Introduction to Python
Basics of Data Analysis Using Pandas

David Durden, Data Services Librarian
Yishan Ding, Statistical Consulting Graduate Assistant

What is Python?

- An interpreted high-level general-purpose programming language
- Both a programming language and a tool that executes scripts written in Python
- Python was created Guido van Rossum and released in 1991

Why use Python?

- Free
- Open-source with active development
- Available on all major OS's (MacOS, Linux, Windows)
- Large community
- Easy-to-learn*
- Encourages re-use and reproducibility
- Interdisciplinary and extensible

Species of Python

Anaconda (https://www.anaconda.com/) is a specific distribution of Python that includes a lot of popular packages by default, including IPython console, Jupyter Notebook, Pandas, and others.

Several OS's come with Python installed by default. However, the system Python is often used by the operating system. Modifying the system Python *can* result in some programs not working correctly.

Outline for Today's Workshop

- Some basic Python commands
- Introduction to Pandas for data analysis
- Data import and preview
- Subsetting data
- Summary statistics and reporting
- Simple plots
- Simple Linear Regression

Set Up Your Environment

We will be using a Jupyter Notebook for this workshop to avoid having to edit text files and execute them at the Command Line.

Python can be run in interactive mode: You can type commands directly into the Python interpreter and see results. Open your Command Prompt or Terminal application and type `python`.

Using the interpreter is useful for testing code and getting immediate results. However, you cannot easily edit those commands or save your work.

Creating Python Scripts

Instead of using the interpreter, we can also write Python programs and then run them.

To get started, use a plain text editor like:

- BBedit (https://www.barebones.com/products/bbedit/index.html)
- Atom (https://atom.io/)
- Notepad++ (https://notepad-plus-plus.org/)

Text documents should be set up a specific way in order to read a valid Python code.

- The first line of your document should always be: `#! /usr/bin/env python`
- Save your file with `.py`

When you have written a program, you can run it using the following basic command:
`python my_program.py`

Getting Started

- Meet the print statement
  - `print("Hello World!")`

In [1]:
```
print("Hello World!")
```

```
Hello World!
```

> ***Fun fact***: *Printing* `Hello world!` *to the screen is a rhetorical function used by programmers to test that their systems are operating correctly. Incidentally, this has also become one of the first programs that you learn in almost every programming language. The creation of this function is attributed to Brian Kernighan, who first published it in a code snippet in A Tutorial Introduction to the Programming Language B in 1973.*

Assign values to a variable

You can store anything in a variable in Python.

- `x = 3 # Assign a single integer value`
- `y = [1, 2, 3] # Assign a list of integer values`
- `z = ["a", "b", "c"] # Assign a list of character values`
- `a = [y, z] # Assign a list of variables`
- `b = a # Assign a variable to a variable`

> *TIP: You can either 'call' a variable directly or print it using the `print()` function.*

Variables

In [32]:
```
x = 3 # Assign a single integer value
y = [1, 2, 3] # Assign a list of integer values
z = ["a", "b", "c"] # Assign a list of character values
a = [x, y] # Assign a list of variables
b = a

print(x)
print(y)
print(z)
print(a)
print(b)
```

```
3
[1, 2, 3]
['a', 'b', 'c']
[3, [1, 2, 3]]
[3, [1, 2, 3]]
```

Using Python as a Calculator

- `35*40/4`
- `((6-4)*(2+5))/5`
- Advanced mathematical concepts are implemented in the `math` library which needs to be imported before we can use it
  - `from math import sqrt`
    - `sqrt(2)`

Do some Arithmetic

In [3]:
```
35*40/4
```

Out[3]:
```
350.0
```

In [4]:
```
from math import sqrt
sqrt(2)
```

Out[4]:
```
1.4142135623730951
```

5 Basic Data Types

Type	Example
Integer	`3` or `int(3)`
Float	`3.1415` or `float(6/5)`
Character	`"a"`, `"b"`, `"University of Maryland"` or `str()`
Logical	`True`, `False` or `bool()`
Complex	`1+4j` or `complex(1, 4)`

- To determine an object type, use `type()`
- To test if an object is a specific type, use `isinstance()`
- To change type, use `int()`, `float()`, `str()`, `bool()`, `complex()`

