Python for Data Scientists

Chapter 4: Data Preparation

Chapter Objectives

In this chapter, we will introduce:

- Pandasql
- A basic refresher on statistics
- Basic ETL (extract, transform, and load) and reshaping data for modeling
- Splitting data into training and testing sets
- Free-form text

Chapter Concepts

Pandasql

Statistics Primer

Basic ETL and Reshaping

Splitting Data

Free-Form Text

Chapter Summary

Pandasql

- Pandasql is a convenient add on to Pandas that lets you use standard SQL queries instead of complex Pandas commands
- If you know how to solve a problem with SQL already, it is sometimes a better choice
- Once installed and imported, you can basically refer to a variable that holds a Pandas DataFrame as if it were a virtual table

```
pip install pandasql

import pandas as pd
from pandasql import sqldf
pysqldf = lambda q: sqldf(q, globals())

iris = pd.read_csv('../Day3-Pandas/iris-data-index-column.csv',
index_col=0, header=0)
display(iris)

query = 'select upper(Class) as Class, Sepal_Length * 10 as S_Length,
Sepal_Width / 10 as S_Width from iris'
iris2 = pysqldf(query)
display(iris2)
```

Chapter Concepts

Statistics

- Statistics is a branch of mathematics that deals with collecting, organizing, analyzing, interpreting, and presenting data
- The math calculations behind it can be complex and time consuming to do, but there are some basic concepts that can be understood and applied without knowing all the details
- Data scientists need to understand a few basic big picture ideas in order to apply the computer models
  - Statistical features of a dataset (central tendency, min, max, IQR, deviation)
  - Probability (Normal, Uniform, Poisson, Binomial distributions)
  - Dimension reduction and sampling
  - Independent vs. Dependent variables and correlation
  - Accuracy Analysis

Statistical Features

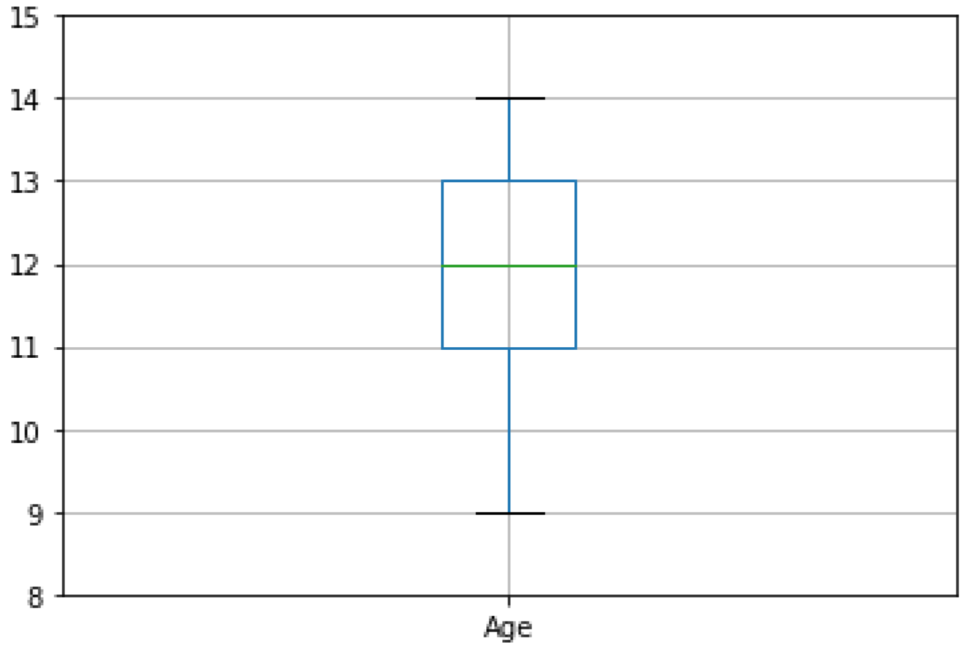
- Sometimes called *Descriptive Statistics* or *Exploratory Data Analysis*
- Gives us an overview of the range of values we find in a dataset
- Central Tendency is a measure of what usually happens and can be expressed with three different measures:
  - Mean or average is the sum of all the values divided by how many values there are
  - Median is the value in the middle of the set when all values are lined up in order
  - Mode is the single value that occurs most often in the set

```
import pandas as pd
df = pd.DataFrame([9,10,10,11,11,11,12,12,12,13,13,13,13,14],
columns=['Age'])
print ("Mean", df.Age.mean(), "Median", df.Age.median(), "Mode",
df.Age.mode()[0], "Count", df.Age.count())
print (df.Age.value_counts())
Mean 11.714285714285714 Median 12.0 Mode 13 Count 14
13    4
12    3
11    3
10    2
14    1
9     1
```

