Analyzing Time Series Data with Socrata and Python

In [ ]:
```
import os
!pip install sodapy
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheel
s/public/simple/
Collecting sodapy
  Downloading sodapy-2.2.0-py2.py3-none-any.whl (15 kB)
Collecting requests>=2.28.1
  Downloading requests-2.28.2-py3-none-any.whl (62 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 62.8/62.8 KB 2.5 MB/s eta 0:00:00
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.9/dist-packa
ges (from requests>=2.28.1->sodapy) (3.4)
Requirement already satisfied: urllib3<1.27,>=1.21.1 in /usr/local/lib/python3.9/d
ist-packages (from requests>=2.28.1->sodapy) (1.26.15)
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.9/dist
-packages (from requests>=2.28.1->sodapy) (2022.12.7)
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.
9/dist-packages (from requests>=2.28.1->sodapy) (2.0.12)
Installing collected packages: requests, sodapy
  Attempting uninstall: requests
    Found existing installation: requests 2.27.1
    Uninstalling requests-2.27.1:
      Successfully uninstalled requests-2.27.1
Successfully installed requests-2.28.2 sodapy-2.2.0
```

Milestone 1

Loading datasets into our notebook

We'll start by loading a city of Chicago permits dataset into a Pandas DataFrame. We'll use the open source sodapy Python Socrata module to do this.

In [ ]:
```
import pandas as pd
from sodapy import Socrata

# load Chicago permits data
chicago_permits_df = pd.read_csv(r'Building_Permits.csv')
```

```
<ipython-input-2-94a595401d0d>:5: DtypeWarning: Columns (75,76,77,78,79,80,81,82,8
3,84,85,86,87,88,89,90,91,92,93,94,95,96,97,98) have mixed types. Specify dtype op
tion on import or set low_memory=False.
  chicago_permits_df = pd.read_csv(r'Building_Permits.csv')
```

In the next few cells we'll do some exploration of our datasets using the `len`, `head`, and `value_counts` functions. We'll start by getting a sense of how many rows are in each of

our datasets with the `len` function.

Now let's see have a peek at the first 10 rows in each of those dataset using the `head` method. You can optionally pass a parameter for the number of rows you want to print if 5 isn't enough.

In [ ]:
```
print(len(chicago_permits_df))
chicago_permits_df.head(10)
```

```
225508
```

Out[ ]:

	ID	PERMIT#	PERMIT_TYPE	REVIEW_TYPE	APPLICATION_START_DATE	ISSUE_
0	1614287	100072880	PERMIT - RENOVATION/ALTERATION	STANDARD PLAN REVIEW	10/14/2005	01/03
1	1614342	100072909	PERMIT - NEW CONSTRUCTION	STANDARD PLAN REVIEW	12/05/2005	03/29
2	1614371	100072936	PERMIT - SIGNS	SIGN PERMIT	10/17/2005	01/12
3	1637148	100086388	PERMIT - RENOVATION/ALTERATION	SELF CERT	01/25/2006	01/26
4	1637165	100086395	PERMIT - RENOVATION/ALTERATION	SELF CERT	01/21/2006	01/21
5	1641218	100089035	PERMIT - SIGNS	SIGN PERMIT	01/10/2006	01/10
6	1641219	100089036	PERMIT - SIGNS	SIGN PERMIT	01/10/2006	01/10
7	1641220	100089037	PERMIT - SIGNS	SIGN PERMIT	01/11/2006	01/11
8	1641221	100089038	PERMIT - SIGNS	SIGN PERMIT	01/11/2006	01/11
9	1641222	100089039	PERMIT - SIGNS	SIGN PERMIT	01/11/2006	01/11

10 rows × 119 columns

Printing our dataframes like this gives us a sense of what columns exist, and quick sense of some of the values in the dataset. But there's an even better way to detrmine the top values for a particular column -- the `value_counts` method.

In [ ]:
```
chicago_permits_df["APPLICATION_START_DATE"].value_counts(dropna=False).head(10)
```

Out[ ]:
```
10/16/2007    329
03/30/2011    299
04/23/2012    290
04/24/2012    273
09/30/2009    260
NaN           255
04/01/2011    243
10/17/2007    242
04/19/2012    242
04/20/2012    242
Name: APPLICATION_START_DATE, dtype: int64
```

Select a subset of a DataFrame & Deal with missing values

In [ ]:
```
chicago_permits_df.shape
```

Out[ ]:
```
(225508, 119)
```

In [ ]:
```
chicago_permits_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 225508 entries, 0 to 225507
Columns: 119 entries, ID to LOCATION
dtypes: float64(23), int64(3), object(93)
memory usage: 204.7+ MB
```

In [ ]:
```
print(chicago_permits_df.isnull().sum())
```

```
ID                          0
PERMIT#                     0
PERMIT_TYPE                 0
REVIEW_TYPE                 3
APPLICATION_START_DATE    255
                         ...
XCOORDINATE                47
YCOORDINATE                47
LATITUDE                   68
LONGITUDE                  68
LOCATION                   68
Length: 119, dtype: int64
```

There are too many columns to analysize and we probably do not care the information in all the columns. So let's only select part of the dataframe we care about. Please make your own decision of columns selection based on what you would like to observie.

