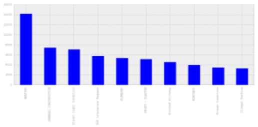
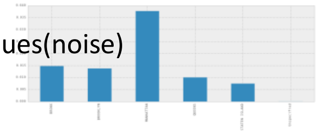
COOKBOOKS

CHAPTER1: reading from a CSV file.
- Reading from a csv file. sep**. encoding. parse_dates.**
- Selecting column.
- Plotting a column. **plot()**
- PUTTING IT ALL TOGETHER
- Reading a table from the web.

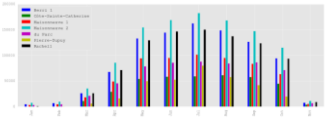
CHAPTER2: Selecting data & finding the most common complaint type
- Selecting columns and rows.
-Selecting multiple columns.
- What's the most common complaint type. **Counting. value_counts() .plot(kind='Bar')**



CHAPTER3: - Which borough has the most noise complaints (or, more selecting data)
- Selecting only noise complaints(selecting a certain value)
- A digression in numpy array. on numpy arrays and series
- So, Which borough has the most noise complaints? selecting boroughs with most values(noise)
plot(kind='bar')

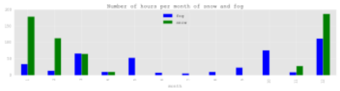


CHAPTER 4: Find out on which weekday people bike the most with groupby and aggregate
- Adding a 'weekday' column to our dataframe**. .index .day .weekday**
- Adding up the cyclists by weekday. **aggregate(sum) # Group the rows by weekday and then add up all the values with the same weekday # alternative with pivot table plot(kind='bar')**
- Putting it together.



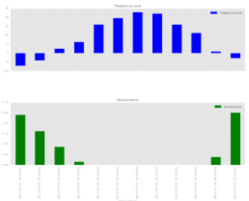
CHAPTER 5: Combining dataframes and scraping Canadian weather data.
- Summary**. pd.read_csv(index_col) .plot(figsize=(15, 6), color='b')**
- Downloading one month of weather data. **url_template= .columns dropna(axis=1, how='any')**
- Plotting the temperature by hour of day.
- Getting the whole year of **data. pd.contact()**
- Saving to a CSV. **weather_2012.to_csv('C:/Users/Michael/Desktop/cookbook/data/weather_2012.csv')**

CHAPTER 6: String Operations- Which month was the snowiest
- String operations. **str.contains('string')**
- Your turn. **.plot(kind='bar', title='Number of hours per month of snow and fog')**



- If we wanted the median temperature each month, we could use the resample() method like this:
weather_2012['Temp (C)'].resample('m', how=np.median).plot(kind='bar')

- Plotting temperature and snowiness stats together.
stats = pd.concat([temperature, snowiness], axis=1)
stats.plot(kind='bar', subplots=True, figsize=(15, 10))



CHAPTER 7: Cleaning up messy data
- How do we know if it's messy? unique()
- Fixing the nan values and string/float confusion. **pd.read_csv(na_values=..)**
- What's up with the dashes? truncate the strings(weghalen na een bepaalde range) **np.nan**(vervangt value met nan**) # Let's say the zips starting with '0' and '1' are okay, for now. (this isn't actually true -- 13221 is in Syracuse, and why?)**
is_close = zips.str.startswith('0') | zips.str.startswith('1')
# There are a bunch of NaNs, but we're not interested in them right now, so we'll say they're Falseis_far = ~(is_close) & zips.notnull()
- Putting it together.

CHAPTER 8: How to deal with timestamps.

- Parsing Unix timestamps. **.astype(int) pd.to_datetime(popcon['atime'])**

```
# maakt van popcon een nieuwe popcon waar alleen de atimes groter zijn dan '1970-01-01'popcon = popcon[popcon['atime'] > '1970-01-01'] nonlibraries.sort('ctime', ascending=False)[:10]
```

CHAPTER 9: Loading data from SQL databases

- **df.sort().head(), df.sort().tail()**

CHAPTER 10: Pivot Tables.

```
- # set the category
df["Status"] = df['Status'].astype('category')
df['Status'].cat.set_categories(['won', 'pending', 'presented', 'declined'], inplace=True)
- #set the index and the values(with np.sum instead of default mean)
pd.pivot_table(df,index=["Manager","Rep"], values=['Price'], aggfunc=[np.sum])
- # a dictionary can also be passed as the aggfunc.
pd.pivot_table(df,index=["Manager","Status"],columns=["Product"],values=["Quantity","Price"],
aggfunc={"Quantity":len,"Price":np.sum},fill_value=0)
- # and in the dictionary a list can be passed.
table = pd.pivot_table(df,index=["Manager","Status"],columns=["Product"],values=["Quantity","Price"],
aggfunc={"Quantity":len,"Price":[np.sum,np.mean]},fill_value=0)
-  # to find everything where manager is Debra
table.query('Manager == ["Debra Henley"]')
```