Box Plot

- Very often it is helpful to see these numbers plotted graphically instead

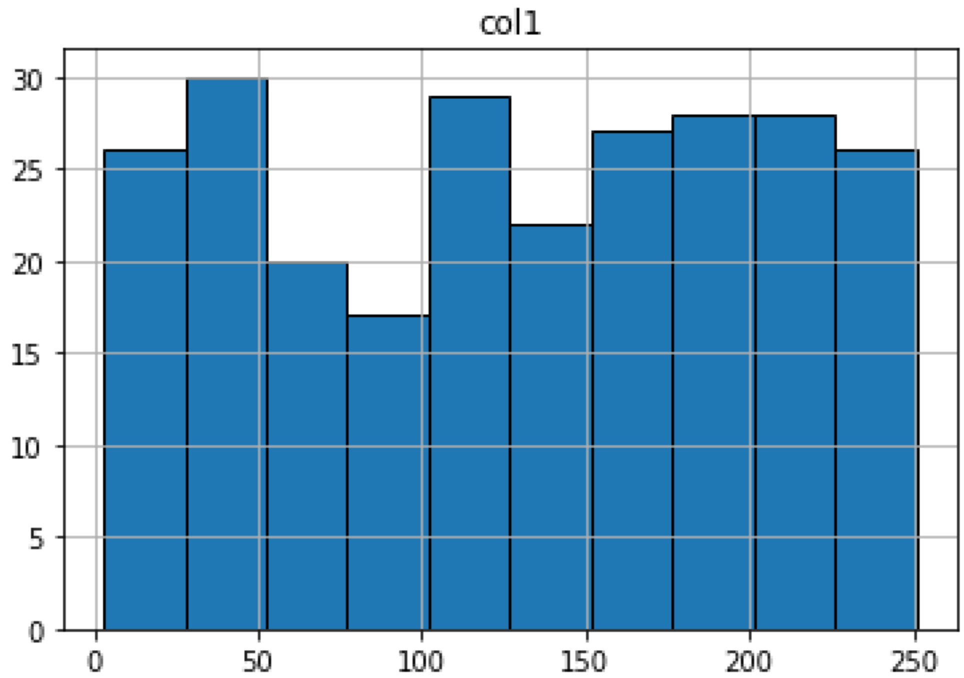
```
import matplotlib as mp
from matplotlib import pyplot as plt
plt.ylim(8,15)
df.boxplot()
```



Histogram

- Useful way to see each value range and how many items are in that range

```
import matplotlib as mp
import numpy as np
df = pd.DataFrame(np.random.rand(253, 1) * 254, columns=['col1'])
df.hist(histtype='bar', ec='black')
```



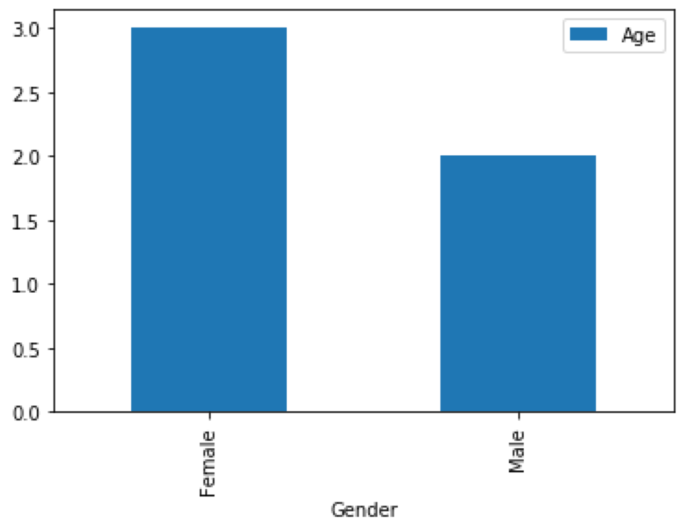
Bar Chart

- For categorical data, a bar chart is a good option

```
df = pd.DataFrame([('Male', 10),('Male', 11), ('Female', 11),
('Female', 12), ('Female', 12)], columns=['Gender','Age'])
x = df.groupby('Gender').count()
print (x)
x.plot(kind='bar')
```