In [ ]:
```
chicago_permits_df.columns
```

Out[ ]:
```
Index(['ID', 'PERMIT#', 'PERMIT_TYPE', 'REVIEW_TYPE', 'APPLICATION_START_DATE',
       'ISSUE_DATE', 'PROCESSING_TIME', 'STREET_NUMBER', 'STREET DIRECTION',
       'STREET_NAME',
       ...
       'PIN9', 'PIN10', 'COMMUNITY_AREA', 'CENSUS_TRACT', 'WARD',
       'XCOORDINATE', 'YCOORDINATE', 'LATITUDE', 'LONGITUDE', 'LOCATION'],
      dtype='object', length=119)
```

In [ ]:
```
chicago_permits_df_sub = chicago_permits_df[['ID', 'PERMIT#', 'PERMIT_TYPE', 'REVIE
```

In [ ]:
```
chicago_permits_df_sub.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 225508 entries, 0 to 225507
Data columns (total 10 columns):
 #   Column                  Non-Null Count   Dtype  
---  ------                  --------------   -----  
 0   ID                      225508 non-null  int64  
 1   PERMIT#                 225508 non-null  int64  
 2   PERMIT_TYPE             225508 non-null  object 
 3   REVIEW_TYPE             225505 non-null  object 
 4   APPLICATION_START_DATE  225253 non-null  object 
 5   STREET DIRECTION        225508 non-null  object 
 6   COMMUNITY_AREA          160321 non-null  float64
 7   LATITUDE                225440 non-null  float64
 8   LONGITUDE               225440 non-null  float64
 9   LOCATION                225440 non-null  object 
dtypes: float64(3), int64(2), object(5)
memory usage: 17.2+ MB
```

The value counts make it clear that a lot of the values in the "application_start_date" column are missing or null. There a variety of ways you can handle missing data, but removing incomplete rows is the simplest. In the next cell, we'll remove rows with null dates. There are also a lot of columns in the permits dataset that we won't use in this analysis. So we'll also filter down our dataset to just the columns we're interested in to reduce the amount of extraneous information.

In [ ]:
```
chicago_permits_df_sub_notnull = chicago_permits_df_sub[chicago_permits_df_sub["APP
chicago_permits_df_sub_notnull = chicago_permits_df_sub_notnull[["APPLICATION_START
chicago_permits_df_sub_notnull.head(10)
```

Out[ ]:

	APPLICATION_START_DATE
0	10/14/2005
1	12/05/2005
2	10/17/2005
3	01/25/2006
4	01/21/2006
5	01/10/2006
6	01/10/2006
7	01/11/2006
8	01/11/2006
9	01/11/2006

Print descriptive statistics

Use info() to get some basic summary about the dataframe, and also describe() helps us to get some descriptive statistics about columns containing the numeric values.

In [ ]:
```
chicago_permits_df_sub.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 225508 entries, 0 to 225507
Data columns (total 10 columns):
 #   Column                  Non-Null Count   Dtype  
---  ------                  --------------   -----  
 0   ID                      225508 non-null  int64  
 1   PERMIT#                 225508 non-null  int64  
 2   PERMIT_TYPE             225508 non-null  object 
 3   REVIEW_TYPE             225505 non-null  object 
 4   APPLICATION_START_DATE  225253 non-null  object 
 5   STREET DIRECTION        225508 non-null  object 
 6   COMMUNITY_AREA          160321 non-null  float64
 7   LATITUDE                225440 non-null  float64
 8   LONGITUDE               225440 non-null  float64
 9   LOCATION                225440 non-null  object 
dtypes: float64(3), int64(2), object(5)
memory usage: 17.2+ MB
```

In [ ]:
```
chicago_permits_df_sub.describe()
```

Out[ ]:

	ID	PERMIT#	COMMUNITY_AREA	LATITUDE	LONGITUDE
count	2.255080e+05	2.255080e+05	160321.000000	225440.000000	225440.000000
mean	2.103785e+06	1.003220e+08	32.072268	41.867700	-87.673926
std	3.338493e+05	2.736262e+05	22.369954	0.086502	0.059792
min	9.706200e+05	1.003330e+05	0.000000	41.644702	-87.914534
25%	1.862418e+06	1.002093e+08	12.000000	41.796370	-87.709467
50%	2.060406e+06	1.002967e+08	28.000000	41.885044	-87.664346
75%	2.253390e+06	1.003840e+08	49.000000	41.932321	-87.632121
max	3.306687e+06	1.010054e+08	77.000000	42.022645	-87.524677

Aggregating based on date

For the purpose of analysizing APPLICATION_START_DATE distribution (feel free to explore other variables), in the dataframe "chicago_permits_df_sub_notnull", each row in our dataset corresponds to a permit application and the only column we've preserved is the date of the application. The task of forecasting number of permit applications is not really interesting (or reliable) at the granularity of day. Predicting at the granularity of week might be interesting, but let's start by grouping by month. To get some datatime functionality from Python, we'll convert our date column to a datetime type.

In [ ]:
```
import datetime

fixed_dates_df = chicago_permits_df_sub_notnull.copy()
fixed_dates_df["APPLICATION_START_DATE"] = fixed_dates_df["APPLICATION_START_DATE"]
fixed_dates_df = fixed_dates_df.set_index(fixed_dates_df["APPLICATION_START_DATE"])
grouped = fixed_dates_df.resample("M").count()
data_df = pd.DataFrame({"count": grouped.values.flatten()}, index=grouped.index)
data_df.head(10)
```

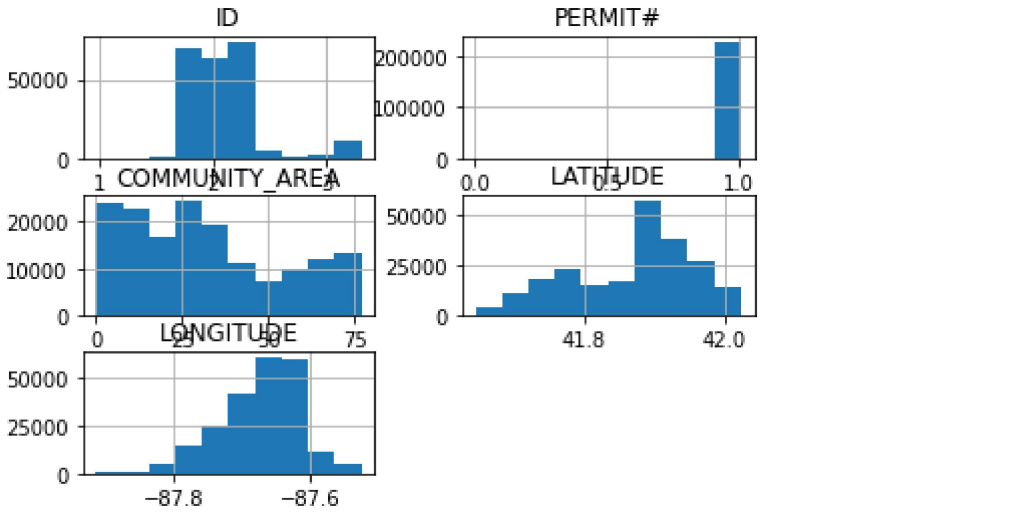
Out[ ]:

