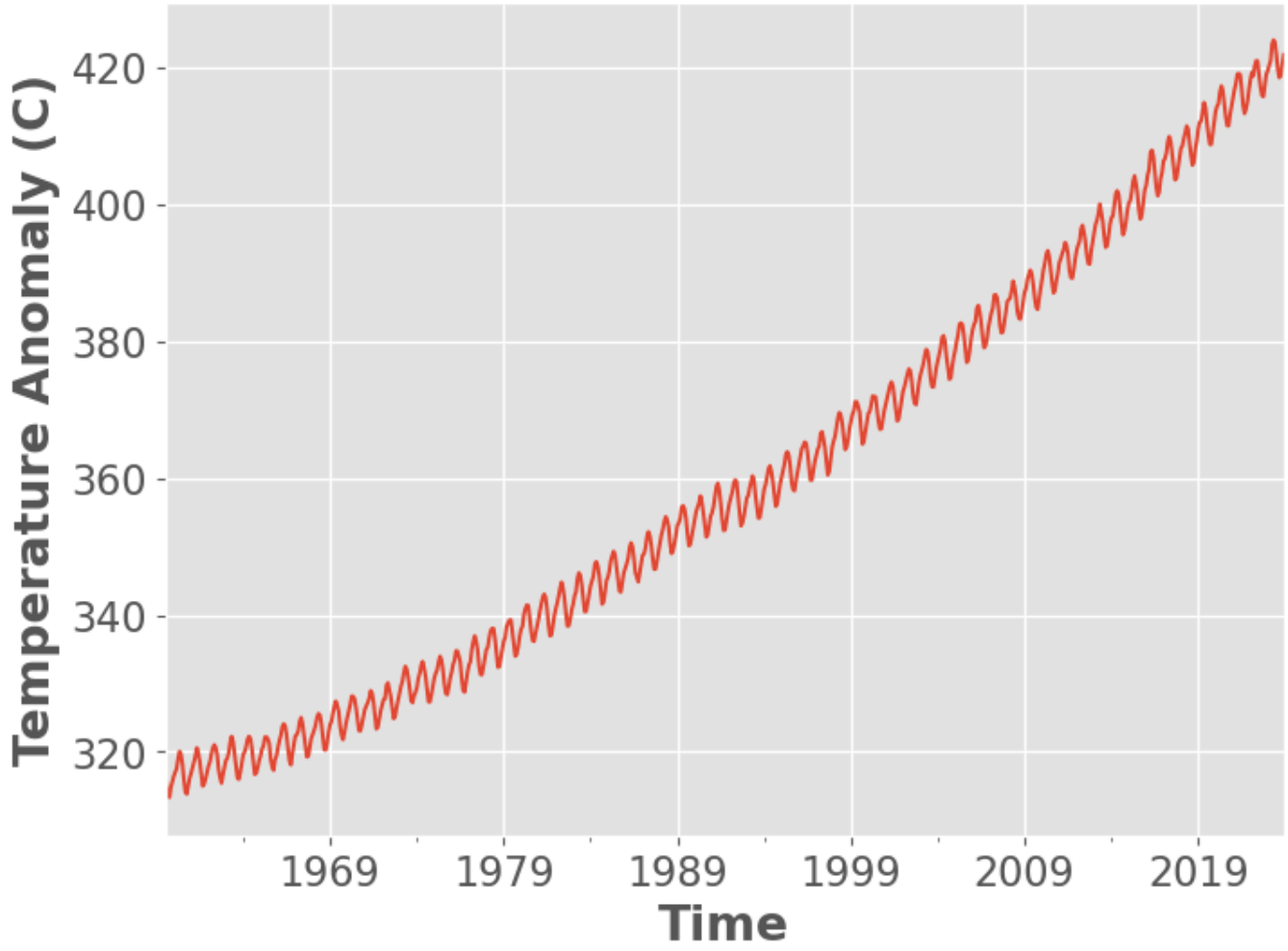
Description

Measurements of global temperature anomalies are available here, and you can read more about the dataset at that link.

Method

Fortunately, the data is readily available online at the linked url above and we can read it in directly with Pandas.

```
# url = "https://data.giss.nasa.gov/gistemp/graphs/graph_data/Global_Mean_Esti
# df = (pd.read_csv(url, delimiter=r"\s+", header=None, skiprows=5,
#                   usecols=[0, 1], names=["Year", "Anomaly"], index_col=0, par
#         .resample("1Y").mean()
#      )

# df.to_csv("data/temperature.csv")

df = pd.read_csv("data/temperature.csv", index_col=0, parse_dates=True)

df.plot.line(xlabel="Time", ylabel="Temperature Anomaly (C)", legend=False);
```