Data Types

In [5]:
```
type(3)
```

Out[5]:
```
int
```

In [33]:
```
type(3.1415)
```

Out[33]:
```
float
```

In [34]:
```
type("Testudo")
```

Out[34]:
```
str
```

In [6]:
```
foo = 6/5
isinstance(foo, float)
```

Out[6]:
```
True
```

In [7]:
```
complex(1, 4)
```

Out[7]:
```
(1+4j)
```

Pandas: The Python Data Analysis Library

Pandas (https://pandas.pydata.org/) is already included by default in Anaconda Python. If you are using a different distribution you will have to download and install Pandas.

Pandas provides easy-to-use methods for working with data structures and performing data analysis using the Python language and enviroment.

Many of the commands and capabilities are similar to R.

> *We have to import Pandas before we can use it, similar to how we imported the `math` library.*

- `import pandas`
- `import pandas as pd`

Import Pandas

In [22]:
```
# To get started using Pandas we have to import it first
import pandas

# Imported libraries can be aliased to make code shorter
# and easier to type
# pd is the common alias for Pandas
import pandas as pd
# This is an optional parameter to limit the amount of
# rows printed to the screen
pd.set_option('display.max_rows', 10)
```

Basic Commands

Here are the basic commands when working with DataFrames in Pandas

`df.max()`
`df.min()`
`df.mean()`
`df.median()`
`df.sum()`
`df.cumsum()`
	`df.var()`
	`df.std()`
	`len(df.index)` or `df.count()`
	`df.describe()`
	`df[]`

Import a Dataset

We will begin by importing the `GSSsubset.csv` file

Pandas uses the `read_csv()` function to import csv files.

> *File names are string data are should be wrapped in quotes!*

In [23]:
```
# Note that pd.read_csv is used because we imported pandas as pd
pd.read_csv("data/GSSsubset.csv")
# For Windows users:
#pd.read_csv("data\GSSsubset.csv")
```

Out[23]:

	id	sex	degree	income	marital	age	height	weig
0	1	MALE	BACHELOR	60967.50	DIVORCED	53	72	190
1	2	FEMALE	BACHELOR	60967.50	MARRIED	26	60	97
2	4	FEMALE	BACHELOR	10161.25	MARRIED	56	68	160
3	14	FEMALE	HIGH SCHOOL	17551.25	MARRIED	40	65	156
4	16	MALE	HIGH SCHOOL	17551.25	MARRIED	56	66	210
...	...	...	...	...	...	...	...	...
989	2531	MALE	HIGH SCHOOL	1478.00	NEVER MARRIED	40	71	230
990	2535	MALE	HIGH SCHOOL	33255.00	DIVORCED	56	72	195
991	2536	MALE	HIGH SCHOOL	8313.75	NEVER MARRIED	24	68	145
992	2537	MALE	HIGH SCHOOL	27712.50	NEVER MARRIED	27	68	180

	id	sex	degree	income	marital	age	height	weig
993	2538	FEMALE	HIGH SCHOOL	15703.75	WIDOWED	71	63	140

994 rows × 9 columns

Let's Examine a Dataset

When importing a dataset we should assign it to a variable for future use.

Declare a variable name and assign the `read_csv()` function from earlier.

In [5]:
```
# df is common short hand in Pandas for "DataFrame", which is the object type
# we are creating when importing a csv file
gss_df = pd.read_csv("data/GSSsubset.csv")
# For Windows users:
# gss_df = pd.read_csv("data\GSSsubset.csv")
```

Preview the First Few Lines of a Dataset

After we assign our dataset to a variable, we can preview it with `head()`

Syntax: ***`dataFrame.head()`***

In [6]:
```
gss_df.head()
# head() can be used on any dataframe object!
```

Out[6]:

	id	sex	degree	income	marital	age	height	weight	h
0	1	MALE	BACHELOR	60967.50	DIVORCED	53	72	190	6
1	2	FEMALE	BACHELOR	60967.50	MARRIED	26	60	97	4
2	4	FEMALE	BACHELOR	10161.25	MARRIED	56	68	160	2
3	14	FEMALE	HIGH SCHOOL	17551.25	MARRIED	40	65	156	3
4	16	MALE	HIGH SCHOOL	17551.25	MARRIED	56	66	210	6

Examine a Dataset

`head()` let's us look at the first few lines of a file.

