ELITE DATA SCIENCE

DATASETS FOR DATA SCIENCE AND MACHINE LEARNING

Here are some of our favorite datasets for DIY data science and machine learning projects. They are broken into categories. To see the latest version, plus detailed annotations, visit on the online dataset list at EliteDataScience.com.

TABLE OF CONTENTS

1. EXPLORATORY ANALYSIS

OUR PICKS:	AGGREGATORS:
Game of Thrones	Kaggle Datasets
World University Rankings	r/datasets
IMDB 5000 Movie Dataset	

2. GENERAL MACHINE LEARNING

OUR PICKS:	AGGREGATORS:
Wine Quality (Regression)	UCI Machine Learning Repository
Credit Card Default (Classification)	
US Census Data (Clustering)	

3. DEEP LEARNING

OUR PICKS:	AGGREGATORS:
MNIST	Deeplearning.net
CIFAR	DeepLearning4J.org
ImageNet	
YouTube 8M	

4. NATURAL LANGUAGE PROCESSING

OUR PICKS:	AGGREGATORS:
Enron Dataset	nlp-datasets (Github)
Amazon Reviews	Quora Answer
Newsgroup Classification	

5. CLOUD MACHINE LEARNING

OUR PICKS:
AWS Public Datasets
Google Cloud Public Datasets
Microsoft Azure Public Datasets

DATASETS FOR DATA SCIENCE AND MACHINE LEARNING

ELITE DATA SCIENCE

6. TIME SERIES ANALYSIS

OUR PICKS:
- EOD Stock Prices
- Zillow Real Estate Research
- Global Education Statistics

AGGREGATORS:
- Quandl
- The World Bank

7. RECOMMENDER SYSTEMS

OUR PICKS:
- MovieLens
- Jester
- Million Song Dataset

AGGREGATORS:
- entaroadun (Github)

8. SPECIFIC INDUSTRIES

OUR PICKS:
- Awesome Public Datasets
- Data.gov

9. STREAMING

OUR PICKS:
- Twitter API
- StockTwits API
- Weather Underground

AGGREGATORS:
- Satori

10. WEB SCRAPING

OUR PICKS:
- ToScrape.com

11. CURRENT EVENTS

AGGREGATORS:
- FiveThirtyEight
- BuzzFeedNews

To see the latest version, plus detailed annotations, visit on the online dataset list at EliteDataScience.com.

ELITE DATA SCIENCE

PYTHON CHEATSHEET:
DATA SCIENCE BASICS

In this cheat sheet, we summarize common and useful functionality from Pandas, NumPy, and Scikit-Learn. To see the most up-to-date full version, visit the online cheatsheet at elitedatascience.com.

SETUP

First, make sure you have the following installed on your computer:

- Python 2.7+ or Python 3
- Pandas
- Jupyter Notebook (optional, but recommended)

*note: We strongly recommend installing the Anaconda Distribution, which comes with all of those packages.

IMPORTING DATA

pd.read_csv(filename)
pd.read_table(filename)
pd.read_excel(filename)
pd.read_sql(query, connection_object)
pd.read_json(json_string)
pd.read_html(url)
pd.read_clipboard()
pd.DataFrame(dict)

EXPLORING DATA

df.shape()
df.head(n)
df.tail(n)
df.info()
df.describe()
s.value_counts(dropna=False)
df.apply(pd.Series.value_counts)
df.describe()
df.mean()
df.corr()
df.count()
df.max()
df.min()
df.median()
df.std()

SELECTING

df[col]
df[[col1, col2]]
s.iloc[0]
s.loc[0]
df.iloc[0,:]
df.iloc[0,0]

DATA CLEANING

df.columns = ['a','b','c']
pd.isnull()
pd.notnull()
df.dropna()
df.dropna(axis=1)
df.dropna(axis=1,thresh=n)
df.fillna(x)
s.fillna(s.mean())
s.astype(float)
s.replace(1,'one')
s.replace([1,3],['one','three'])
df.rename(columns=lambda x: x + 1)
df.rename(columns={'old_name': 'new_ name'})
df.set_index('column_one')
df.rename(index=lambda x: x + 1)