5. Sunspots

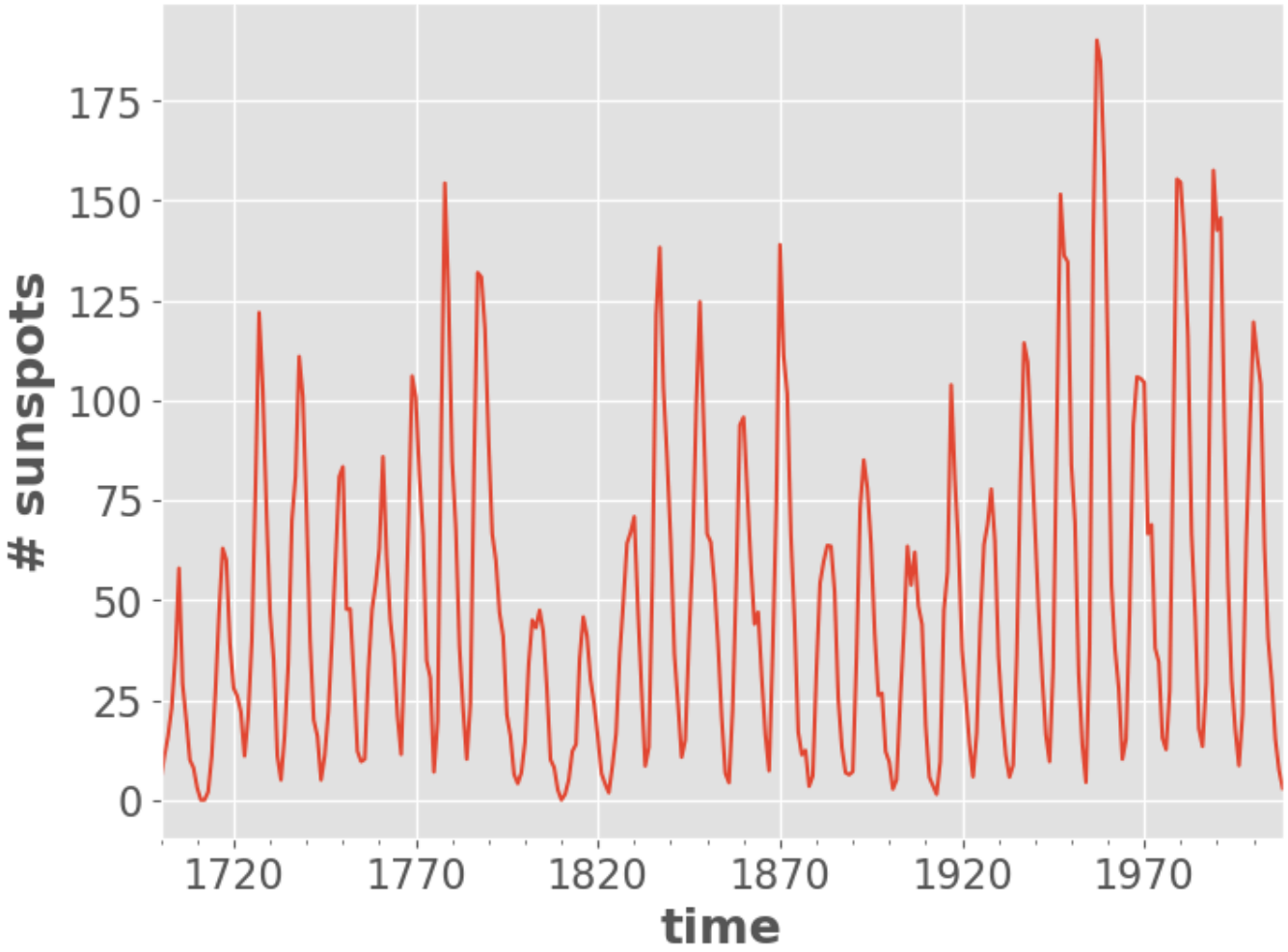
Description

Sunspots are areas of strong magnetic fields on the surface of the sun. These magnetic fields cause the spot to be cooler than it's surroundings, leading to a "dark patch" on the surface of the sun. Sunspots increase and decrease over an average cycle of 11 years. You can read more about them here.

Method

Annual counts of sunspots spanning 1700-2008 data are available through the `statsmodels` package.

```
df = (sm.datasets
        .sunspots
        .load_pandas()
        .data
        .rename(columns={"YEAR":"year", "SUNACTIVITY":"sunspots"})
        .set_index("year")
     )
df.index = pd.to_datetime(df.index.astype(int), format="%Y")
df = df.resample("1Y").mean()
df.to_csv("data/sunspots.csv")
df.plot.line(xlabel="time", ylabel="# sunspots", legend=False);
```