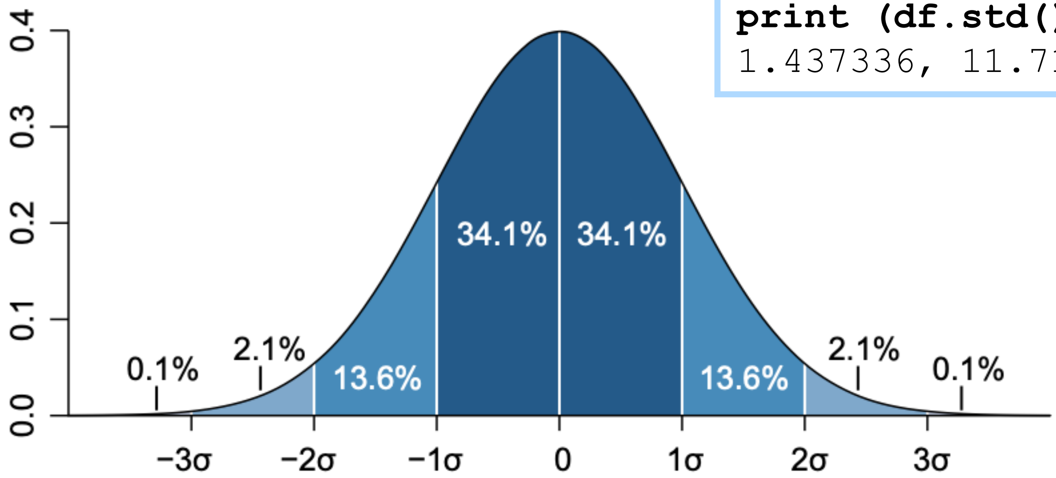
```
        Age
Gender
Female    3
Male      2

<matplotlib.axes._subplots.AxesSubplot at 0x7fed930f1358>
```



Standard Deviation

- Useful for describing how measurements are distributed in the data
- The average IQ is 100 and the standard deviation is 15 points
- A person with an IQ of 115 is one standard deviation above the mean
- A person with an IQ of 85 is one standard deviation below the mean
- A person with an IQ of 130 is two standard deviations above the mean
- By definition, 68% lie within 1 standard deviation, 95% within 2, and 99% within 3




```
print (df.std(), df.mean())
1.437336, 11.714286
```

Chapter Concepts

Basic Data Types

- Most of the models you will use require data to be perfectly clean
  - Cannot have null values
  - Text sometimes needs to be replaced with numbers
  - Numbers need to be on the same scale
- There are some basic data types in machine learning
  - Numeric
    - Continuous – decimal numbers (weight or income)
    - Discreet – whole numbers (number of students)
    - Binned – numbers grouped together to form a category (18-25) (26-35) (36-45)
  - Categorical – text or numeric that represents a category (male/female)
  - Time series – sequence of numbers collected at regular interval (temperature measurement from a weather station)
  - Text – any free-form text needs to be converted to a numeric document term matrix first

Missing Values

- Nulls or missing values can mess up the calculation
  - Need to remove them or replace them
  - Usually replace them with the central tendency (mean, median, mode) or zero
  - NaN is used in Pandas to represent Not a Number
- `isnull()` will return a series of True/False indicating if a value is Null
- `fillna()` replaces nulls with another value

```
import pandas as pd
fatal = pd.read_csv('2012_Workplace_Fatalities_by_State.csv')
print (fatal.columns)

fatal.columns = ['State', 'NumberOfFatalities', 'RateOfFatalities', 'StateRank',
'NumberOfInjuries', 'InjuriesRate', 'PenaltiesAvg', 'PenaltiesRank', 'Inspectors',
'YearsToInspectEachWorkplaceOnce', 'StateFederal']

print (fatal['PenaltiesRank'].mean())
print (fatal['PenaltiesRank'][48:])
print (fatal['PenaltiesRank'][48:].isnull())
fatal['PenaltiesRank'] = fatal['PenaltiesRank'].fillna(fatal['PenaltiesRank'].mean())
print (fatal['PenaltiesRank'][48:])
```

Fixing Up DataFrames

- `insert()` can add a new column to a DataFrame
- `drop()` removes a column from a DataFrame
- Categorical replaces strings with number placeholders
- `astype` converts the data type of a column

```
fatal.insert(11, 'Program', pd.Categorical(fatal['StateFederal']).codes)

print (fatal[['Program', 'StateFederal']])

fatal['NumberOfFatalities'].fillna(0).astype(int)

fatal.drop(['StateFederal'], axis=1, inplace=True)
```

