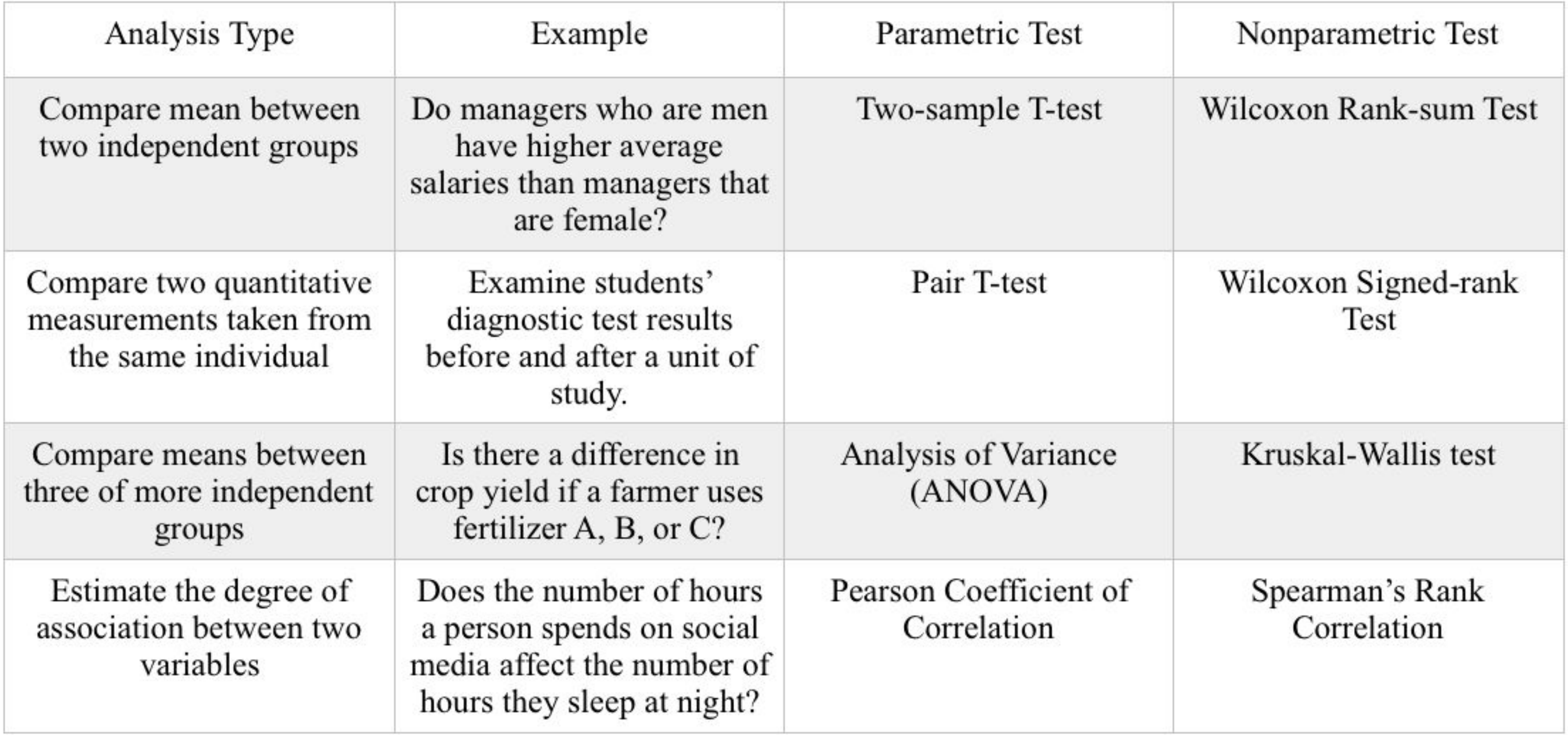
Lecture on Miscellaneous Items

- How to know what model to choose
- How to get data from the USGS

How to Know What Model to Choose

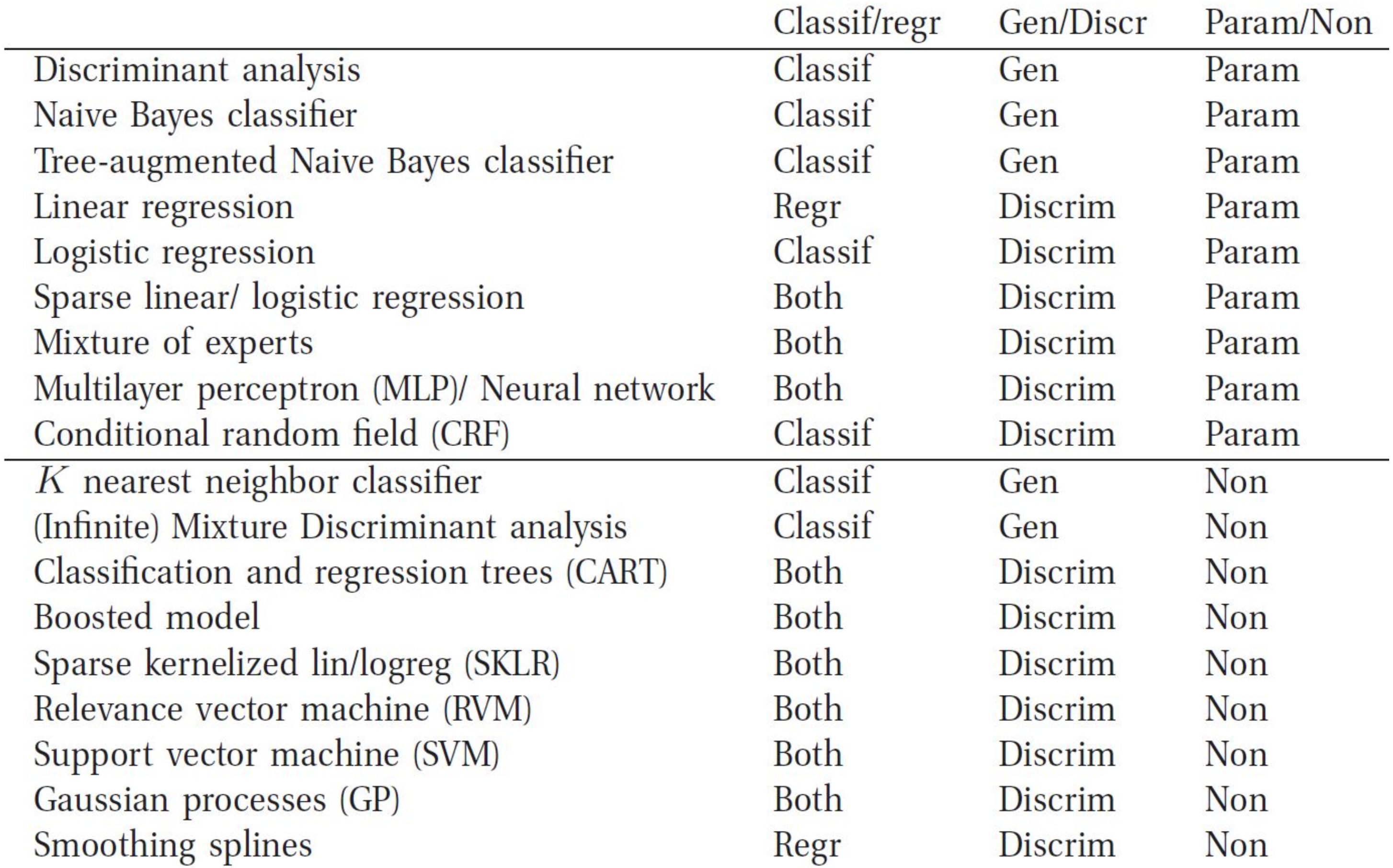
Consider the Characteristics of the Data