6. Southern Oscillation Index

Description

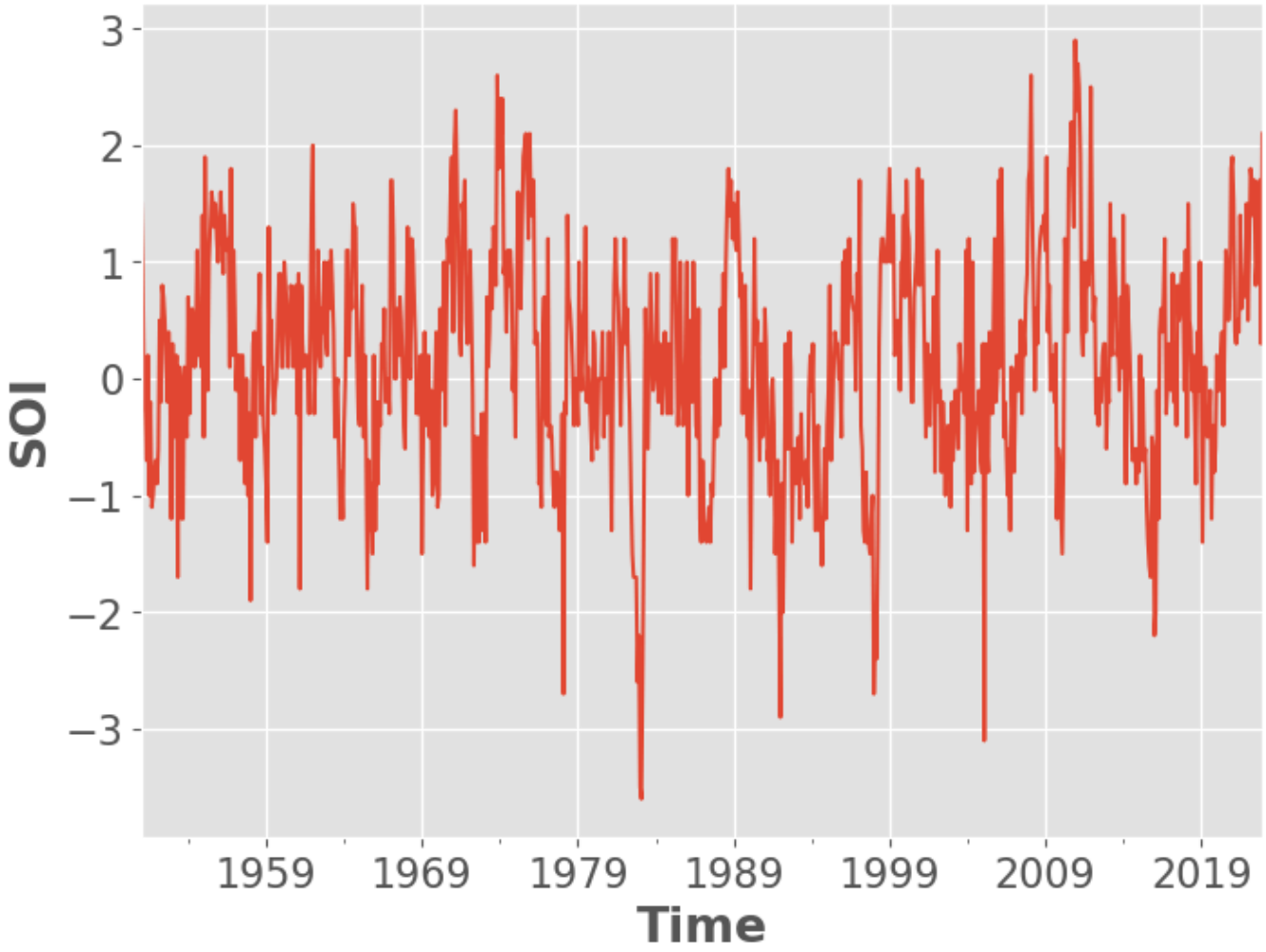
The Southern Oscillation Index (SOI) is a standardized index based on the observed sea level pressure differences between Tahiti and Darwin, Australia. The SOI is one measure of the

large-scale fluctuations in air pressure occurring between the western and eastern tropical Pacific (i.e., the state of the Southern Oscillation) during El Niño and La Niña episodes. In general, smoothed time series of the SOI correspond very well with changes in ocean temperatures across the eastern tropical Pacific. You can read more about the SOI here.

Method

Fortunately, the data is readily available online at the linked url above and we can read it in directly with Pandas.

```
url = "https://www.ncdc.noaa.gov/teleconnections/enso/indicators/soi/data.csv"
date_parser = lambda x: pd.to_datetime(x, format='%Y%m', errors='coerce')
df = pd.read_csv(url, sep='  ', skiprows=3, skipfooter=8, date_parser=date_par
df = df.tail(72) # keep standardized data only
df = pd.melt(df, id_vars=['YEAR'], var_name="MONTH", value_name="SOI") #pivot
df.index = pd.to_datetime(df['YEAR'].astype(str)  + df['MONTH'].astype(str), i
df.drop(columns=['YEAR','MONTH'],inplace=True)
df = df.resample("1M").mean().rename(columns={"Value": "SOI"})

# df.to_csv("data/soi.csv")
df.plot.line(xlabel="Time", ylabel="SOI", legend=False);
```