`tail()` will let us look that last few lines of a file.

`describe()` will give us summary information.

Preview with tail()

In [7]:
```
gss_df.tail()
```

Out[7]:

	id	sex	degree	income	marital	age	height	weight
989	2531	MALE	HIGH SCHOOL	1478.00	NEVER MARRIED	40	71	230
990	2535	MALE	HIGH SCHOOL	33255.00	DIVORCED	56	72	195
991	2536	MALE	HIGH SCHOOL	8313.75	NEVER MARRIED	24	68	145
992	2537	MALE	HIGH SCHOOL	27712.50	NEVER MARRIED	27	68	180
993	2538	FEMALE	HIGH SCHOOL	15703.75	WIDOWED	71	63	140

Return Summary Statistics with describe()

In [8]:
```
print(gss_df.describe())
```

```
                id         income         age      height      weight  \
count   994.000000     994.000000  994.000000  994.000000  994.000000
mean   1271.183099   36887.218352   44.488934   67.414487  181.318913
std     734.068659   34702.256068   13.126865    4.037741   41.641615
min       1.000000     369.500000   19.000000   57.000000   90.000000
25%     648.250000   15703.750000   33.000000   64.000000  150.000000
50%    1254.500000   27712.500000   44.000000   67.000000  175.000000
75%    1915.750000   49882.500000   55.000000   70.000000  205.000000
max    2538.000000  158656.952000   79.000000   79.000000  410.000000

           hrswrk
count  994.000000
mean    42.640845
std     14.394974
min      1.000000
25%     38.000000
50%     40.000000
75%     50.000000
max     89.000000
```

Exercise 1: Import a Dataset

- Import the dataset `ISBN.csv`
- Preview the dataset
  - Call the variable that you assigned to `ISBN.csv`
  - Call the `head()` method on the DataFrame

Examine the Dataset

- `df.shape` where `df` is the name of your DataFrame
  - Returns row x column
- `df.columns`
  - Returns a list of the column names
- `df.index`
  - Returns row information (range and step)
- `df.info()`
  - Returns list of variables, data type, count, size, etc.

Exercise 2

- Examine the dimensions of `ISBN.csv`
  - How many rows does this dataset have?
  - How many columns?
  - What are the variable names?

In [24]:
```
print(gss_df.shape)
```

```
(994, 9)
```

In [25]:
```
print(gss_df.describe())
```

```
                id         income         age      height      weight  \
count   994.000000     994.000000  994.000000  994.000000  994.000000
mean   1271.183099   36887.218352   44.488934   67.414487  181.318913
std     734.068659   34702.256068   13.126865    4.037741   41.641615
min       1.000000     369.500000   19.000000   57.000000   90.000000
25%     648.250000   15703.750000   33.000000   64.000000  150.000000
50%    1254.500000   27712.500000   44.000000   67.000000  175.000000
75%    1915.750000   49882.500000   55.000000   70.000000  205.000000
max    2538.000000  158656.952000   79.000000   79.000000  410.000000

           hrswrk
count  994.000000
mean    42.640845
std     14.394974
min      1.000000
25%     38.000000
50%     40.000000
75%     50.000000
max     89.000000
```

In [40]:
```
gss_df.columns
```

```
Index(['id', 'sex', 'degree', 'income', 'marital', 'age', 'height', 'weight',
       'hrswrk'],
      dtype='object')
```

In [96]:
```
gss_df.index
```

Out[96]:
```
RangeIndex(start=0, stop=994, step=1)
```

In [43]:
```
gss_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 994 entries, 0 to 993
Data columns (total 9 columns):
id          994 non-null int64
sex         994 non-null object
degree      994 non-null object
income      994 non-null float64
marital     994 non-null object
age         994 non-null int64
height      994 non-null int64
weight      994 non-null int64
hrswrk      994 non-null int64
dtypes: float64(1), int64(5), object(3)
memory usage: 70.0+ KB
```