- Parametric vs non-parametric



Analysis Type	Example	Parametric Test	Nonparametric Test
Compare mean between two independent groups	Do managers who are men have higher average salaries than managers that are female?	Two-sample T-test	Wilcoxon Rank-sum Test
Compare two quantitative measurements taken from the same individual	Examine students' diagnostic test results before and after a unit of study.	Pair T-test	Wilcoxon Signed-rank Test
Compare means between three of more independent groups	Is there a difference in crop yield if a farmer uses fertilizer A, B, or C?	Analysis of Variance (ANOVA)	Kruskal-Wallis test
Estimate the degree of association between two variables	Does the number of hours a person spends on social media affect the number of hours they sleep at night?	Pearson Coefficient of Correlation	Spearman's Rank Correlation

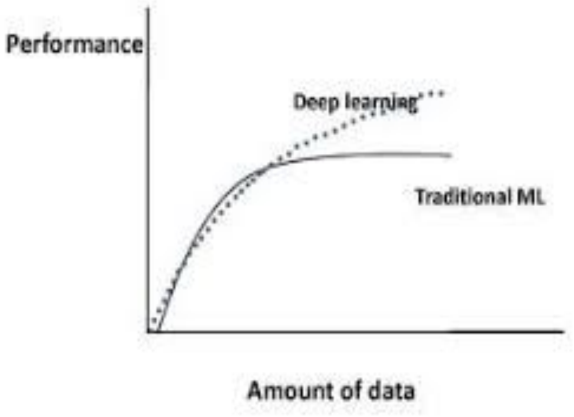
Parametric vs Non-Parametric for Machine Learning

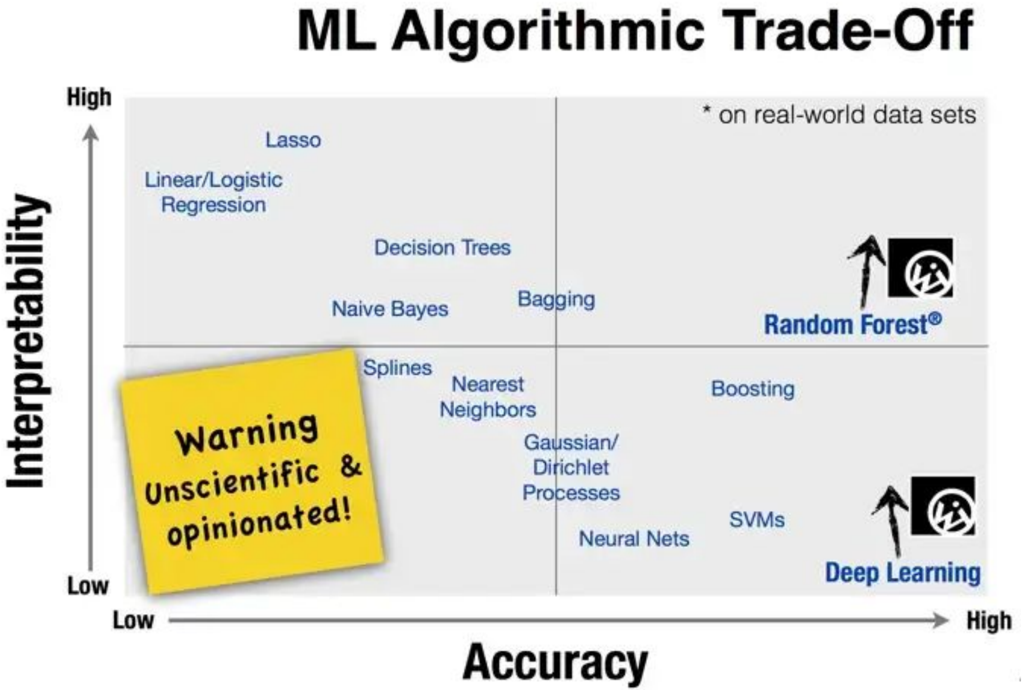


	Classif/regr	Gen/Discr	Param/Non
Discriminant analysis	Classif	Gen	Param
Naive Bayes classifier	Classif	Gen	Param
Tree-augmented Naive Bayes classifier	Classif	Gen	Param
Linear regression	Regr	Discrim	Param
Logistic regression	Classif	Discrim	Param
Sparse linear/ logistic regression	Both	Discrim	Param
Mixture of experts	Both	Discrim	Param
Multilayer perceptron (MLP)/ Neural network	Both	Discrim	Param
Conditional random field (CRF)	Classif	Discrim	Param
K nearest neighbor classifier	Classif	Gen	Non
(Infinite) Mixture Discriminant analysis	Classif	Gen	Non
Classification and regression trees (CART)	Both	Discrim	Non
Boosted model	Both	Discrim	Non
Sparse kernelized lin/logreg (SKLR)	Both	Discrim	Non
Relevance vector machine (RVM)	Both	Discrim	Non
Support vector machine (SVM)	Both	Discrim	Non
Gaussian processes (GP)	Both	Discrim	Non
Smoothing splines	Regr	Discrim	Non