APPLICATION_START_DATE	count
2002-02-28	1
2002-03-31	0
2002-04-30	0
2002-05-31	0
2002-06-30	0
2002-07-31	0
2002-08-31	0
2002-09-30	0
2002-10-31	0
2002-11-30	0

Plotting a histogram

In [ ]:
```
chicago_permits_df_sub.hist()
```

Out[ ]:
```
array([[<Axes: title={'center': 'ID'}>,
        <Axes: title={'center': 'PERMIT#'}>],
       [<Axes: title={'center': 'COMMUNITY_AREA'}>,
        <Axes: title={'center': 'LATITUDE'}>],
       [<Axes: title={'center': 'LONGITUDE'}>, <Axes: >]], dtype=object)
```

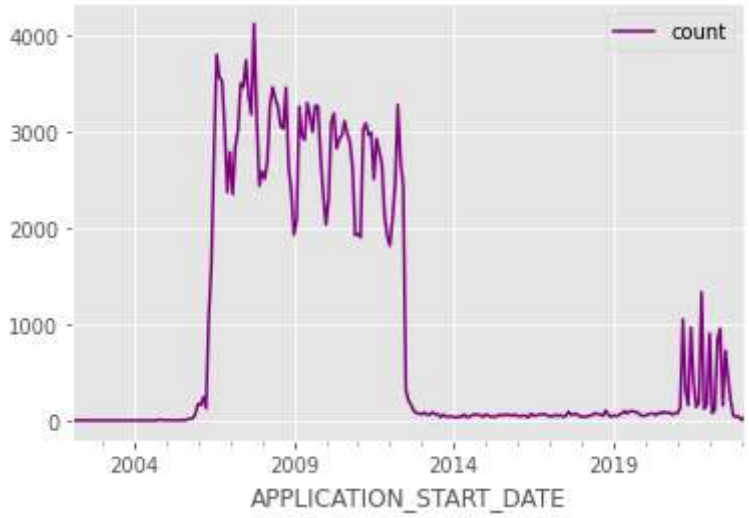


Plotting a time series

In [ ]:
```
import matplotlib.pyplot as plt
from pandas.plotting import register_matplotlib_converters
register_matplotlib_converters()
plt.style.use("ggplot")
```

```
data_df.plot(color="purple")
```

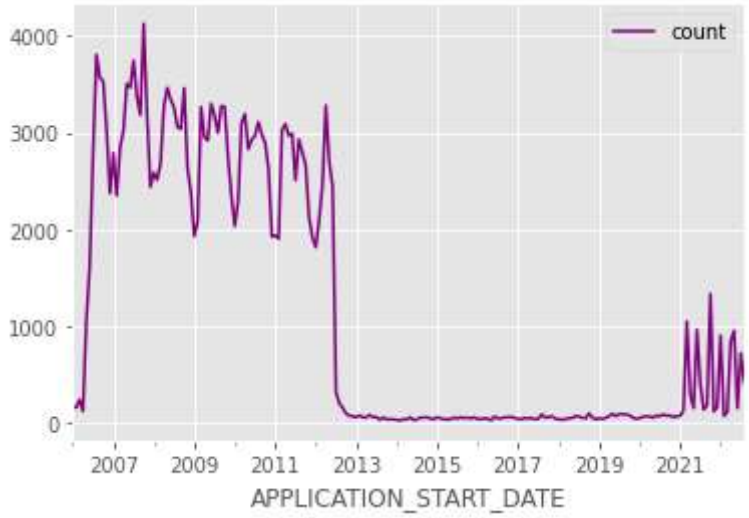
Out[ ]: <Axes: xlabel='APPLICATION_START_DATE'>



Notice how the number of applications in 2005 and before looks suspiciously low. My intuition is that this is a data problem. Let's remove all data from before 2006, since bad data will impact the accuracy of our model. Let's also remove data from before October of this year, since October is incomplete.

In [ ]:
```
def is_between_2006_and_now(date):
    return date > datetime.datetime(2006, 1, 1) and date < datetime.datetime(2022,

data_df = data_df[data_df.index.to_series().apply(is_between_2006_and_now)]
data_df.plot(color="purple")
```

Out[ ]: <Axes: xlabel='APPLICATION_START_DATE'>



This plot makes two things pretty clear. First, there are some clear trends in the time series -- for example, an increase between 2011 and 2017, followed by a levelling off of permit applications. Second, there is a cyclic nature to the time series, which is indicative of there being seasonal variation in permit applications (which isn't surprising).