Rescaling

- Sometimes numbers in a dataset can be on a different scale in one column vs. another
  – Weight might be measured in pounds, heights in inches
  – Value range in different columns can be wildly different
- Rescaling them to a common scale can be helpful for understanding how to compare data in different scale
- Some algorithms require that all the numbers be on the same scale, others might not care but may perform better if they are rescaled first
- Large trial and error to see what works best

```
from sklearn import preprocessing as pp
x = fatal.NumberOfFatalities
print (x.mean(), x.std(), x.min(), x.max()) → 171 624 0 4628
pp.scale(x, with_mean = False, with_std = False) → [ 60. 218. 149.  88. 137.]
pp.scale(x, with_mean = True, with_std = False) → [-111  46  -22. -83 -34]
pp.scale(x, with_mean = False, with_std = True) → [0.09 0.35 0.24 0.14 0.22]
pp.scale(x, with_mean = True, with_std = True) → [-0.17  0.07 -0.03 -0.13  -0.05]
```

concat

- Often you need to combine two DataFrames into one
  - Read in separate files and append them together like `UNION ALL` in SQL
  - Combine columns from calculation to create a new DataFrame structure
- `concat` is a function that can combine two DataFrames together, either along the row or column axis works best

```
df1 = pd.DataFrame([('Male', 10),('Male', 11), ('Female', 11), ('Female', 12),
('Female', 12)], columns=['Gender','Age'])
df2 = pd.DataFrame([('Male', 20),('Male', 21), ('Female', 21), ('Female', 22)],
columns=['Gender','Age'])
df = pd.concat([df1, df2])
print (df)
df3 = pd.DataFrame([('John', 'Smith'), ('Joe','Average'), ('Jane', 'Doe'),
('Jill', 'Hill')], columns = ['First', 'Last'])
df = pd.concat([df1, df3], axis = 1)
print (df)
```

merge

- When you need a feature like a SQL Join to match two DataFrames on a common column value, use the `merge` command

```
person_data = { 'id': ['1', '2', '3', '4', '5'],
     'first_name': ['John', 'Sue', 'Jack', 'Alice', 'Joe'],
     'last_name': ['Smith', 'Miller', 'Sprat', 'Wonderland', 'Blow']}
df1 = pd.DataFrame(person_data, columns = ['id', 'first_name', 'last_name'])

skill_data = {'id' : ['1', '1', '2', '3', '3', '3', '5', '6'],
   'skill' : ['C++', 'Java', 'Java', 'C++', 'Java', 'Python', 'Python', 'Java']}
df2 = pd.DataFrame(skill_data, columns = ['id', 'skill'])
print (pd.merge(df1, df2, on = 'id'))
print (pd.merge(df1, df2, how = 'left'))
```

Recoding Categorical Data

- Sometimes you have a column of categorical data list Status that could have values of Active, Pending, Cancelled
- In some cases, you need to re-encode this data as a sequential number
  - Categorical function will do that

```
person_data = { 'id': ['1', '2', '3', '4', '5'],
  'first_name': ['John', 'Sue', 'Jack', 'Alice', 'Joe'],
  'status': ['Active', 'Active', 'Pending', 'Cancelled',
  'Cancelled']}
df1 = pd.DataFrame(person_data, columns = ['id', 'first_name',
     'status'])
print (df1)
df1.status = pd.Categorical(df1.status).codes
print (df1)
```

Dummy Coding

- Other models require the data be encoded as multiple columns with a 0 or 1 indicating which value it is
- Sometimes you skip the first column as a baseline and sometimes not
- Also referred to as One Hot Encoding

```
person_data = { 'id': ['1', '2', '3', '4', '5'],
  'first_name': ['John', 'Sue', 'Jack', 'Alice', 'Joe'],
  'status': ['Active', 'Active', 'Pending', 'Cancelled',
  'Cancelled']}
df1 = pd.DataFrame(person_data, columns = ['id', 'first_name', 'status'])
print (df1)
dummies = pd.get_dummies(df1.status, drop_first = True)
df2 = pd.concat([df1[['id','first_name']], dummies], axis = 1)
print (df2)
dummies = pd.get_dummies(df1.status, drop_first = False)
df3 = pd.concat([df1[['id','first_name']], dummies], axis = 1)
print (df3)
```

Dummy Coding Example

- In the first case, we encoded three values into two columns
  - Active becomes the baseline as indicated with zeros in Cancelled and Pending
  - This is usually what we need for regression analysis
- In the second case, we encoded the three values into three columns each with a 1 to indicate it is the value
  - Usually need to do this for Neural Networks
- Models like Naive Bayes and Decision Trees that don't use distance calculations don't usually need to be dummy coded

```
   id first_name     status
0   1       John     Active
1   2        Sue     Active
2   3       Jack    Pending
3   4      Alice  Cancelled
4   5        Joe  Cancelled
   id first_name  Cancelled  Pending
0   1       John          0        0
1   2        Sue          0        0
2   3       Jack          0        1
3   4      Alice          1        0
4   5        Joe          1        0
   id first_name  Active  Cancelled  Pending
0   1       John       1          0        0
1   2        Sue       1          0        0
2   3       Jack       0          0        1
3   4      Alice       0          1        0
4   5        Joe       0          1        0
```