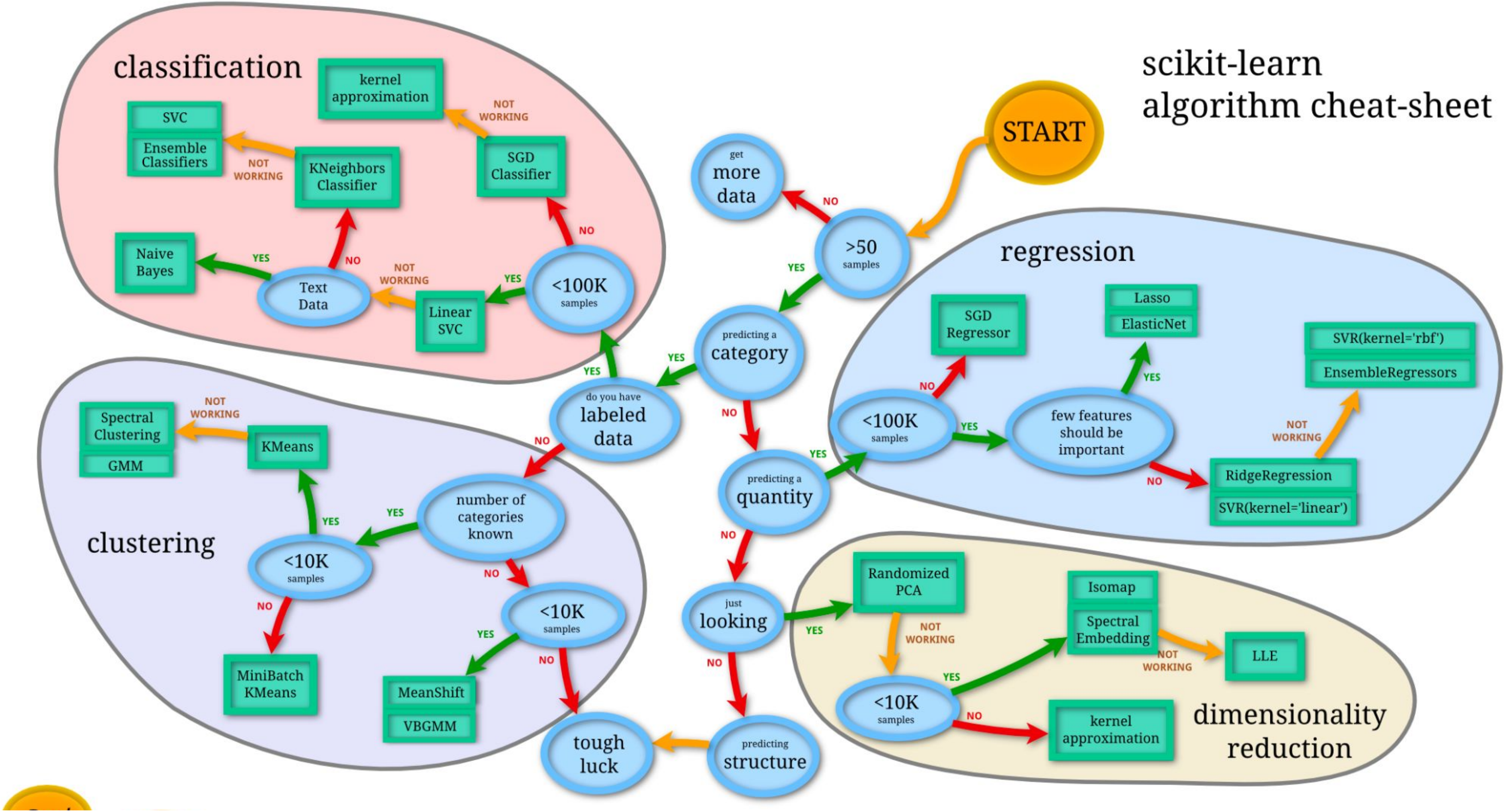
Consider the Characteristics of the Data

- Size of the dataset, goal of analysis
- https://www.kaggle.com/discussions/general/337753





https://medium.com/hackernoon/choosing-the-right-machine-learning-algorithm-68126944ce1f


scikit-learn
algorithm cheat-sheet
START
>50 samples
NO
get more data
YES
predicting a category
YES
do you have labeled data
YES
<100K samples
YES
Linear SVC
NOT WORKING
Text Data
YES
Naive Bayes
NO
KNeighbors Classifier
NOT WORKING
SVC
Ensemble Classifiers
NO
SGD Classifier
NOT WORKING
kernel approximation
classification
NO
number of categories known
YES
<10K samples
YES
KMeans
NOT WORKING
Spectral Clustering
GMM
NO
MiniBatch KMeans
NO
<10K samples
YES
MeanShift
VBGMM
NO
tough luck
clustering
NO
predicting a quantity
YES
<100K samples
NO
SGD Regressor
YES
few features should be important
YES
Lasso
ElasticNet
NO
RidgeRegression
SVR(kernel='linear')
NOT WORKING
SVR(kernel='rbf')
EnsembleRegressors
regression
NO
just looking
YES
Randomized PCA
NOT WORKING
<10K samples
YES
Isomap
Spectral Embedding
NOT WORKING
LLE
NO
kernel approximation
dimensionality reduction
NO
predicting structure
tough luck