7. Airline Passengers

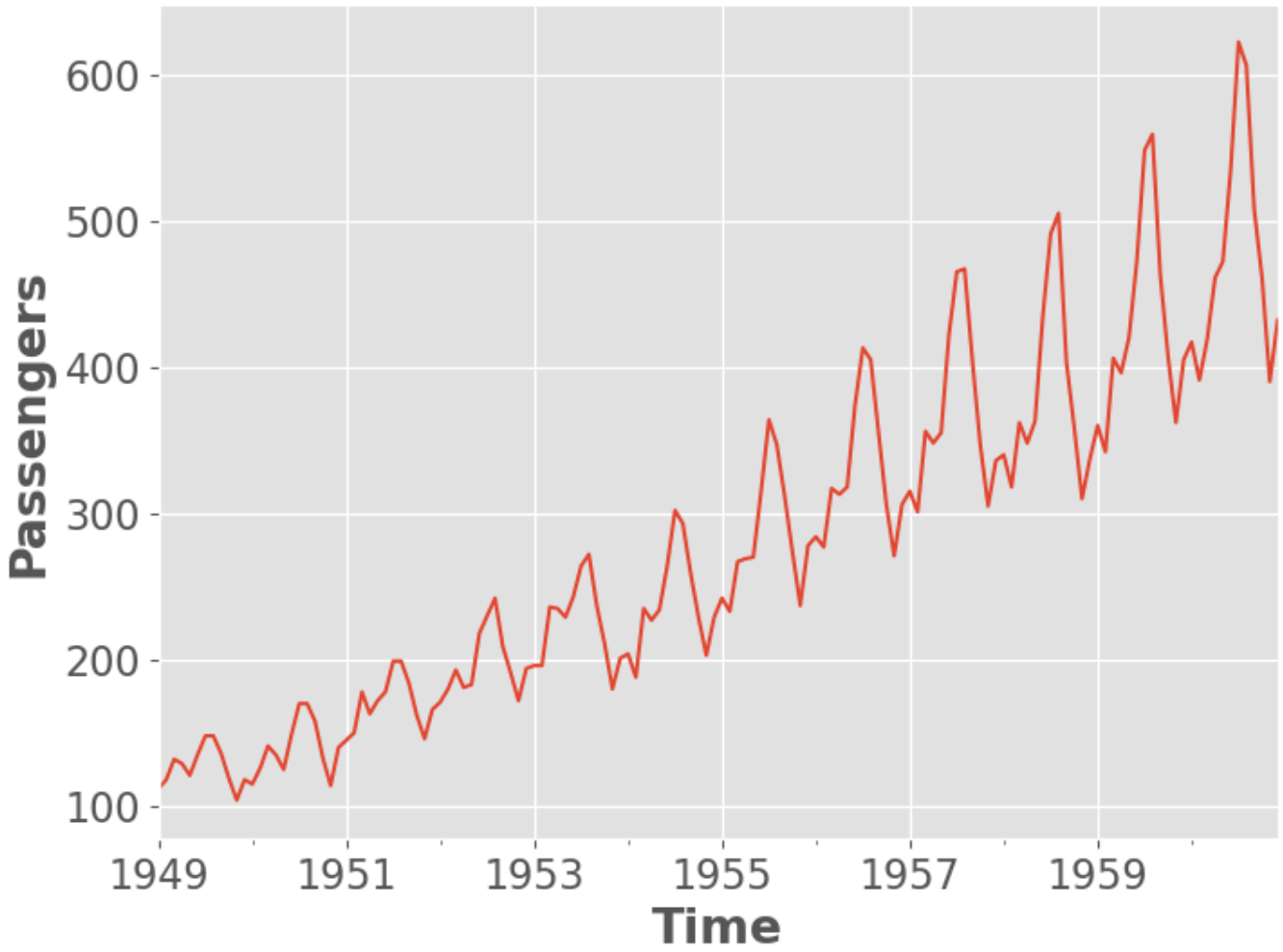
Description

This dataset describes the total number of monthly airline passengers over the period 1949-1960 (as air travel was becoming significantly more accessible and mainstream). The data is a classic example used when teaching time series. It's available all over the internet, but we'll be using a version that Jason Brownlee of Machine Learning Mastery has kindly made available here in csv format.

Method

Data is readily available online at the above link and we can read it in directly with Pandas.

```
url = "https://raw.githubusercontent.com/jbrownlee/Datasets/master/airline-pas
df = (pd.read_csv(url, index_col=0, parse_dates=True)
        .resample("1M").mean()
        .rename(columns={"Month": "Time"})
     )
df.to_csv("data/airline.csv")
df.plot.line(xlabel="Time", ylabel="Passengers", legend=False);
```



