Importing Data

Any kind of data analysis starts with getting hold of some data. Pandas gives you plenty of options for getting data into your Python workbook:

Importing Data
Python

```
d.read_csv(filename) # From a CSV file
d.read_table(filename) # From a delimited text file (like TSV)
d.read_excel(filename) # From an Excel file
d.read_sql(query, connection_object) # Reads from a SQL table/database
d.read_json(json_string) # Reads from a JSON formatted string, URL or file.
d.read_html(url) # Parses an html URL, string or file and extracts tables to a list of dataframes
d.read_clipboard() # Takes the contents of your clipboard and passes it to read_table()
d.DataFrame(dict) # From a dict, keys for columns names, values for data as lists
```

Exploring Data

Once you have imported your data into a Pandas dataframe, you can use these methods to get a sense of what the data looks like:

Exploring Data
Python

```
f.shape() # Prints number of rows and columns in dataframe
f.head(n) # Prints first n rows of the DataFrame
f.tail(n) # Prints last n rows of the DataFrame
f.info() # Index, Datatype and Memory information
f.describe() # Summary statistics for numerical columns
.value_counts(dropna=False) # Views unique values and counts
f.apply(pd.Series.value_counts) # Unique values and counts for all columns
f.describe() # Summary statistics for numerical columns
f.mean() # Returns the mean of all columns
f.corr() # Returns the correlation between columns in a DataFrame
f.count() # Returns the number of non-null values in each DataFrame column
f.max() # Returns the highest value in each column
f.min() # Returns the lowest value in each column
f.median() # Returns the median of each column
f.std() # Returns the standard deviation of each column
```

Selecting

Often, you might need to select a single element or a certain subset of the data to inspect it or perform further analysis. These methods will come in handy:

Selecting Data
Python

```
lf[col] # Returns column with label col as Series
lf[[col1, col2]] # Returns Columns as a new DataFrame
.iloc[0] # Selection by position (selects first element)
.loc[0] # Selection by index (selects element at index 0)
lf.iloc[0,:] # First row
lf.iloc[0,0] # First element of first column
```

Data Cleaning

If you're working with real world data, chances are you'll need to clean it up. These are some helpful methods:

Data Cleaning
Python

```
f.columns = ['a','b','c'] # Renames columns
d.isnull() # Checks for null Values, Returns Boolean Array
d.notnull() # Opposite of s.isnull()
f.dropna() # Drops all rows that contain null values
```

f.dropna(axis=1) # Drops all columns that contain null values
f.dropna(axis=1,thresh=n) # Drops all rows have have less than n non null values
f.fillna(x) # Replaces all null values with x
.fillna(s.mean()) # Replaces all null values with the mean (mean can be replaced with almost any function from the statistics section)
.astype(float) # Converts the datatype of the series to float
.replace(1,'one') # Replaces all values equal to 1 with 'one'
.replace([1,3],['one','three']) # Replaces all 1 with 'one' and 3 with 'three'
f.rename(columns=lambda x: x + 1) # Mass renaming of columns
f.rename(columns={'old_name': 'new_ name'}) # Selective renaming
f.set_index('column_one') # Changes the index
f.rename(index=lambda x: x + 1) # Mass renaming of index

Filter, Sort and Group By

Methods for filtering, sorting and grouping your data:

Filter, Sort, and Group By
Python
f[df[col] > 0.5] # Rows where the col column is greater than 0.5
f[(df[col] > 0.5) & (df[col] < 0.7)] # Rows where 0.5 < col < 0.7
f.sort_values(col1) # Sorts values by col1 in ascending order
f.sort_values(col2,ascending=False) # Sorts values by col2 in descending order
f.sort_values([col1,col2], ascending=[True,False]) # Sorts values by col1 in ascending order then col2 in descending order
f.groupby(col) # Returns a groupby object for values from one column
f.groupby([col1,col2]) # Returns a groupby object values from multiple columns
f.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)
f.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
f.groupby(col1).agg(np.mean) # Finds the average across all columns for every unique column 1 group
f.apply(np.mean) # Applies a function across each column
f.apply(np.max, axis=1) # Applies a function across each row

Joining and Combining

Methods for combining two dataframes:

Joining and Combining
Python
lf1.append(df2) # Adds the rows in df1 to the end of df2 (columns should be identical)
od.concat([df1, df2],axis=1) # Adds the columns in df1 to the end of df2 (rows should be identical)
lf1.join(df2,on=col1,how='inner') # SQL-style joins the columns in df1 with the columns on df2 where the rows for col have identical values. how can be one of 'left', 'right', 'outer',
nner'

Writing Data

And finally, when you have produced results with your analysis, there are several ways you can export your data:

Writing Data
Python
lf.to_csv(filename) # Writes to a CSV file
lf.to_excel(filename) # Writes to an Excel file
lf.to_sql(table_name, connection_object) # Writes to a SQL table
lf.to_json(filename) # Writes to a file in JSON format
lf.to_html(filename) # Saves as an HTML table
lf.to_clipboard() # Writes to the clipboard

Machine Learning

The Scikit-Learn library contains useful methods for training and applying machine learning models. Our Scikit-Learn tutorial provides more context for the code below.

For a complete list of the Supervised Learning, Unsupervised Learning, and Dataset Transformation, and Model Evaluation modules in Scikit-Learn, please refer to its user guide.

Machine Learning
Python

```
 Import libraries and modules
nport numpy as np
nport pandas as pd
·om sklearn.model_selection import train_test_split
·om sklearn import preprocessing
·om sklearn.ensemble import RandomForestRegressor
·om sklearn.pipeline import make_pipeline
·om sklearn.model_selection import GridSearchCV
·om sklearn.metrics import mean_squared_error, r2_score
·om sklearn.externals import joblib
 Load red wine data.
ataset_url = 'http://mlr.cs.umass.edu/ml/machine-learning-databases/wine-quality/winequality-red.csv'
ata = pd.read_csv(dataset_url, sep=';')
 Split data into training and test sets
 = data.quality
 = data.drop('quality', axis=1)
_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
ipeline = make_pipeline(preprocessing.StandardScaler(),
                  RandomForestRegressor(n_estimators=100))
 Declare hyperparameters to tune
yperparameters = { 'randomforestregressor__max_features' : ['auto', 'sqrt', 'log2'],
            'randomforestregressor__max_depth': [None, 5, 3, 1]}
 Tune model using cross-validation pipeline
lf = GridSearchCV(pipeline, hyperparameters, cv=10)
lf.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
red = clf.predict(X_test)
rint r2_score(y_test, pred)
rint mean_squared_error(y_test, pred)
 Save model for future use
oblib.dump(clf, 'rf_regressor.pkl')
 To load: clf2 = joblib.load('rf_regressor.pkl')
```