Subset the Dataset

1. Subset by row/column name	2. Subset by row/column index
df.loc[rowname, colname]	**df.iloc[rowindex, colindex]**
-data.loc[2, 'degree'] -data.loc[:3,'id':'income'] -data.loc[[1,3,5],['id', 'degree', 'income']	-data.iloc[1, 3] -data.iloc[:5,] -data.iloc[900:, [2,3,5,8]]

3. Subset by variable name	4. Subset by specific criteria
df['colname'] **df.colname**	**data[data['variable'] <operator> 'criteria']**
-data['id'] -data[['id', 'degree', 'income']] -data.degree	-data[data['age'] < 50] -data[data['sex'] == 'MALE'] -data[(data['age'] < 50) & (data['sex'] == 'MALE')]

Refer to this (https://cmdlinetips.com/2019/03/how-to-select-one-or-more-columns-in-pandas/) article for a more in-depth explanation of subsetting in Pandas.

In [26]:
```
# Here is an example of a subset for males less than 50 years of age
# Note that this expression includes the head() method!
gss_df[(gss_df['age'] < 50) & (gss_df['sex'] == 'MALE')].head()
```

Out[26]:

	id	sex	degree	income	marital	age	height	weight	h
7	27	MALE	HIGH SCHOOL	17551.25	SEPARATED	35	70	180	4
11	38	MALE	HIGH SCHOOL	33255.00	DIVORCED	31	67	150	3
12	40	MALE	BACHELOR	73900.00	MARRIED	35	69	200	5
13	44	MALE	HIGH SCHOOL	24017.50	NEVER MARRIED	26	76	165	4
16	47	MALE	JUNIOR COLLEGE	40645.00	NEVER MARRIED	30	72	185	4

Exercise 3

Using the `ISBN.csv` dataset answer the following question:

- In which years was the *Journal of Crystal Growth* requested?

  Hint: You will need to subset by criteria and then subset by column name!

- You can also sor data using the following code:
  - `df.sort_values(by=['colname'], inplace=False, ascending=False)`

Summary Report

- Summary of each variable: `df.describe()`
- Summary statistics for subgroups/categories:
  - **Example question:** According to the `GSSsubset` dataset, what is the average income by sex?

In [99]:
```
gss_df.groupby(['sex'])[['income']].aggregate('mean')
```

Out[99]:

	income
sex	
FEMALE	27300.446119
MALE	46095.814166

Exercise 4: Answering Questions with Data

- Import the dataset *`lib_checkout.csv`*
- How many books were checked out by undgrads in total?
- How many books were checked out by graduate students in total?
- How many books were checked out by day of the week on average?

Bonus question:

- *How many books were checked out by grad students and undergraduate students by day of the week on average? (use only **one** command)*
- *Assign the result to a variable named* `weekmean`

In [27]:
```
# Read the lib_checkout dataset and store in a variable
libdata = pd.read_csv("data/lib_checkout.csv")
# For Windows users
#libdata = pd.read_csv("data\lib_checkout.csv")

# Subset by undergrad
ulib = libdata[libdata['status'] == 'Undergrad']
```

In [11]:
```
# Preview the subset
ulib.head()
```

Out[11]:

	date	day	count	status	laptop_checkout	book_checkou
0	9/1/2014	Monday	6570	Undergrad	329	1643
2	9/2/2014	Tuesday	5852	Undergrad	585	1522
4	9/3/2014	Wednesday	7060	Undergrad	847	1836
6	9/4/2014	Thursday	8065	Undergrad	968	2097
8	9/5/2014	Friday	6323	Undergrad	759	1770