FILTER, SORT AND GROUP BY

df[df[col] > 0.5]
df[(df[col] > 0.5) & (df[col] < 0.7)]
df.sort_values(col1)
df.sort_values(col2,ascending=False)
df.sort_values([col1,col2], ascending=[True,False])
df.groupby(col)
df.groupby([col1,col2])
df.groupby(col1)[col2].mean()
df.pivot_table(index=col1, values= col2,col3], aggfunc=mean)
df.groupby(col1).agg(np.mean)
df.apply(np.mean)
df.apply(np.max, axis=1)

JOINING AND COMBINING

df1.append(df2)
pd.concat([df1, df2],axis=1)
df1.join(df2,on=col1,how='inner')

WRITING DATA

df.to_csv(filename)
df.to_excel(filename)
df.to_sql(table_name, connection_object)
df.to_json(filename)
df.to_html(filename)
df.to_clipboard()

ELITE DATA SCIENCE

SCIKIT-LEARN CHEATSHEET: PYTHON MACHINE LEARNING TUTORIAL

In this step-by-step Python machine learning cheatsheet, you'll learn how to use Scikit-Learn to build and tune a supervised learning model!

Scikit-Learn, also known as sklearn, is Python's premier general-purpose machine learning library. While you'll find other packages that do better at certain tasks, Scikit-Learn's versatility makes it the best starting place for most ML problems.

To see the most up-to-date full tutorial, as well as installation instructions, visit the online tutorial at elitedatascience.com.

SETUP

Make sure the following are installed on your computer:

- Python 2.7+ or Python 3
- NumPy
- Pandas
- Scikit-Learn (a.k.a. sklearn)

*We strongly recommend installing Python through Anaconda (installation guide). It comes with all of the above packages already installed.

IMPORT LIBRARIES AND MODULES

```
import numpy as np
import pandas as pd

from sklearn.model_selection import train_test_split
from sklearn import preprocessing
from sklearn.ensemble import RandomForestRegressor
from sklearn.pipeline import make_pipeline
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import mean_squared_error, r2_score
from sklearn.externals import joblib
```

LOAD RED WINE DATA

```
dataset_url = 'http://mlr.cs.umass.edu/ml/machine-learning-databas
          es/wine-quality/winequality-red.csv'
data = pd.read_csv(dataset_url, sep=';')
```

SPLIT DATA INTO TRAINING AND TEST SETS

```
y = data.quality
X = data.drop('quality', axis=1)
X_train, X_test, y_train, y_test = train_test_split(X, y,
                                                    test_size=0.2,
                                                    random_state=123,
                                                    stratify=y)
```

DECLARE DATA PREPROCESSING STEPS

```
pipeline = make_pipeline(preprocessing.StandardScaler(),
                         RandomForestRegressor(n_estimators=100))
```

DECLARE HYPERPARAMETERS TO TUNE

```
hyperparameters = { 'randomforestregressor__max_features' : ['auto',
                    'sqrt', 'log2'],
                    'randomforestregressor__max_depth':
                    [None, 5, 3, 1]}
```

TUNE MODEL USING CROSS-VALIDATION PIPELINE

```
clf = GridSearchCV(pipeline, hyperparameters, cv=10)

clf.fit(X_train, y_train)
```

REFIT ON THE ENTIRE TRAINING SET

```
# No additional code needed if clf.refit == True (default is True)
```

EVALUATE MODEL PIPELINE ON TEST DATA

```
pred = clf.predict(X_test)
print r2_score(y_test, pred)
print mean_squared_error(y_test, pred)
```

SAVE MODEL FOR FUTURE USE

```
joblib.dump(clf, 'rf_regressor.pkl')
# To load: clf2 = joblib.load('rf_regressor.pkl')
```

To see the most up-to-date full tutorial, explanations, and additional context, visit the online tutorial at elitedatascience.com. We also have plenty of other tutorials and guides.