8. Canadian Beer Production

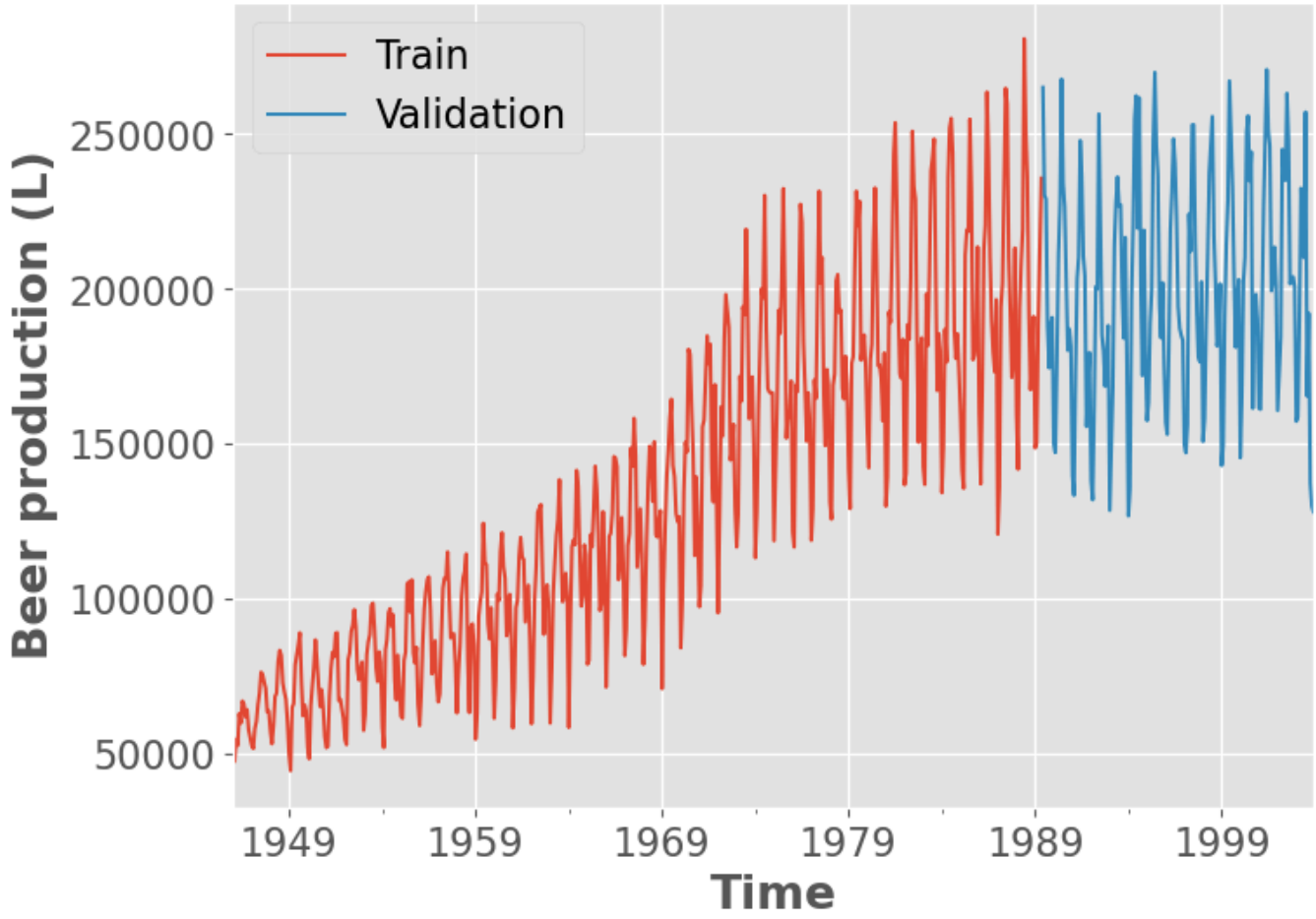
Description

This dataset describes the total monthly beer production in Canada over the period 1949-1960. It's available on the Statistics Canada website here.

Method

I accessed the data via that Statistics Canada API, documented here.

```
# Download
PID = "16100091"  # data identifier
url = f"https://www150.statcan.gc.ca/t1/wds/rest/getFullTableDownloadCSV/{PID}
r = requests.get(url)
urlretrieve(r.json()["object"], "data/beer.zip")
# Unzip
with zipfile.ZipFile("data/beer.zip", "r") as f:
    f.extractall("data/")
# Wrangle
df = (
    pd.read_csv(
        f"data/{PID}.csv",
        index_col=0,
        usecols=[0, 3, 10],
        header=None,
        names=["Time", "Type", "Production"],
        skiprows=1,
        parse_dates=True,
    )
    .query("Type == 'Beer production'")
    .drop(columns="Type")
    .resample("1M")
    .mean()
    .loc[:"2003"]
    .rename_axis(index="Time")
)
train_split = "1989-05"
valid_split = "1989-06"
df_train = df.loc[:train_split]
df_train.to_csv("data/beer_train.csv")
df_valid = df.loc[valid_split:]
df_valid.to_csv("data/beer_valid.csv")
# Delete files
os.remove(f"data/{PID}.csv")
os.remove(f"data/{PID}_MetaData.csv")
os.remove(f"data/beer.zip")
# Plot
(pd.concat((df_train, df_valid), axis=1)
   .set_axis(["Train", "Validation"], axis=1)
   .plot
   .line(xlabel="Time", ylabel="Beer production (L)")
);
```