Considerations for Popular ML models

	Random Forest	Logistic Regression	Linear SVM	Naive Bayes	Decision Trees	KNN
Regr vs Class	**R or C**	**C**	**C**	**C**	**R or C**	**R or C**
Large vs small dataset	**L**	**S**	**S**	**S or L**	**L**	**L**
Large vs small # of features	**S**	**L**	**L**	**L**	**S**	**S**
Accurate vs interpretable	**A**	**I**	**A**	**A**	**I**	**A**
Fast vs slow training	**S**	**F**	**F**	**F**	**F**	**F**

Regression vs Classification vs Clustering

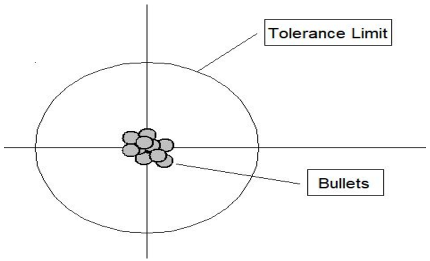
Small vs Large Training Data

Small vs Large Number of Features

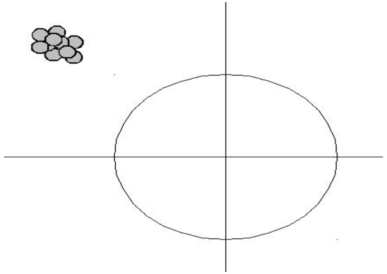
Accurate vs Interpretable Output

Slow vs Fast Training

Tradeoffs

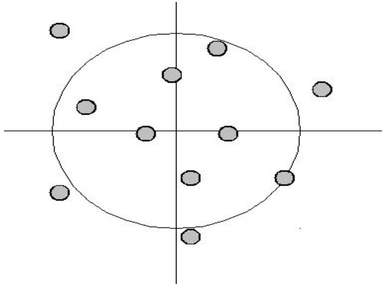


Low Var, Low Bias

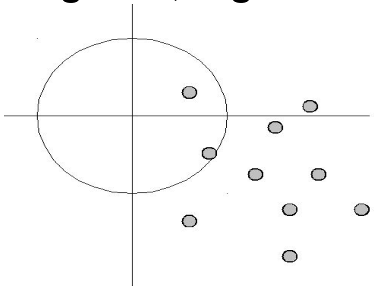


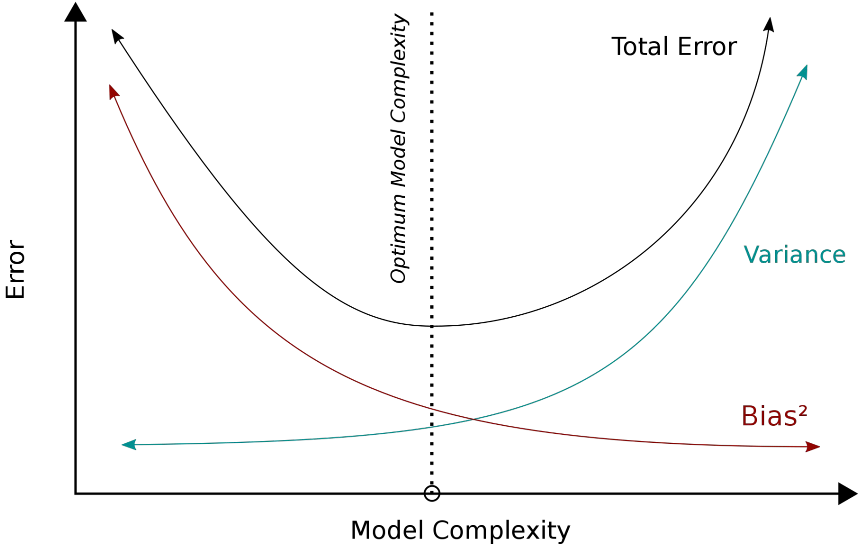
Low Var, High Bias

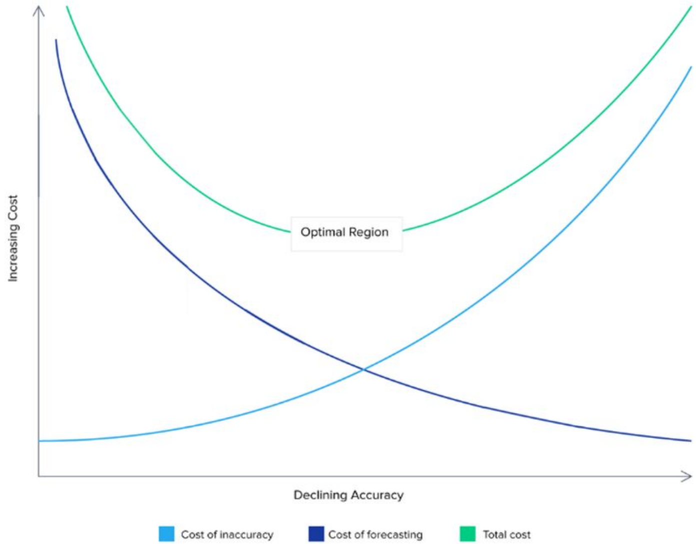
High Var, Low Bias



High Var, High Bias







https://www.advancinganalytics.co.uk/blog/2021/06/22/10-incredibly-useful-time-series-forecasting-algorithms