ELITE DATA SCIENCE

CHECKLIST:

FEATURE ENGINEERING IDEAS

Follow me on LinkedIn for more:
Steve Nouri
https://www.linkedin.com/in/stevenouri/

Feature engineering, the process creating new input features for machine learning, is one of the most effective ways to improve predictive models. Check out the most up-to-date full guide here.

INDICATOR VARIABLES

- **Indicator variable from thresholds:** Let's say you're studying alcohol preferences by U.S. consumers and your dataset has an `age` feature. You can create an indicator variable for `age >= 21` to distinguish subjects who were over the legal drinking age.
- **Indicator variable from multiple features:** You're predicting real-estate prices and you have the features `n_bedrooms` and `n_bathrooms`. If houses with 2 beds and 2 baths command a premium as rental properties, you can create an indicator variable to flag them.
- **Indicator variable for special events:** You're modeling weekly sales for an e-commerce site. You can create two indicator variables for the weeks of Black Friday and Christmas.
- **Indicator variable for groups of classes:** You're analyzing website conversions and your dataset has the categorical feature `traffic_source`. You could create an indicator variable for `paid_traffic` by flagging observations with traffic source values of `"Facebook Ads"` or `"Google Adwords"`.

INTERACTION FEATURES

- **Sum of two features:** Let's say you wish to predict revenue based on preliminary sales data. You have the features `sales_blue_pens` and `sales_black_pens`. You could sum those features if you only care about overall `sales_pens`.
- **Difference between two features:** You have the features `house_built_date` and `house_purchase_date`. You can take their difference to create the feature `house_age_at_purchase`.
- **Product of two features:** You're running a pricing test, and you have the feature `price` and an indicator variable `conversion`. You can take their product to create the feature `earnings`.
- **Quotient of two features:** You have a dataset of marketing campaigns with the features `n_clicks` and `n_impressions`. You can divide clicks by impressions to create `click_through_rate`, allowing you to compare across campaigns of different volume.

FEATURE REPRESENTATION

- **Date and time features:** Let's say you have the feature `purchase_datetime`. It might be more useful to extract `purchase_day_of_week` and `purchase_hour_of_day`. You can also aggregate observations to create features such as `purchases_over_last_30_days`.
- **Numeric to categorical mappings:** You have the feature `years_in_school`. You might create a new feature `grade` with classes such as `"Elementary School"`, `"Middle School"`, and `"High School"`.
- **Grouping sparse classes:** You have a feature with many classes that have low sample counts. You can try grouping similar classes and then grouping the remaining ones into a single `"Other"` class.
- **Creating dummy variables:** Depending on your machine learning implementation, you may need to manually transform categorical features into dummy variables. You should always do this *after* grouping sparse classes.

EXTERNAL DATA

- **Time series data:** The nice thing about time series data is that you only need one feature, some form of `date`, to layer in features from another dataset.
- **External API's:** There are plenty of API's that can help you create features. For example, the Microsoft Computer Vision API can return the number of faces from an image.
- **Geocoding:** Let's say have you `street_address`, `city`, and `state`. Well, you can geocode them into `latitude` and `longitude`. This will allow you to calculate features such as local demographics (e.g. `median_income_within_2_miles`) with the help of another dataset.
- **Other sources of the same data:** How many ways could you track a Facebook ad campaign? You might have Facebook's own tracking pixel, Google Analytics, and possibly another third-party software. Each source can provide information that the others don't track. Plus, any differences between the datasets could be informative (e.g. bot traffic that one source ignores while another source keeps).

ERROR ANALYSIS (POST-MODELING)

- **Start with larger errors:** Error analysis is typically a manual process. You won't have time to scrutinize every observation. We recommend starting with those that had higher error scores. Look for patterns that you can formalize into new features.
- **Segment by classes:** Another technique is to segment your observations and compare the average error within each segment. You can try creating indicator variables for the segments with the highest errors.
- **Unsupervised clustering:** If you have trouble spotting patterns, you can run an unsupervised clustering algorithm on the misclassified observations. We don't recommend blindly using those clusters as a new feature, but they can make it easier to spot patterns. Remember, the goal is to understand why observations were misclassified.
- **Ask colleagues or domain experts:** This is a great complement to any of the other three techniques. Asking a domain expert is especially useful if you've identified a pattern of poor performance (e.g. through segmentations) but don't yet understand why.