9. White noise

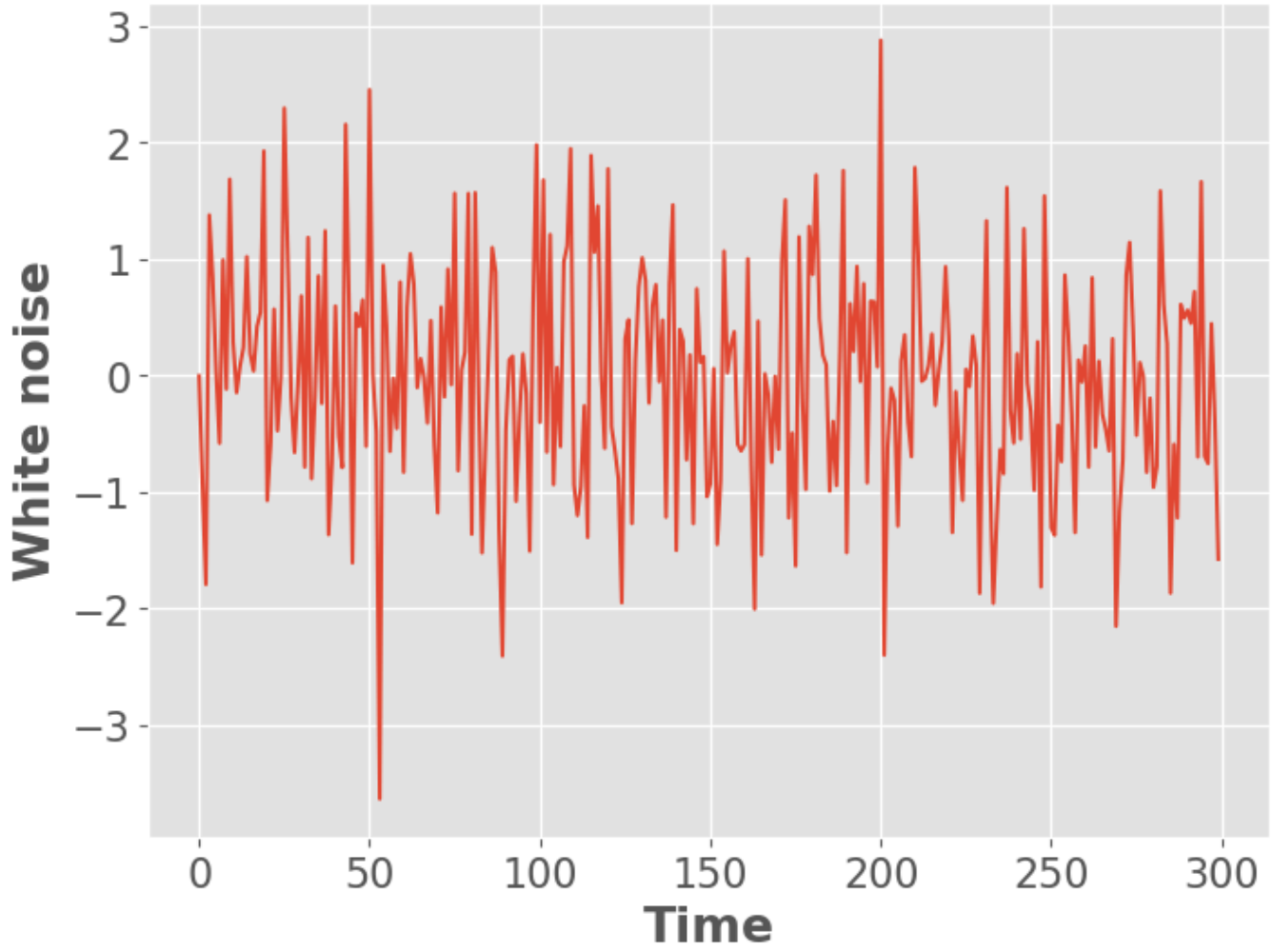
Description

Here I'll just be generating some simple i.i.d Gaussian white noise. Read more about white noise in Lecture 1.

Method

I'll be generating a white noise series by drawing from a standard normal distribution using `numpy`.

```
n = 300
df = pd.DataFrame(np.random.randn(n),
                  columns=["White noise"])
df.plot.line(xlabel="Time", ylabel="White noise", legend=False);
df.to_csv("data/white-noise.csv")
```



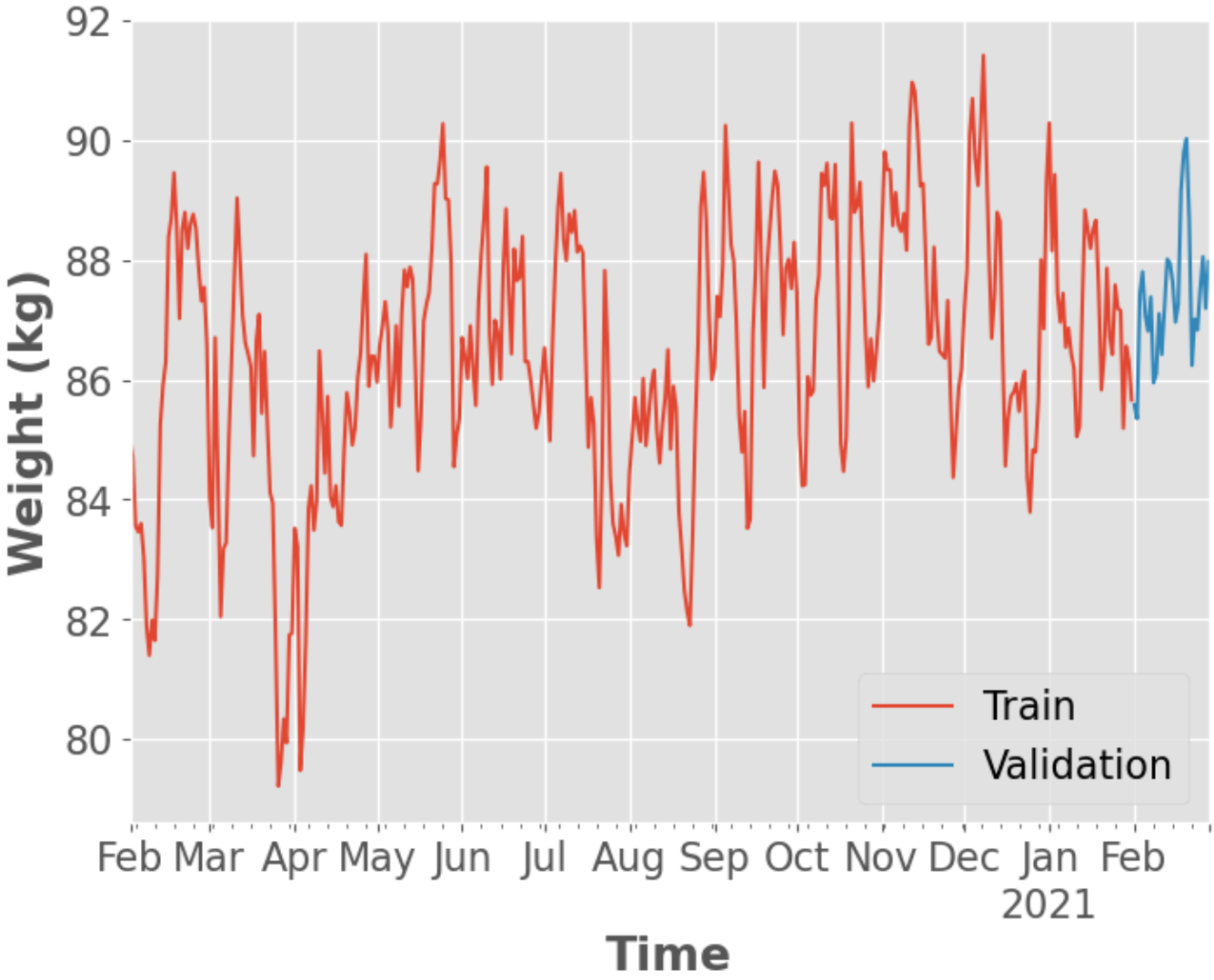
10. Weight data

This is just a record of daily body weight (kg) over the last year.