The Rules Sometimes Need to be Broken

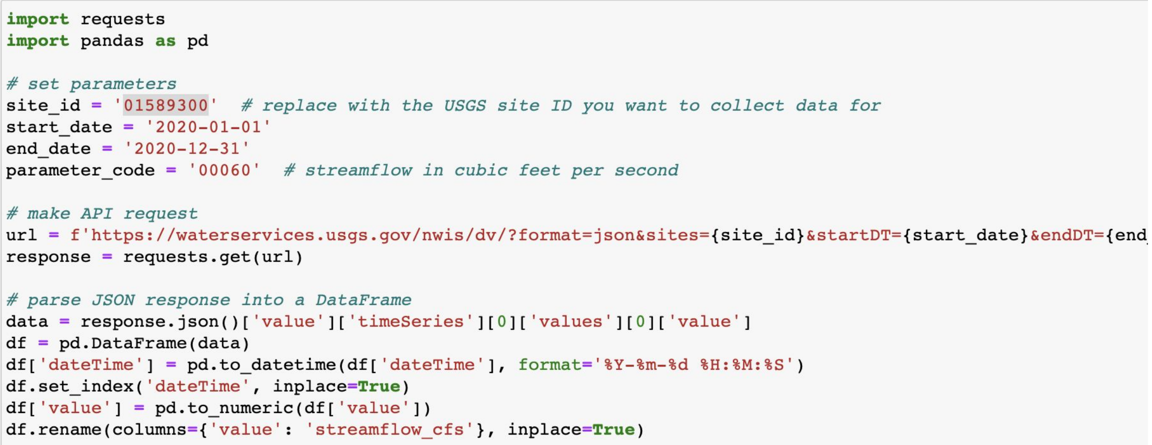
- Always do a literature search for models that have performed well with data similar to your own
- ML algorithm selection rules/roadmaps are just guidelines
- Unless there is an explicit reason not to do so, you should test as many different relevant algorithms as possible
- Every dataset is different, and persistence and experimentation are the keys to getting good results with ML

How to Get Data From the USGS

https://waterservices.usgs.gov/rest/

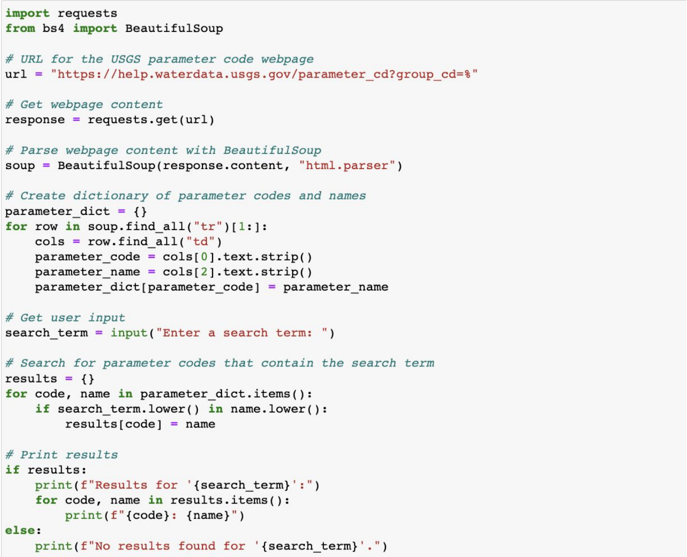
Simple single-site, single-param version

I asked ChatGPT to "write a script to collect data from the USGS NWIS"



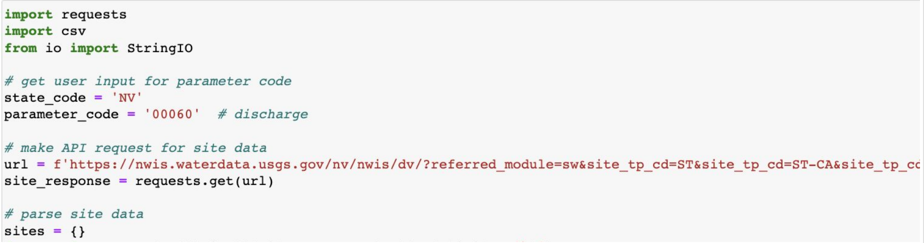
```python
import requests
import pandas as pd

# set parameters
site_id = '01589300'  # replace with the USGS site ID you want to collect data for
start_date = '2020-01-01'
end_date = '2020-12-31'
parameter_code = '00060'  # streamflow in cubic feet per second

# make API request
url = f'https://waterservices.usgs.gov/nwis/dv/?format=json&sites={site_id}&startDT={start_date}&endDT={end
response = requests.get(url)

# parse JSON response into a DataFrame
data = response.json()['value']['timeSeries'][0]['values'][0]['value']
df = pd.DataFrame(data)
df['dateTime'] = pd.to_datetime(df['dateTime'], format='%Y-%m-%d %H:%M:%S')
df.set_index('dateTime', inplace=True)
df['value'] = pd.to_numeric(df['value'])
df.rename(columns={'value': 'streamflow_cfs'}, inplace=True)
```

Search for parameter codes by keyword. You can also use: https://help.waterdata.usgs.gov/parameter_cd?group_cd=%

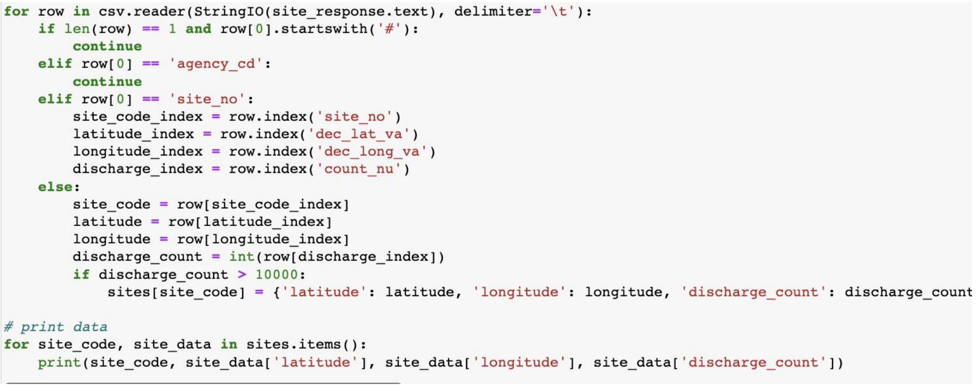


```python
import requests
from bs4 import BeautifulSoup

# URL for the USGS parameter code webpage
url = "https://help.waterdata.usgs.gov/parameter_cd?group_cd=%"

# Get webpage content
response = requests.get(url)

# Parse webpage content with BeautifulSoup
soup = BeautifulSoup(response.content, "html.parser")

# Create dictionary of parameter codes and names
parameter_dict = {}
for row in soup.find_all("tr")[1:]:
    cols = row.find_all("td")
    parameter_code = cols[0].text.strip()
    parameter_name = cols[2].text.strip()
    parameter_dict[parameter_code] = parameter_name

# Get user input
search_term = input("Enter a search term: ")

# Search for parameter codes that contain the search term
results = {}
for code, name in parameter_dict.items():
    if search_term.lower() in name.lower():
        results[code] = name

# Print results
if results:
    print(f"Results for '{search_term}':")
    for code, name in results.items():
        print(f"{code}: {name}")
else:
    print(f"No results found for '{search_term}'.")
```