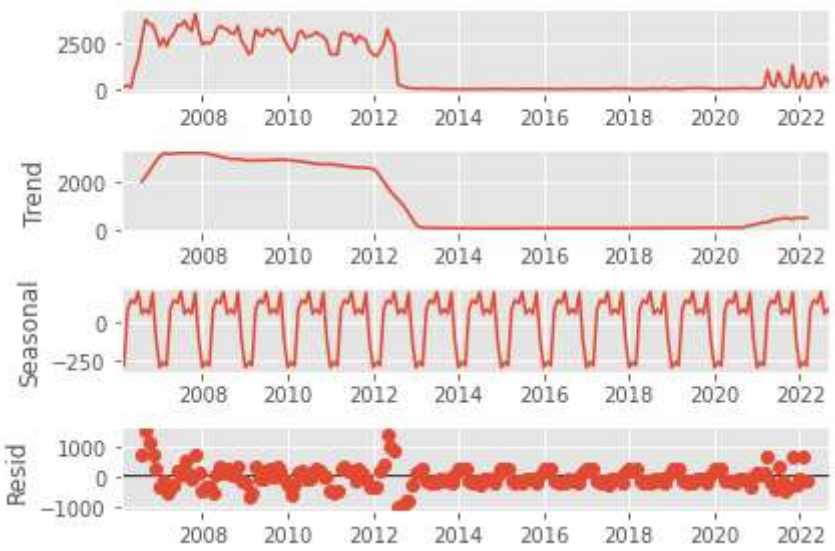
Milestone 2

Part 1

Understanding long-term trends and seasonality

The types of modeling used on time series assume "stationarity". For a time series to be stationary, mean and variance should be constant over time (and of course, most real life time series will not be stationary). To satisfy this assumption, we need to remove trend and seasonality from our series. The underlying modeling code we will use will do this for us. Doing this ourselves explicitly enables us to make educated guesses about reasonable parameters for our model. We'll use the `seasonal_decompose` function to do this.

In [ ]:
```
from statsmodels.tsa.seasonal import seasonal_decompose
result = seasonal_decompose(data_df)
fig = result.plot()
```



Forecasting with Prophet

Next, we'll create a Prophet model. Doing so requires only that we rename our count column, and that we have a new column "ds" which is our timestamps. After training our model, we'll use it to predict a year into the future.

In [ ]:
```
!pip install prophet
from fbprophet import Prophet
model = Prophet()
train_df = data_df.rename(columns={"count":'y'})
```

```
train_df["ds"] = train_df.index
model.fit(train_df)
```

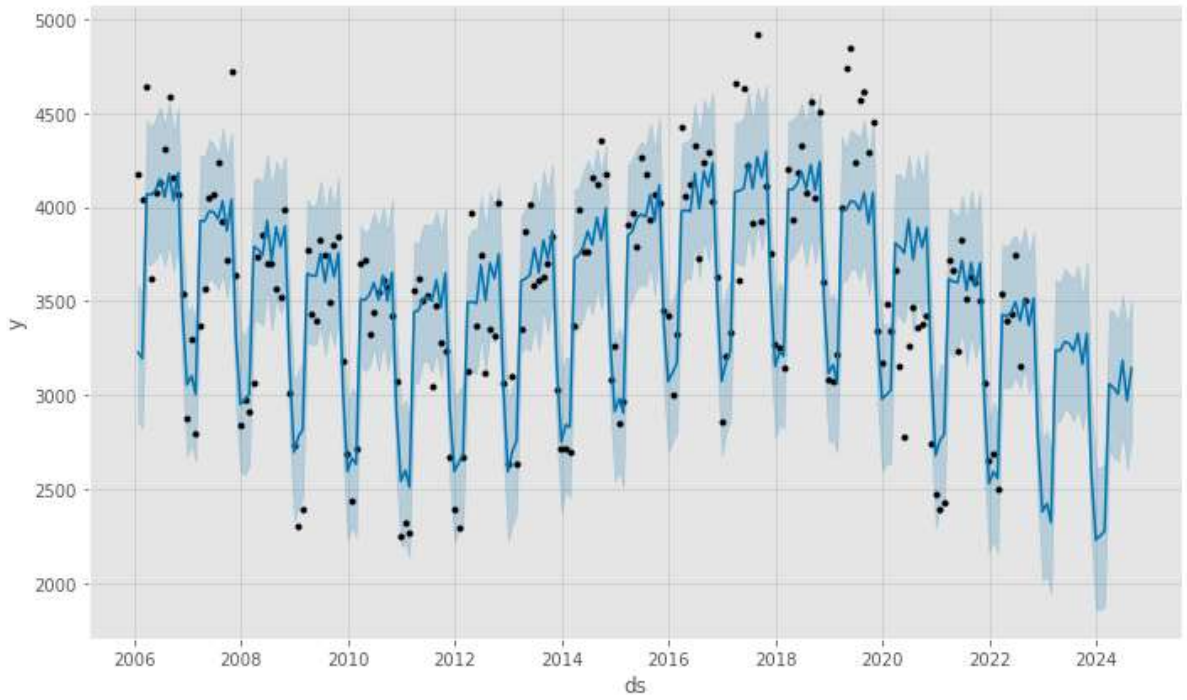
```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheel
s/public/simple/
Requirement already satisfied: prophet in /usr/local/lib/python3.9/dist-packages
(1.1.2)
Requirement already satisfied: pandas>=1.0.4 in /usr/local/lib/python3.9/dist-pack
ages (from prophet) (1.4.4)
Requirement already satisfied: LunarCalendar>=0.0.9 in /usr/local/lib/python3.9/di
st-packages (from prophet) (0.0.9)
Requirement already satisfied: matplotlib>=2.0.0 in /usr/local/lib/python3.9/dist-
packages (from prophet) (3.7.1)
Requirement already satisfied: convertdate>=2.1.2 in /usr/local/lib/python3.9/dist
-packages (from prophet) (2.4.0)
Collecting cmdstanpy>=1.0.4
  Downloading cmdstanpy-1.1.0-py3-none-any.whl (83 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 83.2/83.2 KB 2.8 MB/s eta 0:00:00
Requirement already satisfied: holidays>=0.14.2 in /usr/local/lib/python3.9/dist-p
ackages (from prophet) (0.21.13)
Requirement already satisfied: python-dateutil>=2.8.0 in /usr/local/lib/python3.9/
dist-packages (from prophet) (2.8.2)
Requirement already satisfied: tqdm>=4.36.1 in /usr/local/lib/python3.9/dist-packa
ges (from prophet) (4.65.0)
Requirement already satisfied: numpy>=1.15.4 in /usr/local/lib/python3.9/dist-pack
ages (from prophet) (1.22.4)
Requirement already satisfied: pymeeus<=1,>=0.3.13 in /usr/local/lib/python3.9/dis
t-packages (from convertdate>=2.1.2->prophet) (0.5.12)
Requirement already satisfied: hijri-converter in /usr/local/lib/python3.9/dist-pa
ckages (from holidays>=0.14.2->prophet) (2.2.4)
Requirement already satisfied: korean-lunar-calendar in /usr/local/lib/python3.9/d
ist-packages (from holidays>=0.14.2->prophet) (0.3.1)
Requirement already satisfied: pytz in /usr/local/lib/python3.9/dist-packages (fro
m LunarCalendar>=0.0.9->prophet) (2022.7.1)
Requirement already satisfied: ephem>=3.7.5.3 in /usr/local/lib/python3.9/dist-pac
kages (from LunarCalendar>=0.0.9->prophet) (4.1.4)
Requirement already satisfied: contourpy>=1.0.1 in /usr/local/lib/python3.9/dist-p
ackages (from matplotlib>=2.0.0->prophet) (1.0.7)
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.9/dist-packa
ges (from matplotlib>=2.0.0->prophet) (0.11.0)
Requirement already satisfied: importlib-resources>=3.2.0 in /usr/local/lib/python
3.9/dist-packages (from matplotlib>=2.0.0->prophet) (5.12.0)
Requirement already satisfied: pyparsing>=2.3.1 in /usr/local/lib/python3.9/dist-p
ackages (from matplotlib>=2.0.0->prophet) (3.0.9)
Requirement already satisfied: kiwisolver>=1.0.1 in /usr/local/lib/python3.9/dist-
packages (from matplotlib>=2.0.0->prophet) (1.4.4)
Requirement already satisfied: fonttools>=4.22.0 in /usr/local/lib/python3.9/dist-
packages (from matplotlib>=2.0.0->prophet) (4.39.2)
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.9/dist-pa
ckages (from matplotlib>=2.0.0->prophet) (23.0)
Requirement already satisfied: pillow>=6.2.0 in /usr/local/lib/python3.9/dist-pack
ages (from matplotlib>=2.0.0->prophet) (8.4.0)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.9/dist-packages
(from python-dateutil>=2.8.0->prophet) (1.16.0)
Requirement already satisfied: zipp>=3.1.0 in /usr/local/lib/python3.9/dist-packag
es (from importlib-resources>=3.2.0->matplotlib>=2.0.0->prophet) (3.15.0)
Installing collected packages: cmdstanpy
  Attempting uninstall: cmdstanpy
    Found existing installation: cmdstanpy 0.9.5
```