ELITE DATA SCIENCE

PYTHON CHEATSHEET: HANDLING IMBALANCED CLASSES

This Python cheatsheet will cover some of the most useful methods for handling machine learning datasets that have a disproportionate ratio of observations in each class. These "imbalanced" classes render standard accuracy metrics useless.

To see the most up-to-date full tutorial and download the sample dataset, visit the online tutorial at elitedatascience.com.

SETUP

Make sure the following are installed on your computer:

- Python 2.7+ or Python 3
- NumPy
- Pandas
- Scikit-Learn (a.k.a. sklearn)

LOAD SAMPLE DATASET

```
import pandas as pd
import numpy as np

df = pd.read_csv('balance-scale.data',
                 names=['balance', 'var1', 'var2', 'var3', 'var4'])
```

*Up-to-date link to the sample dataset can be found here.

UP-SAMPLE MINORITY CLASS

```
df_majority = df[df.balance==0]
df_minority = df[df.balance==1]

df_minority_upsampled = resample(df_minority,
                                 replace=False,
                                 n_samples=49,
                                 random_state=123)

df_upsampled = pd.concat([df_majority, df_minority_upsampled])
```

DOWN-SAMPLE MAJORITY CLASS

```
df_majority = df[df.balance==0]
df_minority = df[df.balance==1]

df_majority_downsampled = resample(df_majority,
                                   replace=False,
                                   n_samples=49,
                                   random_state=123)

df_downsampled = pd.concat([df_majority_downsampled, df_minority])
```

CHANGE YOUR PERFORMANCE METRIC

```
from sklearn.metrics import roc_auc_score

prob_y_2 = clf_2.predict_proba(X)
prob_y_2 = [p[1] for p in prob_y_2]
print( roc_auc_score(y, prob_y_2) )
```

USE COST-SENSITIVE ALGORITHMS

```
from sklearn.svm import SVC
clf = SVC(kernel='linear', class_weight='balanced', probability=True)
```

USE TREE-BASED ALGORITHMS

```
from sklearn.ensemble import RandomForestClassifier
clf = RandomForestClassifier()
```

Honorable Mentions

- Create Synthetic Samples (Data Augmentation) - A close cousin of upsampling.
- Combine Minority Classes - Group together similar classes.
- Reframe as Anomaly Detection - Treat minority classes as outliers.

To see the most up-to-date full tutorial, explanations, and additional context, visit the online tutorial at elitedatascience.com. We also have plenty of other tutorials and guides.

KERAS CHEATSHEET:

PYTHON DEEP LEARNING TUTORIAL

This cheatsheet will take you step-by-step through training a convolutional neural network in Python using the famous MNIST dataset for handwritten digits classification. Our classifier will boast over 99% accuracy.

Keras is our recommended library for deep learning in Python, especially for beginners. Its minimalist, modular approach makes it a breeze to get deep neural networks up and running.

To see the most up-to-date full tutorial, as well as installation instructions, visit the online tutorial at elitedatascience.com.

SETUP

Make sure you have the following installed on your computer:

- Python 2.7+ or Python 3
- SciPy with NumPy
- Matplotlib (Optional, recommended for exploratory analysis)
- Theano*

*note: TensorFlow is also supported (as an alternative to Theano), but we stick with Theano to keep it simple. The main difference is that you'll need to reshape the data slightly differently before feeding it to your network.