Find all sites with more than 10,000 values for a particular variable

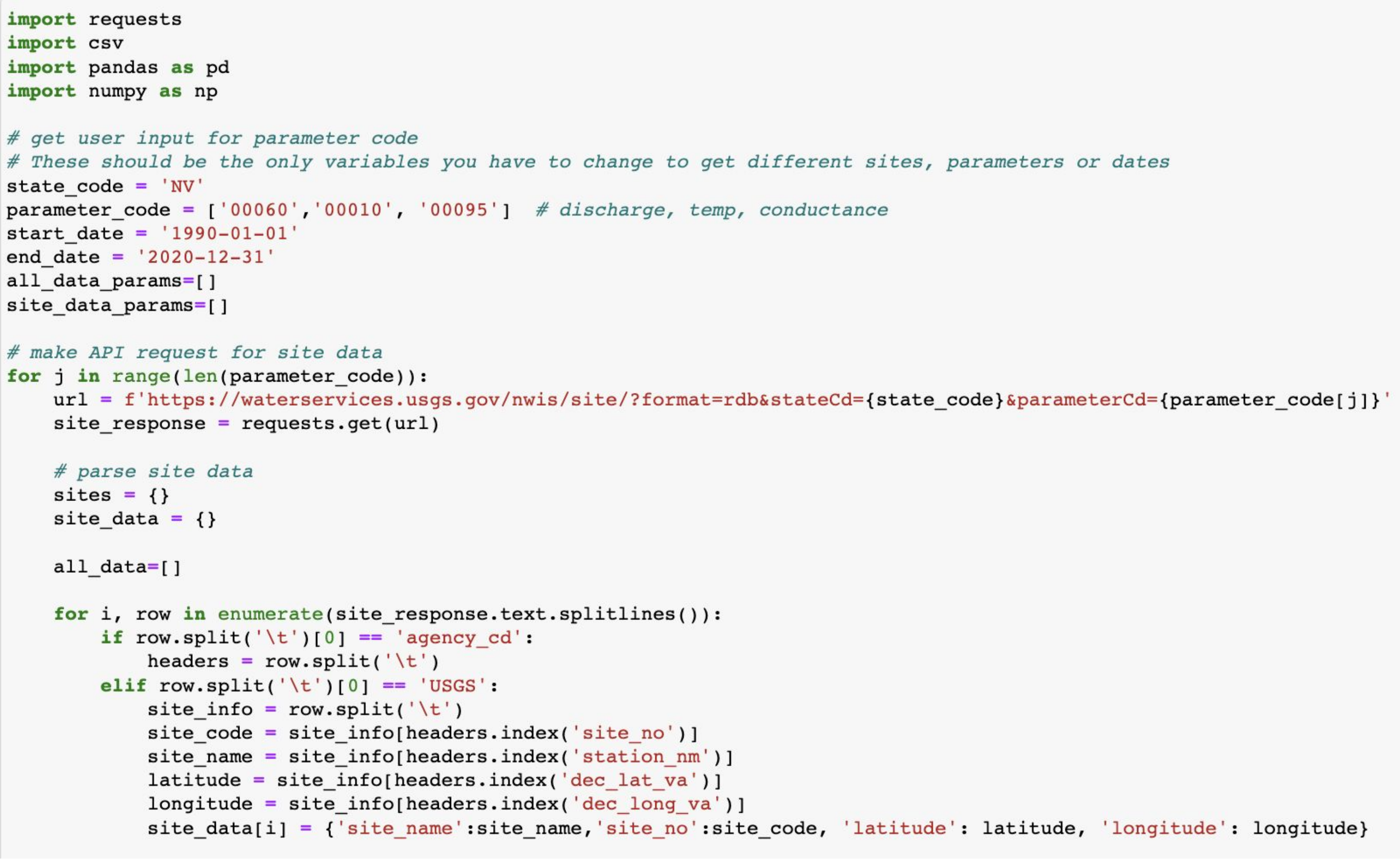


```python
import requests
import csv
from io import StringIO

# get user input for parameter code
state_code = 'NV'
parameter_code = '00060'  # discharge

# make API request for site data
url = f'https://nwis.waterdata.usgs.gov/nv/nwis/dv/?referred_module=sw&site_tp_cd=ST&site_tp_cd=ST-CA&site_tp_cd
site_response = requests.get(url)

# parse site data
sites = {}
```