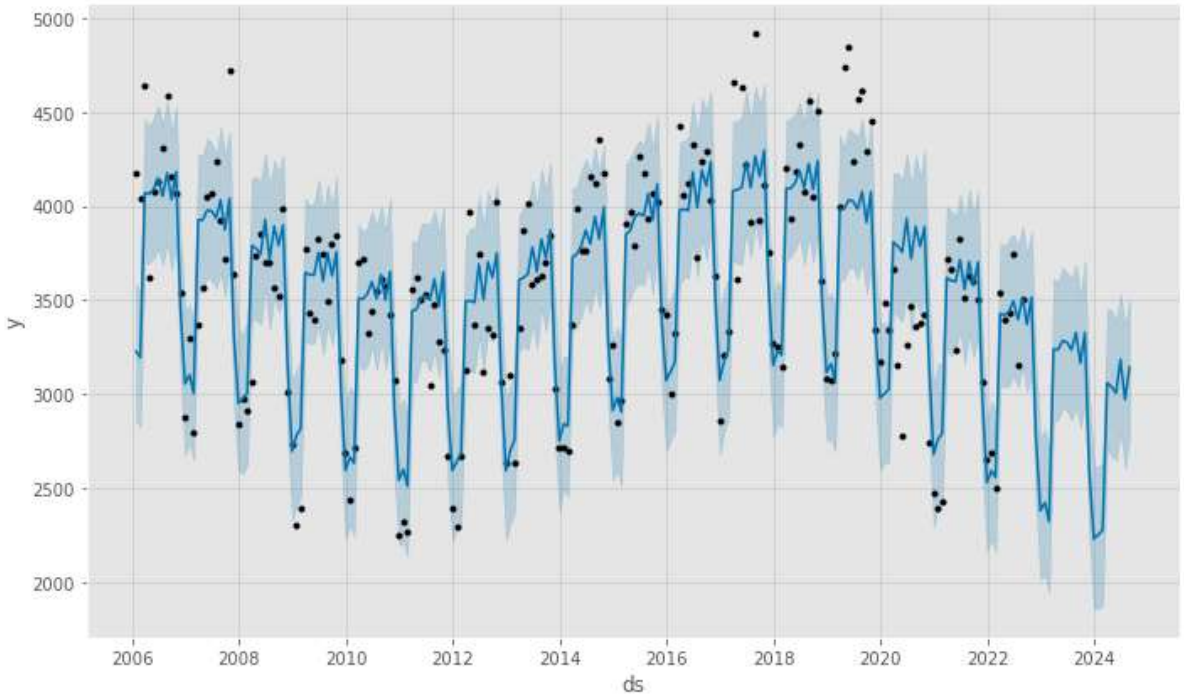
```
    Uninstalling cmdstanpy-0.9.5:
      Successfully uninstalled cmdstanpy-0.9.5
Successfully installed cmdstanpy-1.1.0
```

```
---------------------------------------------------------------------------
ModuleNotFoundError                       Traceback (most recent call last)
<ipython-input-22-4652dee4b1bc> in <module>
      1 get_ipython().system('pip install prophet')
----> 2 from fbprophet import Prophet
      3 model = Prophet()
      4 train_df = data_df.rename(columns={"count":'y'})
      5 train_df["ds"] = train_df.index

ModuleNotFoundError: No module named 'fbprophet'

---------------------------------------------------------------------------
NOTE: If your import is failing due to a missing package, you can
manually install dependencies using either !pip or !apt.

To view examples of installing some common dependencies, click the
"Open Examples" button below.
---------------------------------------------------------------------------
```

In [ ]:
```
pd.plotting.register_matplotlib_converters()
future = model.make_future_dataframe(24, freq='M', include_history=True)
forecast = model.predict(future)
model.plot(forecast)
```

Out[ ]:





In [ ]:
```
train_df
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-23-ef797c2b3b5b> in <module>
----> 1 train_df

NameError: name 'train_df' is not defined
```

In [ ]:
```
train_df.size
```

Out[ ]:
```
400
```

Q 1.1

What are your observations about long-term trends and seasonality?

