**Importing Data**

pd.read\_csv(filename) # From a CSV file

pd.read\_table(filename) # From a delimited text file (like TSV)

pd.read\_excel(filename) # From an Excel file

pd.read\_sql(query, connection\_object) # Reads from a SQL table/database

pd.read\_json(json\_string) # Reads from a JSON formatted string, URL or file.

pd.read\_html(url) # Parses an html URL, string or file and extracts tables to a list of dataframes

pd.read\_clipboard() # Takes the contents of your clipboard and passes it to read\_table()

pd.DataFrame(dict) # From a dict, keys for columns names, values for data as lists

**Exploring Data**

df.shape() # Prints number of rows and columns in dataframe

df.head(n) # Prints first n rows of the DataFrame

df.tail(n) # Prints last n rows of the DataFrame

df.info() # Index, Datatype and Memory information

df.describe() # Summary statistics for numerical columns

s.value\_counts(dropna=False) # Views unique values and counts

df.apply(pd.Series.value\_counts) # Unique values and counts for all columns

df.describe() # Summary statistics for numerical columns

df.mean() # Returns the mean of all columns

df.corr() # Returns the correlation between columns in a DataFrame

df.count() # Returns the number of non-null values in each DataFrame column

df.max() # Returns the highest value in each column

df.min() # Returns the lowest value in each column

df.median() # Returns the median of each column

df.std() # Returns the standard deviation of each column

**Selecting data**

df[col] # Returns column with label col as Series

df[[col1, col2]] # Returns Columns as a new DataFrame

s.iloc[0] # Selection by position (selects first element)

s.loc[0] # Selection by index (selects element at index 0)

df.iloc[0,:] # First row

df.iloc[0,0] # First element of first column

**Data Cleaning**

df.columns = ['a','b','c'] # Renames columns

pd.isnull() # Checks for null Values, Returns Boolean Array

pd.notnull() # Opposite of s.isnull()

df.dropna() # Drops all rows that contain null values

df.dropna(axis=1) # Drops all columns that contain null values

df.dropna(axis=1,thresh=n) # Drops all rows have have less than n non null values

df.fillna(x) # Replaces all null values with x

s.fillna(s.mean()) # Replaces all null values with the mean (mean can be replaced with almost any function from the statistics section)

s.astype(float) # Converts the datatype of the series to float

s.replace(1,'one') # Replaces all values equal to 1 with 'one'

s.replace([1,3],['one','three']) # Replaces all 1 with 'one' and 3 with 'three'

df.rename(columns=lambda x: x + 1) # Mass renaming of columns

df.rename(columns={'old\_name': 'new\_ name'}) # Selective renaming

df.set\_index('column\_one') # Changes the index

df.rename(index=lambda x: x + 1) # Mass renaming of index

df.drop\_duplicates(subset ="First Name", keep = False, inplace = True) #drop duplicated rows based on columns provided in subset if not all the column will be considered. Keep Denotes the occurrence which should be marked as duplicate. It’s value can be {‘first’, ‘last’, False}, False for all

df[df.duplicated(['column\_one'])] #all duplicated rows based on one column

**Filter, Sort and Group By**

df[df[col] > 0.5] # Rows where the col column is greater than 0.5

df[(df[col] > 0.5) & (df[col] < 0.7)] # Rows where 0.5 < col < 0.7

df.sort\_values(col1) # Sorts values by col1 in ascending order

df.sort\_values(col2,ascending=False) # Sorts values by col2 in descending order

df.sort\_values([col1,col2], ascending=[True,False]) # Sorts values by col1 in ascending order then col2 in descending order

df.groupby(col) # Returns a groupby object for values from one column

df.groupby([col1, col2]) # Returns a groupby object values from multiple columns

df.groupby(col1)[col2].mean() # Returns the mean of the values in col2, grouped by the values in col1 (mean can be replaced with almost any function from the statistics section)

df.groupby([col1, col2], as\_index= False)[col3].mean() # returns aggregated values as table

df.pivot\_table(index=col1, values= col2,col3], aggfunc=mean) # Creates a pivot table that groups by col1 and calculates the mean of col2 and col3

df.groupby(col1).agg(np.mean) # Finds the average across all columns for every unique column 1 group

df.apply(np.mean) # Applies a function across each column

df.apply(np.max, axis=1) # Applies a function across each row

**Joining and Combining**

df1.append(df2) # Adds the rows in df1 to the end of df2 (columns should be identical)

pd.concat([df1, df2],axis=1) # Adds the columns in df1 to the end of df2 (rows should be identical)

df1.join(df2,on=col1,how='inner') # SQL-style joins the columns in df1 with the columns on df2 where the r

**Writing Data**

df.to\_csv(filename) # Writes to a CSV file

df.to\_excel(filename) # Writes to an Excel file

df.to\_sql(table\_name, connection\_object) # Writes to a SQL table

df.to\_json(filename) # Writes to a file in JSON format

df.to\_html(filename) # Saves as an HTML table

df.to\_clipboard() # Writes to the clipboard

**Machine Learning**

# 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-databases/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')