Method

No method here, this is just recorded data.

```
df_train = pd.read_csv("data/weight_train.csv", index_col=0, parse_dates=True)
df_valid = pd.read_csv("data/weight_valid.csv", index_col=0, parse_dates=True)
(pd.concat((df_train, df_valid), axis=1)
   .set_axis(["Train", "Validation"], axis=1)
   .plot
   .line(xlabel="Time", ylabel="Weight (kg)")
);
```



11. Monthly Champagne Sales

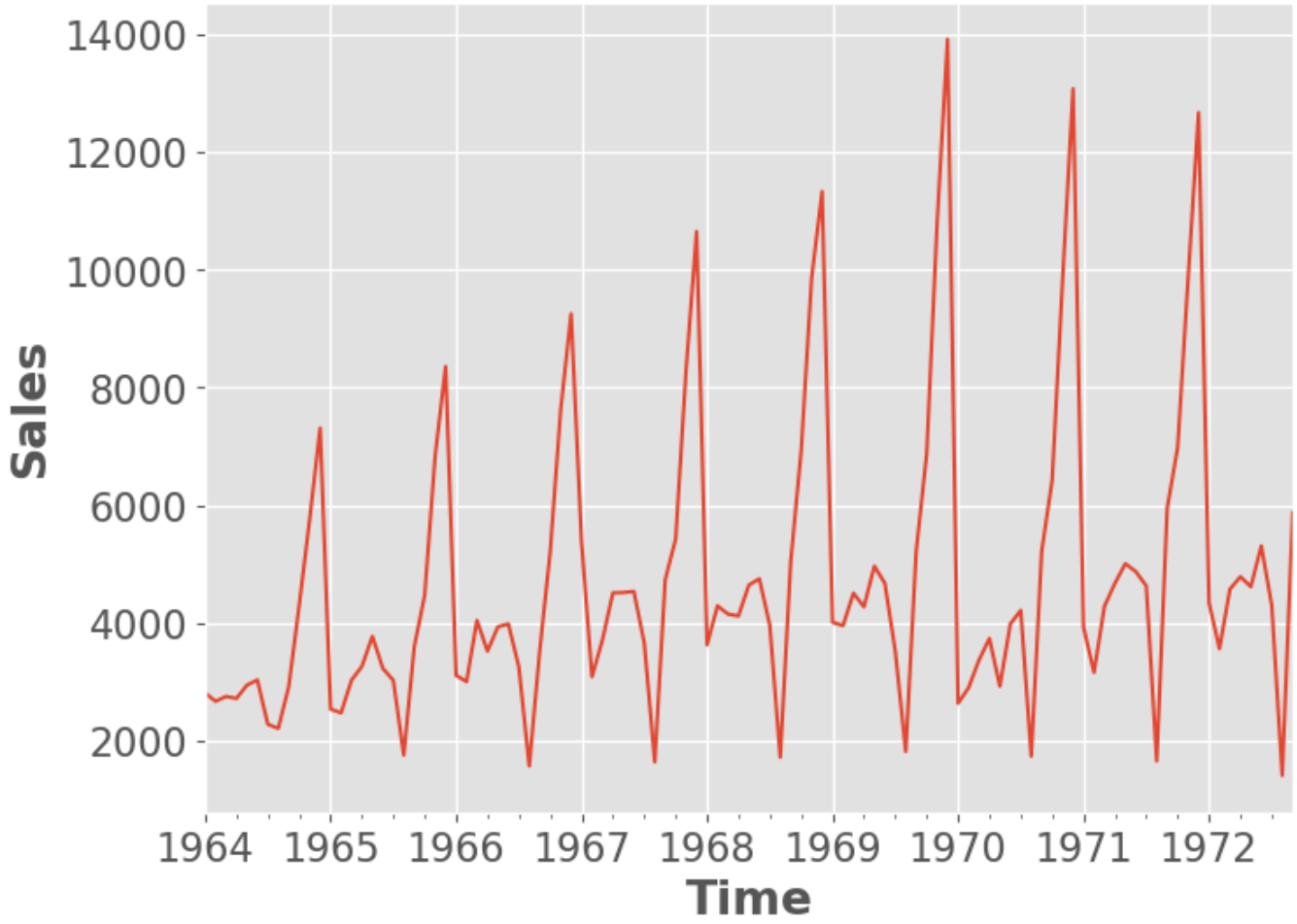
Description

This dataset describes the monthly sales of champagne for the Perrin Freres label from 1964 to 1972. The data was compiled by Makridakis and Wheelwright, 1989, and we'll be using a version that Jason Brownlee of Machine Learning Mastery has kindly made available here in csv format.