Plotting in Python/Pandas

We need to import `matplotlib`, a popular scientific plotting library

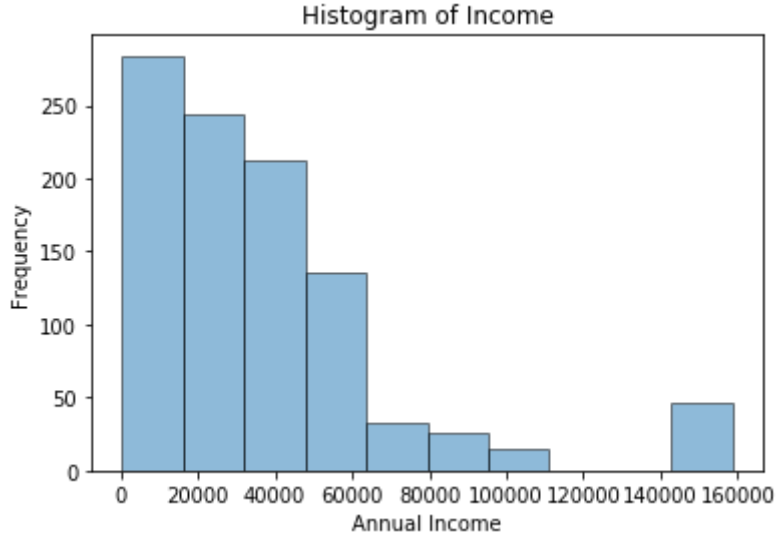
In [12]:
```
# This command forces plots to display in Jupyter Notebooks
%matplotlib inline
# This command imports matplotlib and assigns the alias plt
import matplotlib.pyplot as plt
```

Univariate Graph: Histogram

More information on plotting historgrams here (https://pythonspot.com/matplotlib-histogram/)

In [13]:
```
# Graph the data
plt.hist(gss_df['income'], edgecolor='black', linewidth='1', alpha=0.5)
# Add title and axes labels
plt.title('Histogram of Income')
plt.xlabel('Annual Income')
plt.ylabel('Frequency')
```

Out[13]:
```
Text(0,0.5,'Frequency')
```

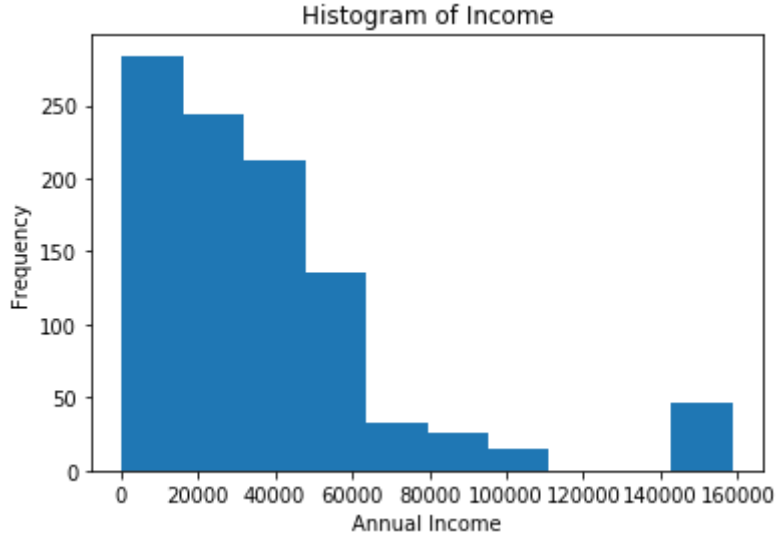


Another way to Create a Histogram

The `plot()` function is useful for when you don't want to mess with the aesthetics of graph.

In [14]:
```
gss_df['income'].plot(kind='hist', title='Histogram of Income').set(xlabel='Annual Income')
```

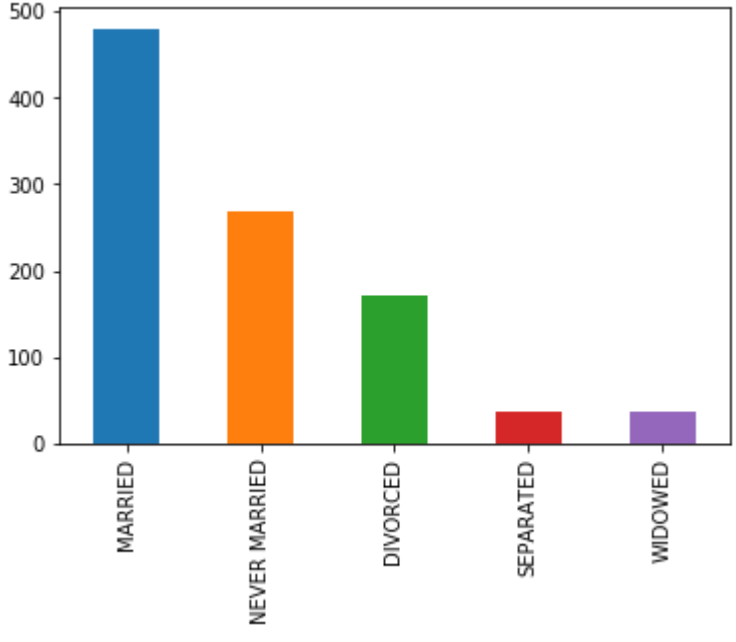
Out[14]:
```
[Text(0.5,0,'Annual Income')]
```