In [ ]:
```
#In the future there is a decline in the number of applications. There is a cyclic
# it is based on seasonailty.Think there will be an increase from 2024-2028
```

Q 1.2

Try forecasting with Prophet using different time periods (different number of months). What do you find interesting about the predictions?

In [ ]:
```
#I found that the patterns are still cyclic and follow the same pattern as the rest
```

Part 2

Export the Chicago data portal "Sidewalk_Cafe_Permits.csv" from https://data.cityofchicago.org/Community-Economic-Development/Sidewalk-Cafe-Permits/nxj5-ix6z Please also read the description about this dateset.

Q2

We have learned data loading, cleaning, aggregation in milestone 1. Follow a similar procedure and fill in the **TODO**s in the code boxes. Your result should be the same as (or similar to) the results shown below the code boxes.

In [ ]:
```
# TODO 1:
# 1. Read the data from csv and save the dataframe named "towed_vehicles_df"
# 2. Print length of the towed_vehicles_df, and print the first 10 rows of towed_ve
towed_vehicles_df = pd.read_csv(r"Sidewalk_Cafe_Permits.csv")
print(len(towed_vehicles_df))
towed_vehicles_df.head(10)
```

```
21658
```

Out[ ]:

	PERMIT NUMBER	ACCOUNT NUMBER	SITE NUMBER	LEGAL NAME	DOING BUSINESS AS NAME	ISSUED DATE	EXPIRATION DATE	PAYMEN DA
0	1556602	328992	1	THE LIFEWAY KEFIR SHOP LLC	LIFEWAY KEFIR SHOP	07/16/2021	02/28/2022	07/16/20
1	1531303	399498	1	JERRY'S SANDWICHES LS, LLC	JERRY'S SANDWICHES	07/16/2021	02/28/2022	07/16/20
2	1553078	463188	1	ETTA RIVER NORTH, LLC	ETTA	07/16/2021	02/28/2022	07/16/20
3	1534556	252742	1	SQUARE KITCHEN, LLC	FORK	07/16/2021	02/28/2022	07/16/20
4	1556006	337178	1	ROCCO'S, LLC	RANALLI'S	07/16/2021	02/28/2022	07/16/20
5	1536621	414414	1	BBSC #4 LLC	BROWN BAG SEAFOOD CO.	07/16/2021	02/28/2022	07/16/20
6	1559950	34063	1	GASTHAUS ZUM LOEWEN, INC.	THE REVELER	07/16/2021	02/28/2022	07/16/20
7	1540360	23957	1	TEMPO CAFE LIMITED	TEMPO CAFE	07/16/2021	02/28/2022	07/15/20
8	1543349	425540	1	MI FOGATA INC.	MI FOGATA INC.	07/16/2021	02/28/2022	07/16/20
9	1555187	340126	1	SHINE RESTAURANT CORP.	SHINE RESTAURANT, RISE SUSHI RESTAURANT	07/17/2021	02/28/2022	07/14/20

10 rows × 22 columns

In [ ]:
```
# TODO 2:
# Print the 10 dates ("ISSUED DATE") with the most number of business licenses issu
towed_vehicles_df.columns
issued_count = towed_vehicles_df.groupby('ISSUED DATE')['PERMIT NUMBER'].count()
top_10 = issued_count.sort_values(ascending=False).head(10)
print(top_10)
```

```
ISSUED DATE
03/20/2014    76
03/25/2015    72
03/19/2013    67
04/02/2015    65
03/05/2009    59
04/10/2013    59
03/22/2018    53
03/09/2011    52
02/26/2016    50
06/01/2021    48
Name: PERMIT NUMBER, dtype: int64
```

In [ ]:
```
# TODO 3:
# 1.Extract only the dates ("Tow Date") from towed_vehicles_df
# 2.Drop the rows with notnull values and reset the index
# 3.Print first 10 rows of the result towed_vehicles_date_df
tow_date = towed_vehicles_df[['ISSUED DATE']]
tow_date.dropna()
tow_date.head(10)
```

Out[ ]:

	ISSUED DATE
0	07/16/2021
1	07/16/2021
2	07/16/2021
3	07/16/2021
4	07/16/2021
5	07/16/2021
6	07/16/2021
7	07/16/2021
8	07/16/2021
9	07/17/2021