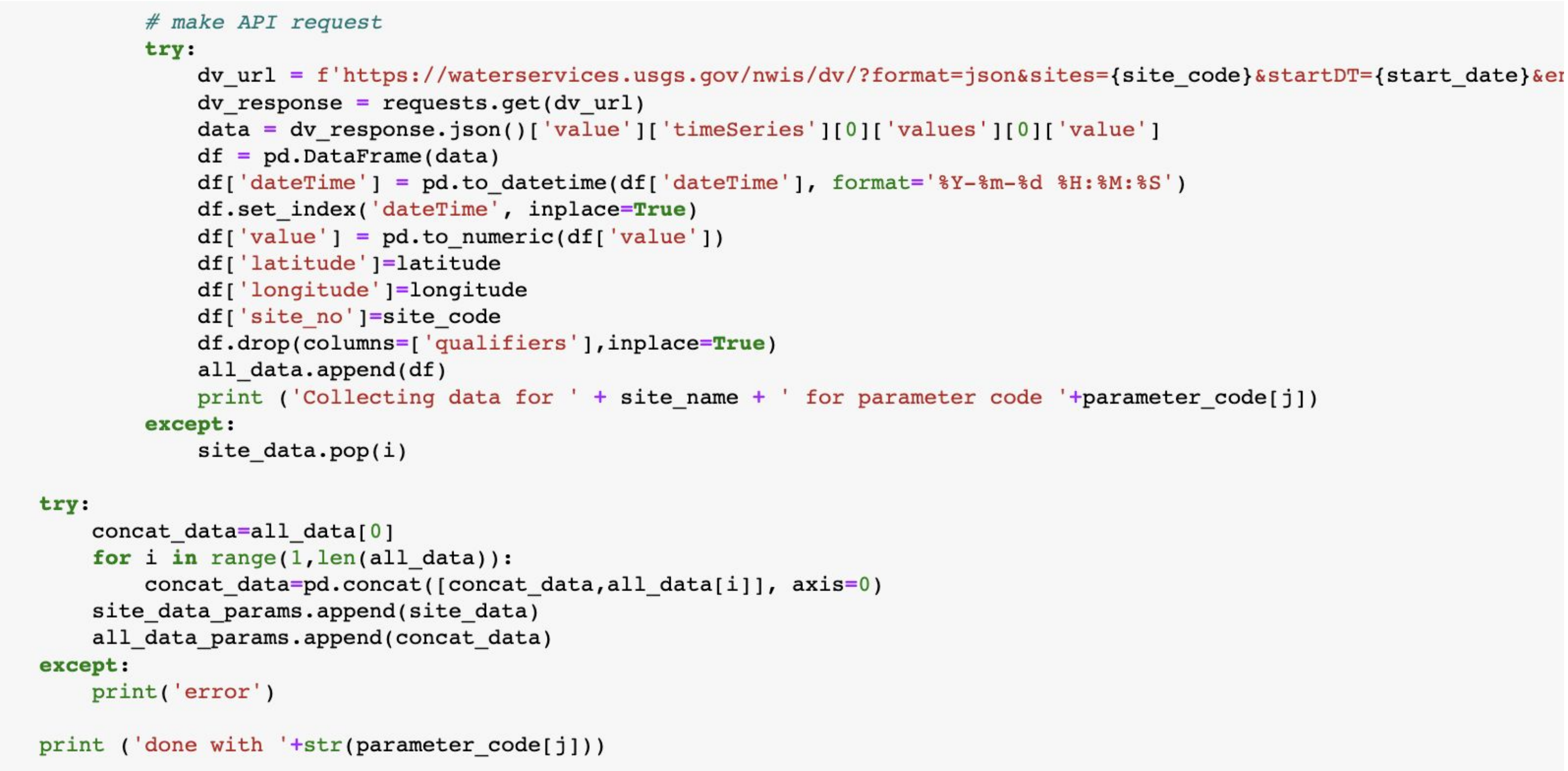
```python
for row in csv.reader(StringIO(site_response.text), delimiter='\t'):
    if len(row) == 1 and row[0].startswith('#'):
        continue
    elif row[0] == 'agency_cd':
        continue
    elif row[0] == 'site_no':
        site_code_index = row.index('site_no')
        latitude_index = row.index('dec_lat_va')
        longitude_index = row.index('dec_long_va')
        discharge_index = row.index('count_nu')
    else:
        site_code = row[site_code_index]
        latitude = row[latitude_index]
        longitude = row[longitude_index]
        discharge_count = int(row[discharge_index])
        if discharge_count > 10000:
            sites[site_code] = {'latitude': latitude, 'longitude': longitude, 'discharge_count': discharge_count

# print data
for site_code, site_data in sites.items():
    print(site_code, site_data['latitude'], site_data['longitude'], site_data['discharge_count'])
```

Multi-site, multi-parameter version



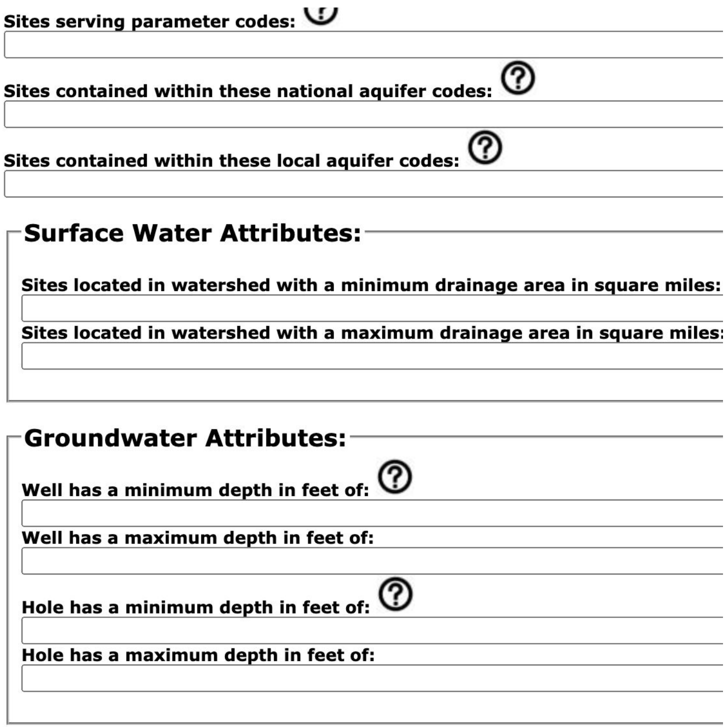
```python
import requests
import csv
import pandas as pd
import numpy as np

# get user input for parameter code
# These should be the only variables you have to change to get different sites, parameters or dates
state_code = 'NV'
parameter_code = ['00060','00010', '00095']  # discharge, temp, conductance
start_date = '1990-01-01'
end_date = '2020-12-31'
all_data_params=[]
site_data_params=[]

# make API request for site data
for j in range(len(parameter_code)):
    url = f'https://waterservices.usgs.gov/nwis/site/?format=rdb&stateCd={state_code}&parameterCd={parameter_code[j]}'
    site_response = requests.get(url)

    # parse site data
    sites = {}
    site_data = {}

    all_data=[]

    for i, row in enumerate(site_response.text.splitlines()):
        if row.split('\t')[0] == 'agency_cd':
            headers = row.split('\t')
        elif row.split('\t')[0] == 'USGS':
            site_info = row.split('\t')
            site_code = site_info[headers.index('site_no')]
            site_name = site_info[headers.index('station_nm')]
            latitude = site_info[headers.index('dec_lat_va')]
            longitude = site_info[headers.index('dec_long_va')]
            site_data[i] = {'site_name':site_name,'site_no':site_code, 'latitude': latitude, 'longitude': longitude}
```