Univariate Graph: Bar Chart

In [15]:
```
# To plot a single variable on a bar chart, we need
# to count the individual values first
gss_df['marital'].value_counts().plot(kind='bar')
```

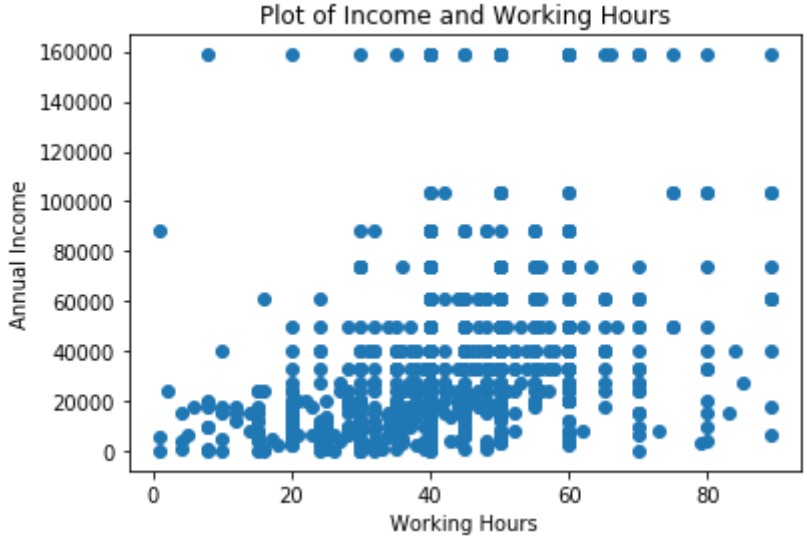
Out[15]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x1188dc940>
```



Bivariate Graph: Scatterplot

In [16]:
```
# Graph the data
plt.scatter(gss_df['hrswrk'], gss_df['income'])
# Set the axes and title labels
plt.title('Plot of Income and Working Hours')
plt.xlabel('Working Hours')
plt.ylabel('Annual Income')
```

Out[16]:
```
Text(0,0.5,'Annual Income')
```



Exercise 5: Plot Some Data

Dataset: *`lib_checkout.csv`*

Draw a scatterplot showing `laptop_checkout` as the x variable and `book_checkout` as the y variable for **undergraduate students only**

Hint:

- *Create a subset of the original data by `status` and save to a new DataFrame*
- *Plot using the new DataFrame*

Simple Linear Regression

$$y = \beta_0 + \beta_1 x + e$$

- y : dependent variable
- x : independent variable
- e : error term
- β_0 : intercept
- β_1 : slope

Simple Linear Regression

1. **F-test $\mathbf{p}$-value**: *Is this model statistically significant?*
2. $\mathbf{r^2}$: *How much variance in y can be explained by the variance in x?*
3. **Intercept**
4. **Slope**

Linear Regressions Using `statsmodels.api`

In [17]:
```
import numpy as np
import statsmodels.api as sm

# This is the difficult way of implementing a linear model in Python
y1 = gss_df['income'] # Dependent variable
x1 = gss_df['hrswrk'] # Independent variable
y1, x1 = np.array(y1), np.array(x1) # Convert to numpy arrays
x1 = sm.add_constant(x1) # Add constant (column of 1's)
model = sm.OLS(y1, x1) # Define the model
fitted = model.fit() # Fit the model
print(fitted.summary()) # Print the results
```