Method

Data is readily available online at the above link and we can read it in directly with Pandas.

```
url = "https://raw.githubusercontent.com/jbrownlee/Datasets/master/monthly_cha
df = (pd.read_csv(url, index_col=0, parse_dates=True)
        .rename_axis(index=["Time"])
        .resample("1M").mean()
     )
df.to_csv("data/champagne.csv")
df.plot.line(xlabel="Time", ylabel="Sales", legend=False);
```



12. Hourly Pollution Levels in Beijing

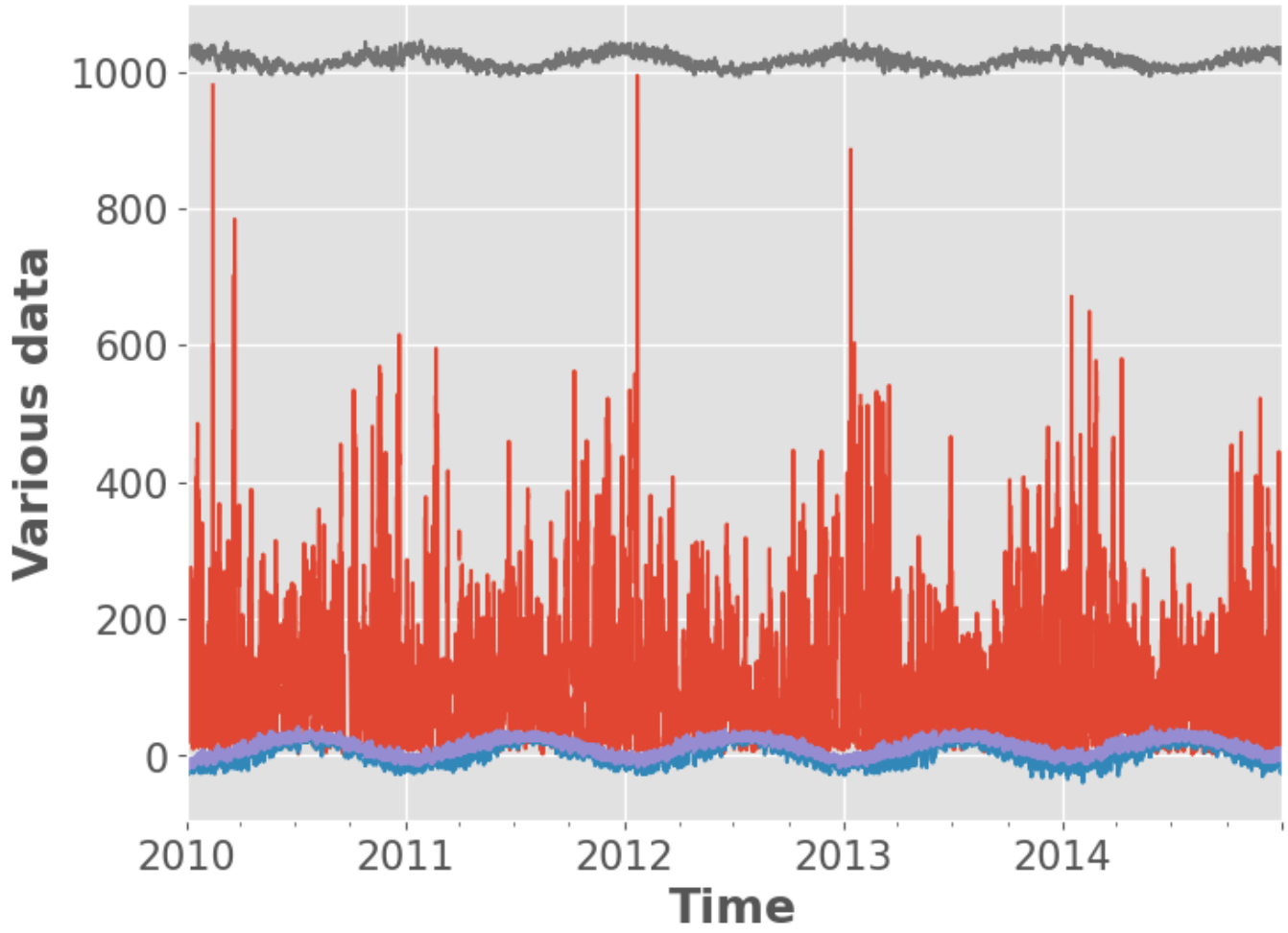
Description

This dataset is an hourly, multivariate dataset of pollution (PM2.5 concentration) and other meteorological variables described here. The data was compiled by Liang et al., 2015, and we'll be using a version that Jason Brownlee of Machine Learning Mastery has kindly made available here in csv format.

Method

Data is readily available online at the above link and we can read it in directly with Pandas.

```
df = (pd.read_csv("https://raw.githubusercontent.com/jbrownlee/Datasets/master
                  parse_dates={"Time" : [1, 2, 3, 4]},
                  date_parser=lambda x: datetime.strptime(x, "%Y %m %d %H"))
        .set_index("Time")
        .rename(columns={"TEMP": "Temperature", "PRES": "Pressure", "pm2.5": "
        .drop(columns=["No", "cbwd", "Iws", "Is", "Ir"])
    )
df.to_csv("data/pollution.csv")
df.plot.line(xlabel="Time", ylabel="Various data", legend=False);
```



13. Monthly Unemployment in Australia

Description

This dataset is a monthly dataset of the number of unemployed persons in Australia. The data is available through the Australian Bureau of Statistics.

Method

Data is readily available online at the above link and we can read it in directly with Pandas.

```
url = "https://www.abs.gov.au/statistics/labour/employment-and-unemployment/la
df = pd.read_excel(url, sheet_name="Data1", header=None, skiprows=10, usecols=
                   index_col=0, parse_dates=True, names=["Time", "Unemployed p
df *= 1000
df_train = df.loc[:"2012"]
df_valid = df.loc["2013":].subtract(70000)  # bias correction
df_train.to_csv("data/unemployed_train.csv")
df_valid.to_csv("data/unemployed_valid.csv")
(pd.concat((df_train, df_valid), axis=1)
   .set_axis(["Train", "Validation"], axis=1)
   .plot
   .line(xlabel="Time", ylabel="Unemployed persons")
);
```