Part 3

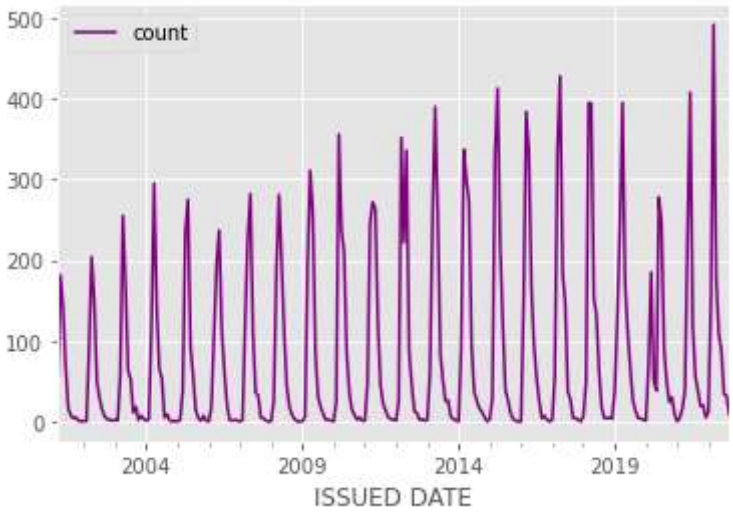
Time series

In [ ]:
```
fixed_cafe_dates_df = towed_vehicles_df.copy()
fixed_cafe_dates_df["ISSUED DATE"] = fixed_cafe_dates_df["ISSUED DATE"].apply(pd.to
fixed_cafe_dates_df = fixed_cafe_dates_df.set_index(fixed_cafe_dates_df["ISSUED DAT
grouped_cafe_dates = fixed_cafe_dates_df.resample("M").count()
data_cafe_df = pd.DataFrame({"count": grouped_cafe_dates.values.flatten()}, index=g
data_cafe_df.head(10)
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-29-396b8c6297f4> in <module>
      3 fixed_cafe_dates_df = fixed_cafe_dates_df.set_index(fixed_cafe_dates_df["I
SSUED DATE"])
      4 grouped_cafe_dates = fixed_cafe_dates_df.resample("M").count()
----> 5 data_cafe_df = pd.DataFrame({"count": grouped_cafe_dates.values.flatten
()}, index=grouped_cafe_dates.index)
      6 data_cafe_df.head(10)

/usr/local/lib/python3.9/dist-packages/pandas/core/frame.py in __init__(self, dat
a, index, columns, dtype, copy)
    634         elif isinstance(data, dict):
    635             # GH#38939 de facto copy defaults to False only in non-dict ca
ses
--> 636             mgr = dict_to_mgr(data, index, columns, dtype=dtype, copy=cop
y, typ=manager)
    637         elif isinstance(data, ma.MaskedArray):
    638             import numpy.ma.mrecords as mrecords

/usr/local/lib/python3.9/dist-packages/pandas/core/internals/construction.py in di
ct_to_mgr(data, index, columns, dtype, typ, copy)
    500         # TODO: can we get rid of the dt64tz special case above?
    501
--> 502     return arrays_to_mgr(arrays, columns, index, dtype=dtype, typ=typ, con
solidate=copy)
    503
    504

/usr/local/lib/python3.9/dist-packages/pandas/core/internals/construction.py in ar
rays_to_mgr(arrays, columns, index, dtype, verify_integrity, typ, consolidate)
    123
    124         # don't force copy because getting jammed in an ndarray anyway
--> 125         arrays = _homogenize(arrays, index, dtype)
    126         # _homogenize ensures
    127         #  - all(len(x) == len(index) for x in arrays)

/usr/local/lib/python3.9/dist-packages/pandas/core/internals/construction.py in _h
omogenize(data, index, dtype)
    626                 val, index, dtype=dtype, copy=False, raise_cast_failure=Fa
lse
    627             )
--> 628             com.require_length_match(val, index)
    629
    630         homogenized.append(val)

/usr/local/lib/python3.9/dist-packages/pandas/core/common.py in require_length_mat
ch(data, index)
    555     """
    556     if len(data) != len(index):
--> 557         raise ValueError(
    558             "Length of values "
    559             f"({len(data)}) "

ValueError: Length of values (5830) does not match length of index (265)
```

In [ ]:
```
data_cafe_df.plot(color="purple")
```

Out[ ]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x1df2c411e50>
```

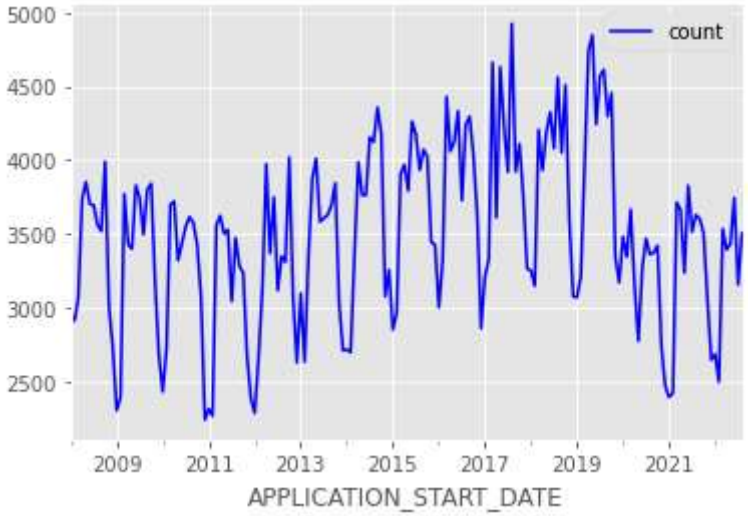


In [ ]:
```
def is_between_2008_and_now(date):
    return date > datetime.datetime(2008, 1, 1) and date < datetime.datetime(2022,

data_cafe_df = data_cafe_df[data_cafe_df.index.to_series().apply(is_between_2008_an
data_cafe_df.plot(color="purple")
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-27-6687ca047baf> in <module>
      2     return date > datetime.datetime(2008, 1, 1) and date < datetime.dateti
me(2022, 9, 27)
      3
----> 4 data_cafe_df = data_cafe_df[data_cafe_df.index.to_series().apply(is_betwee
n_2008_and_now)]
      5 data_cafe_df.plot(color="purple")

NameError: name 'data_cafe_df' is not defined
```

In [ ]:
```
dataperm_df = data_df[data_df.index.to_series().apply(is_between_2008_and_now)]
dataperm_df.plot(color="blue")
```

Out[ ]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x1df70a2e070>
```