```
/anaconda3/lib/python3.6/site-packages/statsmodels/compat/pandas.py:56: Future
Warning: The pandas.core.datetools module is deprecated and will be removed in
a future version. Please use the pandas.tseries module instead.
  from pandas.core import datetools

                            OLS Regression Results
==============================================================================
=
Dep. Variable:                      y   R-squared:                       0.11
8
Model:                            OLS   Adj. R-squared:                  0.11
7
Method:                 Least Squares   F-statistic:                     132.
3
Date:                Wed, 24 Jul 2019   Prob (F-statistic):           7.79e-2
9
Time:                        10:21:35   Log-Likelihood:                -1174
0.
No. Observations:                 994   AIC:                          2.348e+0
4
Df Residuals:                     992   BIC:                          2.349e+0
4
Df Model:                           1
Covariance Type:            nonrobust
==============================================================================
```

Linear Regressions Using `statsmodels.formula.api`

In [18]:
```
import statsmodels.formula.api as smf

# This is the easier way of implementing a linear model using
# R-like formulas
model1 = smf.ols(formula='income ~ hrswrk', data=gss_df).fit() # Note the lowercase 'ols'
print(model1.summary())                                        # Required to use patsy formulae
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                 income   R-squared:                       0.118
Model:                            OLS   Adj. R-squared:                  0.117
Method:                 Least Squares   F-statistic:                     132.3
Date:                Wed, 24 Jul 2019   Prob (F-statistic):           7.79e-29
Time:                        10:22:05   Log-Likelihood:                -11740.
No. Observations:                 994   AIC:                         2.348e+04
Df Residuals:                     992   BIC:                         2.349e+04
Df Model:                           1
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
Intercept   1621.4901   3235.485      0.501      0.616   -4727.690    7970.670
hrswrk       827.0410     71.895     11.503      0.000     685.957     968.125
==============================================================================
Omnibus:                      421.362   Durbin-Watson:                   1.708
Prob(Omnibus):                  0.000   Jarque-Bera (JB):             1713.357
Skew:                           2.027   Prob(JB):                         0.00
Kurtosis:                       7.993   Cond. No.                         141.
==============================================================================

Warnings:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
```

Exercise 6: Linear Regressions

Use the undergraduate subset created in Exercise 4

1. Run a simple linear regression of `book_checkout` (dependent variable, y) on `laptop_checkout` (independent variable, x)
2. Using the regression output, write down the regression model as comment in your code
3. Is the model significant on alpha = 0.05?

In [29]:
```
#y = ulib['book_checkout']
#x = ulib['laptop_checkout']
model = smf.ols(formula='book_checkout ~ laptop_checkout', data=ulib).fit()
print(model.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:          book_checkout   R-squared:                       0.216
Model:                            OLS   Adj. R-squared:                  0.188
Method:                 Least Squares   F-statistic:                     7.714
Date:                Wed, 24 Jul 2019   Prob (F-statistic):            0.00967
Time:                        11:14:33   Log-Likelihood:                -217.97
No. Observations:                  30   AIC:                             439.9
Df Residuals:                      28   BIC:                             442.7
Df Model:                           1
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
Intercept      1316.3691    158.720      8.294      0.000     991.245    1641.493
laptop_checkout     0.7280      0.262      2.777      0.010       0.191       1.265
==============================================================================
Omnibus:                        1.573   Durbin-Watson:                   2.523
Prob(Omnibus):                  0.455   Jarque-Bera (JB):                1.193
Skew:                           0.481   Prob(JB):                        0.551
Kurtosis:                       2.832   Cond. No.                     1.47e+03
==============================================================================

Warnings:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
```

```
[2] The condition number is large, 1.47e+03. This might indicate that there ar
e
strong multicollinearity or other numerical problems.
```

Thank you!

David Durden

✉️ durden@umd.edu (mailto:durden@umd.edu)
🖥️ https://lib.umd.edu/data ()

Yishan Ding

✉️ ysding@umd.edu (mailto:ysding@umd.edu)
🖥️ https://www.lib.umd.edu/rc/statistical-consulting ()