IMPORT LIBRARIES AND MODULES

```
import numpy as np
np.random.seed(123)  # for reproducibility

from keras.models import Sequential
from keras.layers import Dense, Dropout, Activation, Flatten
from keras.layers import Convolution2D, MaxPooling2D
from keras.utils import np_utils
from keras.datasets import mnist
```

LOAD PRE-SHUFFLED MNIST DATA INTO TRAIN AND TEST SETS

```
(X_train, y_train), (X_test, y_test) = mnist.load_data()
```

PREPROCESS INPUT DATA

```
X_train = X_train.reshape(X_train.shape[0], 1, 28, 28)
X_test = X_test.reshape(X_test.shape[0], 1, 28, 28)
X_train = X_train.astype('float32')
X_test = X_test.astype('float32')
X_train /= 255
X_test /= 255
```

PREPROCESS CLASS LABELS

```
Y_train = np_utils.to_categorical(y_train, 10)
Y_test = np_utils.to_categorical(y_test, 10)
```

DEFINE MODEL ARCHITECTURE

```
model = Sequential()

model.add(Convolution2D(32, 3, 3, activation='relu',
          input_shape=(1,28,28)))
model.add(Convolution2D(32, 3, 3, activation='relu'))
model.add(MaxPooling2D(pool_size=(2,2)))
model.add(Dropout(0.25))

model.add(Flatten())
model.add(Dense(128, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(10, activation='softmax'))
```

COMPILE MODEL

```
model.compile(loss='categorical_crossentropy',
              optimizer='adam',
              metrics=['accuracy'])
```

FIT MODEL ON TRAINING DATA

```
model.fit(X_train, Y_train,
          batch_size=32, nb_epoch=10, verbose=1)
```

EVALUATE MODEL ON TEST DATA

```
score = model.evaluate(X_test, Y_test, verbose=0)
```

To see the most up-to-date full tutorial, explanations, and additional context, visit the online tutorial at elitedatascience.com. We also have plenty of other tutorials and guides.

ELITE DATA SCIENCE

SEABORN CHEATSHEET:

PYTHON DATA VIZ TUTORIAL

This Seaborn cheatsheet covers common and useful functions for creating charts and statistical plots in Python. To see the full gallery of what's possible, visit the online version at elitedatascience.com.

SETUP

First, make sure you have the following installed on your computer:

- Python 2.7+ or Python 3
- Pandas
- Matplotlib
- Seaborn
- Jupyter Notebook (optional, but recommended)

*note: We strongly recommend installing the Anaconda Distribution, which comes with all of those packages.

IMPORT LIBRARIES AND DATASET

```
import pandas as pd
from matplotlib import pyplot as plt
%matplotlib inline
import seaborn as sns
df = pd.read_csv('Pokemon.csv', index_col=0)
```

*Up-to-date link to the sample dataset can be found here.

SCATTERPLOT

```
sns.lmplot(x='Attack', y='Defense', data=df)
```

ADJUSTING AXES LIMITS

```
sns.lmplot(x='Attack', y='Defense', data=df)
plt.ylim(0, None)
plt.xlim(0, None)
```

PREPROCESS W/ PANDAS + BOXPLOT

```
stats_df = df.drop(['Total', 'Stage', 'Legendary'], axis=1)
sns.boxplot(data=stats_df)
```

SET THEME + VIOLINPLOT

```
sns.set_style('whitegrid')
sns.violinplot(x='Type 1', y='Attack', data=df)
```

SET CUSTOM COLOR PALETTE

```
pkmn_type_colors = ['#78C850', '#F08030', '#6890F0', '#A8B820',
                    '#A8A878', '#A040A0', '#F8D030', '#E0C068'
                    '#EE99AC', '#C03028', '#F85888', '#B8A038',
                    '#705898', '#98D8D8', '#7038F8']
sns.violinplot(x='Type 1', y='Attack', data=df,
               palette=pkmn_type_colors)
```

OVERLAYING PLOTS

```
plt.figure(figsize=(10,6))
sns.violinplot(x='Type 1', y='Attack', data=df,
               inner=None, palette=pkmn_type_colors)
sns.swarmplot(x='Type 1',
              y='Attack',
              data=df,
              color='k',
              alpha=0.7)
plt.title('Attack by Type')
```