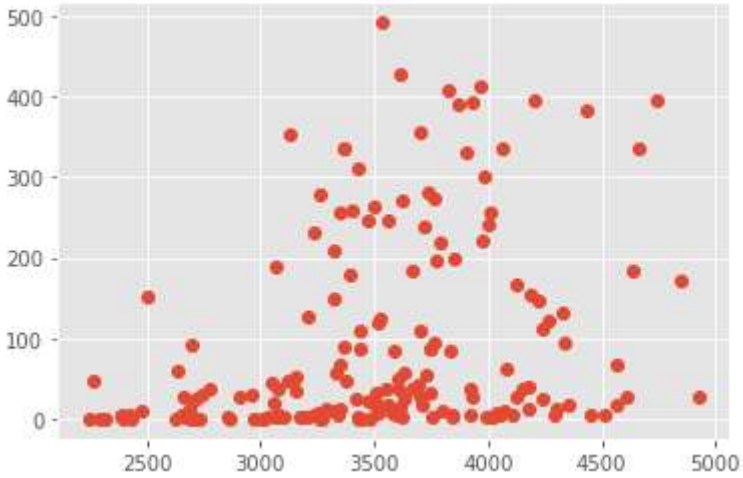


Simple linear regression

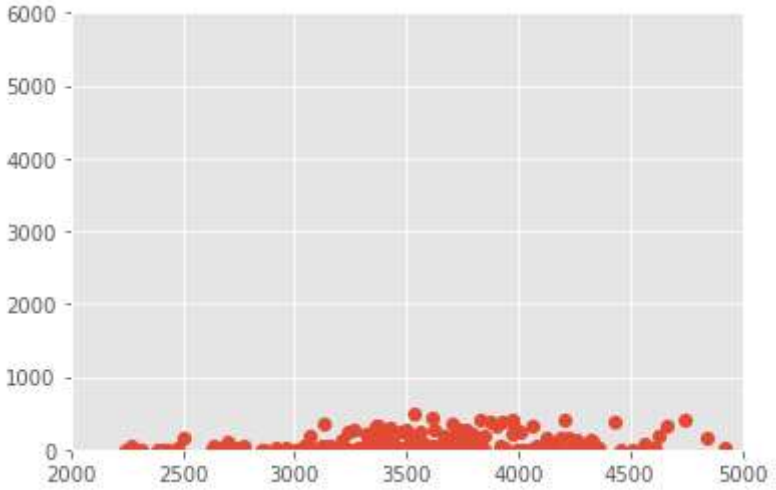
We will explore the relationship between building permits and sidewalk cafe permits, using simple linear regression.

In [ ]:
```
perms, cafe = dataperm_df.to_numpy(), data_cafe_df.to_numpy()
```

In [ ]:
```
_ = plt.plot(perms,cafe,'o')
```



In [ ]:
```
_ = plt.plot(perms,cafe,'o')
_ = plt.ylim([0,6000])
_ = plt.xlim([2000,5000])
plt.show()
```



In [ ]:
```
import numpy as np
AA = np.vstack([perms.T, np.ones(len(perms))]).T
AA.shape
```

Out[ ]:
```
(176, 2)
```

In [ ]:
```
m,c = np.linalg.lstsq(AA,cafe,rcond=None)[0]
```

In [ ]:
```
_ = plt.plot(perms,cafe,'o')
_ = plt.plot(perms,m*perms+c, 'r', label='Fitted line cafe = %.2f permits + %.2f' %
_ = plt.legend()
_ = plt.xlabel('Building Permits')
_ = plt.ylabel('Sidewalk cafe', rotation='horizontal')
_ = plt.title('Building Permits vs Sidewalk cafe')
_ = plt.xlim([2000,5000])
_ = plt.ylim([0,6000])
_ = plt.grid(b=True, which='major', color='#666666', linestyle='-')
plt.show()
```

In [ ]:
```
# remove outlier and redo the regression:
cafe[cafe.argmax()]=cafe.mean()
```

In [ ]:
```
AA = np.vstack([perms.T, np.ones(len(perms))]).T
m,c = np.linalg.lstsq(AA,cafe,rcond=None)[0]
_ = plt.plot(perms,cafe,'o')
_ = plt.plot(perms,m*perms+c, 'r', label='Fitted line cafe = %.2f permits + %.2f' %
_ = plt.legend()
_ = plt.xlabel('Building Permits')
_ = plt.ylabel('Sidewalk cafe', rotation='horizontal')
_ = plt.title('Building Permits vs Sidewalk cafe')
_ = plt.xlim([2000,5000])
_ = plt.ylim([0,6000])
_ = plt.grid(b=True, which='major', color='#666666', linestyle='-')
plt.show()
```

In [ ]:
```
#TODO: Learn Polynomial regression with the tutoriols. https://www.w3schools.com/py
#https://medium.com/analytics-vidhya/polynomial-fitting-in-python-using-just-one-li
#Polynomial regression on the Building Permits and Sidewalk cafe dataset.
```

Q3

Turn in an example where you compare two data sets using a regression analysis. It can be a simple linear regression as the instructor did in the class or more sophisticated if you would like. You can use Python or Matlab or even Excel if you want.

In [ ]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression

df1 = pd.read_csv("weight-height.csv")
df2 = pd.read_csv("SOCR-HeightWeight.csv")
a, b = np.polyfit(df1['Height'], df1['Weight'], 1)
c, d = np.polyfit(df2['Height(Inches)'], df2['Weight(Pounds)'], 1)
plt.scatter(df1['Height'], df1['Weight'], label='Data set 1')
plt.scatter(df2['Height(Inches)'], df2['Weight(Pounds)'], label='Data set 2')
x = df1['Height']
plt.plot(x, a*x+b, color='purple', label = 'Data set 1 Line of Best Fit')
```

```
x = df2['Height(Inches)']
plt.plot(x, c*x+d, color='yellow', label = 'Data set 2 Line of Best Fit')



plt.legend()
plt.xlabel('Height')
plt.ylabel('Weight')

plt.show()
```