Multi-site, multi-parameter version



```
        # make API request
        try:
            dv_url = f'https://waterservices.usgs.gov/nwis/dv/?format=json&sites={site_code}&startDT={start_date}&en
            dv_response = requests.get(dv_url)
            data = dv_response.json()['value']['timeSeries'][0]['values'][0]['value']
            df = pd.DataFrame(data)
            df['dateTime'] = pd.to_datetime(df['dateTime'], format='%Y-%m-%d %H:%M:%S')
            df.set_index('dateTime', inplace=True)
            df['value'] = pd.to_numeric(df['value'])
            df['latitude']=latitude
            df['longitude']=longitude
            df['site_no']=site_code
            df.drop(columns=['qualifiers'],inplace=True)
            all_data.append(df)
            print ('Collecting data for ' + site_name + ' for parameter code '+parameter_code[j])
        except:
            site_data.pop(i)

try:
    concat_data=all_data[0]
    for i in range(1,len(all_data)):
        concat_data=pd.concat([concat_data,all_data[i]], axis=0)
    site_data_params.append(site_data)
    all_data_params.append(concat_data)
except:
    print('error')

print ('done with '+str(parameter_code[j]))
```

Use More Tags to Filter Search



Sites serving parameter codes:

Sites contained within these national aquifer codes:

Sites contained within these local aquifer codes:

Surface Water Attributes:

Sites located in watershed with a minimum drainage area in square miles:

Sites located in watershed with a maximum drainage area in square miles:

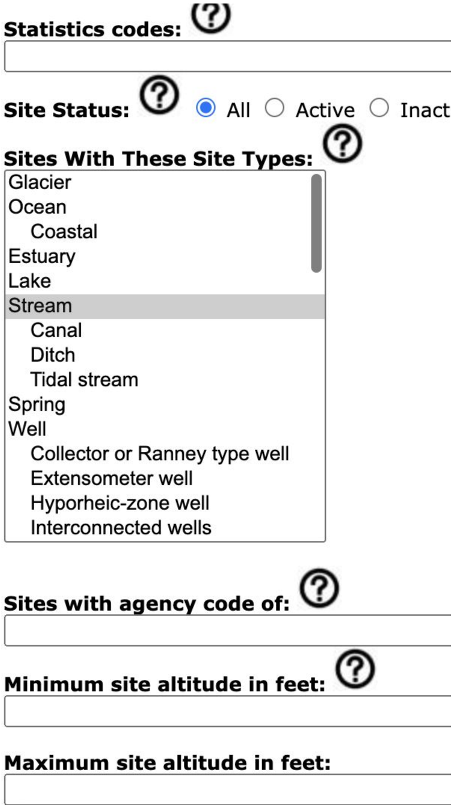
Groundwater Attributes:

Well has a minimum depth in feet of:

Well has a maximum depth in feet of:

Hole has a minimum depth in feet of:

Hole has a maximum depth in feet of:



Statistics codes:

Site Status: ● All ○ Active ○ Inact

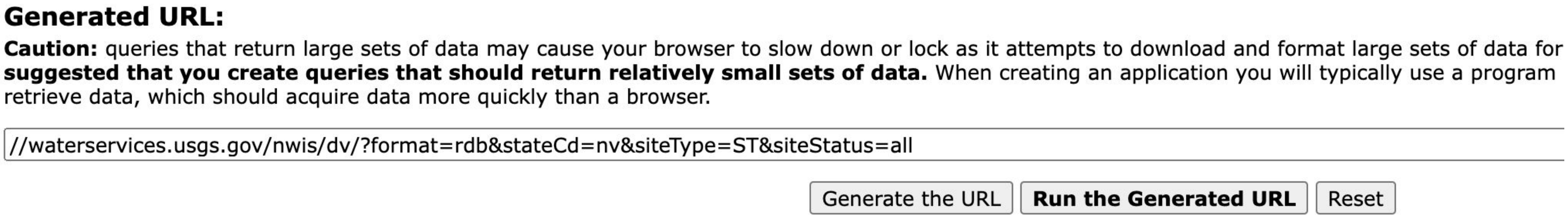
Sites With These Site Types:

- Glacier
- Ocean
  - Coastal
- Estuary
- Lake
- Stream
  - Canal
  - Ditch
  - Tidal stream
- Spring
- Well
  - Collector or Ranney type well
  - Extensometer well
  - Hyporheic-zone well
  - Interconnected wells

Sites with agency code of:

Minimum site altitude in feet:

Maximum site altitude in feet:



Generated URL:

Caution: queries that return large sets of data may cause your browser to slow down or lock as it attempts to download and format large sets of data for **suggested that you create queries that should return relatively small sets of data.** When creating an application you will typically use a program retrieve data, which should acquire data more quickly than a browser.

```
//waterservices.usgs.gov/nwis/dv/?format=rdb&stateCd=nv&siteType=ST&siteStatus=all
```