Chapter Concepts

Splitting Data

- Supervised models require that they be trained with a set of data first
- After training, you need to test the results using another set of data which has the known values you are trying to predict
- By comparing the predicted values to the known values, you can determine how good a model is at predicting
- Run the same data through multiple different algorithms with different parameters to tweak the result until you find the combination that yields the best results
- Pandas and Scikit-learn offer many ways to split a dataset into a training and testing set
  - `sample`
  - `train_test_split`
- It is important to examine the two sets to make sure they are fairly representative of the whole set and not skewed in some way

Sample

- Sample is a method built into DataFrame objects
- The following recipe can create a random sample and then return the rest to another set

```
train = fatal.sample(frac=0.8,random_state=200)
test = fatal[~fatal.index.isin(train.index)]
x0 = fatal.Program
x1 = train.Program
x2 = test.Program
print(x0.value_counts()/x0.count())
print (x1.value_counts()/x1.count())
print (x2.value_counts()/x2.count())
print (fatal.shape, train.shape, test.shape)

1      0.5
0      0.5

1      0.592
0      0.470

1      0.625
0      0.375

(42, 11) (34, 11) (8, 11)
```

train_test_split

- Convenient function to split the sets in one step
- The following recipe can create a random sample and then return the rest to another set

```
from sklearn.model_selection import train_test_split
train, test = train_test_split(fatal, test_size=0.2)
x0 = fatal.Program
x1 = train.Program
x2 = test.Program
print(x0.value_counts()/x0.count())
print (x1.value_counts()/x1.count())
print (x2.value_counts()/x2.count())
print (fatal.shape, train.shape, test.shape)

1      0.5
0      0.5

1      0.515
0      0.484

1      0.555
0      0.444

(42, 11) (33, 11) (9, 11)
```

Chapter Concepts

Text Processing

- Raw unformatted text can come in many forms
  - Emails
  - Resumes
  - White Papers
  - Tweets
- Words don't process well mathematically, so you need to convert the words into a matrix of numbers that can be run through the algorithms
- Document Term Matrix (DTM) is a restructuring of text data that describes the frequency that words or terms occur in a collection of documents (often called a corpus)
- There are many steps involved in getting the data to this format
  - Split lines into words (tokenization)
  - Removing punctuation, numbers, etc.
  - Standardizing on upper/lower case
  - Removing trivial words (of, and, or, is, etc.) called stop words
  - Stemming versions of words (run, running, runs)

Text Processing (continued)

- The steps for doing this are formulaic and there are many recipes to get there
- The result is a big matrix which can be fed into any of the models just like regular data
- Common usages include:
  – Classify a document into a category (spam/not spam)
  – Determine the overall sentiment of the document
  – Plagiarism detection
  – Finding similar papers for research

Example DTM

```
import pandas as pd
from sklearn.feature_extraction.text import CountVectorizer

def corpus_from_dir(folder):
   import os
   ret = dict(docs = [open(os.path.join(folder,f)).read() \
         for f in os.listdir(folder)], \
         ColNames = map(lambda x: x.split('.')[0], os.listdir(folder)))
   return ret

def tdm_df(docs, colNames = None, **kwargs):
   vectorizer = CountVectorizer(**kwargs)
   x1 = vectorizer.fit_transform(docs)
   df = pd.DataFrame(x1.toarray().transpose(), \
        index = vectorizer.get_feature_names())
   return df

corpus = corpus_from_dir('text')
print (corpus)
df = tdm_df(docs = corpus['docs'], colNames = corpus['ColNames'], \
     stop_words = 'english')
print (df)
```

Chapter Concepts

Next Steps

- ETL can be done at so many levels
  - At the source of the data using SQL
  - There are many ETL tools besides Python
- Explore the different ways to rescale and normalize data
- Text processing has so much more to it than just Document Term Matrix
  - Sentiment analysis
  - Word clouds
  - Term frequency-inverse document frequency
- Even binary data like images and sound can be turned into numeric data that can be run through models
  - Look for APIs to do things like image and facial recognition

Chapter Summary

In this chapter, we have introduced:

- A basic refresher on statistics
- Basic ETL (extract, transform, and load) and reshaping data for modeling
- Splitting data into training and testing sets
- Free-form text