PUTTING IT ALL TOGETHER

```
stats_df.head()
melted_df = pd.melt(stats_df,
                    id_vars=["Name", "Type 1", "Type 2"],
                    var_name="Stat")
sns.swarmplot(x='Stat', y='value', data=melted_df, hue='Type 1')
plt.figure(figsize=(10,6))
sns.swarmplot(x='Stat', y='value', data=melted_df,
              hue='Type 1', split=True, palette=pkmn_type_colors)
plt.ylim(0, 260)
plt.legend(bbox_to_anchor=(1, 1), loc=2
```

OTHER PLOT TYPES

```
corr = stats_df.corr()
sns.heatmap(corr)

sns.distplot(df.Attack)
sns.countplot(x='Type 1', data=df, palette=pkmn_type_colors)
plt.xticks(rotation=-45)

g = sns.factorplot(x='Type 1', y='Attack', data=df,
                   hue='Stage', col='Stage', kind='swarm')
g.set_xticklabels(rotation=-45)

sns.kdeplot(df.Attack, df.Defense)
sns.jointplot(x='Attack', y='Defense', data=df
```

ELITE DATA SCIENCE

PANDAS CHEATSHEET:

PYTHON DATA WRANGLING TUTORIAL

This Pandas cheatsheet will cover some of the most common and useful functionalities for data wrangling in Python. Broadly speaking, data wrangling is the process of reshaping, aggregating, separating, or otherwise transforming your data from one format to a more useful one.

Pandas is the best Python library for wrangling relational (i.e. table-format) datasets, and it will be doing most of the heavy lifting for us.

To see the most up-to-date full tutorial and download the sample dataset, visit the online tutorial at elitedatascience.com.

SETUP

First, make sure you have the following installed on your computer:

- Python 2.7+ or Python 3
- Pandas
- Jupyter Notebook (optional, but recommended)

*note: We strongly recommend installing the Anaconda Distribution, which comes with all of those packages. Simply follow the instructions on that download page.

Once you have Anaconda installed, simply start Jupyter (either through the command line or the Navigator app) and open a new notebook.

IMPORT LIBRARIES AND DATASET

```
import pandas as pd
pd.options.display.float_format = '{:,.2f}'.format
pd.options.display.max_rows = 200
pd.options.display.max_columns = 100

df = pd.read_csv('BNC2_sample.csv',
                 names=['Code', 'Date', 'Open', 'High', 'Low'
                        'Close', 'Volume', 'VWAP', 'TWAP'])
```

*The sample dataset can be downloaded here.

FILTER UNWANTED OBSERVATIONS

```
gwa_codes = [code for code in df.Code.unique() if 'GWA_' in code]
df = df[df.Code.isin(gwa_codes)]
```

PIVOT THE DATASET

```
pivoted_df = df.pivot(index='Date', columns='Code', values='VWAP')
```

SHIFT THE PIVOTED DATASET

```
delta_dict = {}
for offset in [7, 14, 21, 28]:
delta_dict['delta_{}'.format(offset)] = pivoted_df /
                                        pivoted_df.shift(offset) - 1
```

MELT THE SHIFTED DATASET

```
melted_dfs = []
for key, delta_df in delta_dict.items():
    melted_dfs.append( delta_df.reset_index().melt(id_vars=['Date'],
                       value_name=key) )

return_df = pivoted_df.shift(-7) / pivoted_df - 1.0
melted_dfs.append( return_df.reset_index().melt(id_vars=['Date'],
                   value_name='return_7') )
```

REDUCE-MERGE THE MELTED DATA

```
from functools import reduce

base_df = df[['Date', 'Code', 'Volume', 'VWAP']]
feature_dfs = [base_df] + melted_dfs

abt = reduce(lambda left,right: pd.merge(left,right,on=['Date',
                                          'Code']), feature_dfs)
```

AGGREGATE WITH GROUP-BY

```
abt['month'] = abt.Date.apply(lambda x: x[:7])
gb_df = abt.groupby(['Code', 'month']).first().reset_index()
```

To see the most up-to-date full tutorial, explanations, and additional context, visit the online tutorial at elitedatascience.com. We also have plenty of other tutorials and guides.

Follow me on LinkedIn for more:
Steve Nouri
https://www.linkedin.com/in/stevenouri/