

# Data Characterization and Regression

---

# Today's Outline

- Course introduction
  - Structure and grading
- Univariate Characterizations
  - Descriptive Statistics
  - Histograms
  - Q-Q Plots
  - Boxplots
- Bivariate Characterizations
- Linear Regressions in Python

**Note: Assignment 1 will be posted next Monday, due on October 9th**

# About This Course

- The purpose of this course will be to demonstrate how to implement the statistical techniques you learn in ECON 430
- Whenever time allows, you will be given opportunities to practice these techniques **during class**
- As a review, an intuitive explanation of the technique and some practical considerations may be covered



# More About this Class

- Assignments will be constructed to match the content of this lab and ECON 430
- All of three homeworks written by me should be completed in Python
  - Professor Rojas will provide additional examples in R as well as python
- Students often spend many hours troubleshooting simple implementation issues
- This course will help you resolve any questions you have about:
  - Interpreting results
  - Resolving errors
  - How techniques could be used in practice

# Getting Data

- Data will often come from the **wooldridge** module in python
- Simply need to know the name of the dataset being worked with (no download required)
- Other times data will be provided

```
1 # import the wooldridge module
2 import wooldridge as woo
3
4 # use the .data() function to pull down the necessary dataset
5 salary = woo.data('ceosal1')
6 salary.head()
```

	salary	pcsalary	sales	roe	pcroe	ros	indus	finance	consprod	utility	lsalary	lsales
0	1095	20	27595.000000	14.1	106.400002	191	1	0	0	0	6.998509	10.225389
1	1001	32	9958.000000	10.9	-30.600000	13	1	0	0	0	6.908755	9.206132
2	1122	9	6125.899902	23.5	-16.299999	14	1	0	0	0	7.022868	8.720281
3	578	-9	16246.000000	5.9	-25.700001	-21	1	0	0	0	6.359574	9.695602
4	1368	7	21783.199219	13.8	-3.000000	56	1	0	0	0	7.221105	9.988894

# Example Dataset

- Dataset used to investigate the relationship between CEO salary and returns
- Sample of data was reported in the May 6, 1991 issue of Businessweek

- **salary**: 1990 salary, thousands \$
- **pcsalary**: percent change salary, 89-90
- **sales**: 1990 firm sales, millions \$
- **roe**: return on equity, 88-90 avg
- **pcroe**: percent change roe, 88-90
- **ros**: return on firm's stock, 88-90
- **indus**: =1 if industrial firm
- **finance**: =1 if financial firm
- **consprod**: =1 if consumer product firm
- **utility**: =1 if transport. or utilities
- **lsalary**: natural log of salary
- **lsales**: natural log of sales

```
1 # import the wooldridge module
2 import wooldridge as woo
3
4 # use the .data() function to pull down the necessary dataset
5 salary = woo.data('ceosal1')
6 salary.head()
```

]:

	salary	pcsalary	sales	roe	pcroe	ros	indus	finance	consprod	utility	lsalary	lsales
0	1095	20	27595.000000	14.1	106.400002	191	1	0	0	0	6.998509	10.225389
1	1001	32	9958.000000	10.9	-30.600000	13	1	0	0	0	6.908755	9.206132

# Pandas Describe

- The `df.describe()` method produces a table of summary statistics
- This allows us to quickly identify issues with our dataset
- Issues Include:
  - NAs
  - Unexpected values
  - Differences in variable magnitude

```
1 # Describing the data
2 salary.describe()
```

	salary	pcsalary	sales	roe	pcroe	ros	indus	finance	consprod	utility	lsalary	lsale
count	209.000000	209.000000	209.000000	209.000000	209.000000	209.000000	209.000000	209.000000	209.000000	209.000000	209.000000	209.000000
mean	1281.119617	13.282297	6923.793282	17.184211	10.800478	61.803828	0.320574	0.220096	0.287081	0.172249	6.950386	8.292268
std	1372.345308	32.633921	10633.271088	8.518509	97.219399	68.177052	0.467818	0.415306	0.453486	0.378503	0.566374	1.01316
min	223.000000	-61.000000	175.199997	0.500000	-98.900002	-58.000000	0.000000	0.000000	0.000000	0.000000	5.407172	5.16592
25%	736.000000	-1.000000	2210.300049	12.400000	-21.200001	21.000000	0.000000	0.000000	0.000000	0.000000	6.601230	7.70088
50%	1039.000000	9.000000	3705.199951	15.500000	-3.000000	52.000000	0.000000	0.000000	0.000000	0.000000	6.946014	8.21749
75%	1407.000000	20.000000	7177.000000	20.000000	19.500000	81.000000	1.000000	0.000000	1.000000	0.000000	7.249215	8.87863
max	14822.000000	212.000000	97649.898438	56.299999	977.000000	418.000000	1.000000	1.000000	1.000000	1.000000	9.603868	11.48914

# Data Types and NAs

```
1 # Get information about the data types
2 # Also data structure
3 salary.info()
```

```
<class 'pandas.core.frame.DataFrame'>
```

```
RangeIndex: 209 entries, 0 to 208
```

```
Data columns (total 12 columns):
```

#	Column	Non-Null Count	Dtype
0	salary	209 non-null	int64
1	pcsalary	209 non-null	int64
2	sales	209 non-null	float64
3	roe	209 non-null	float64
4	pcroe	209 non-null	float64
5	ros	209 non-null	int64
6	indus	209 non-null	int64
7	finance	209 non-null	int64
8	consprod	209 non-null	int64
9	utility	209 non-null	int64
10	lsalary	209 non-null	float64
11	lsales	209 non-null	float64

```
dtypes: float64(5), int64(7)
```

```
memory usage: 19.7 KB
```

```
1 # Count the number of null values in every column
2 salary.isnull().sum()
```

```
salary      0
pcsalary    0
sales        0
roe          0
pcroe       0
ros         0
indus       0
finance     0
consprod    0
utility     0
lsalary     0
lsales      0
dtype: int64
```

```
1 # Check if there are any null values in the dataframe
2 # True if any contain an NA
3 salary.isnull().any()
```

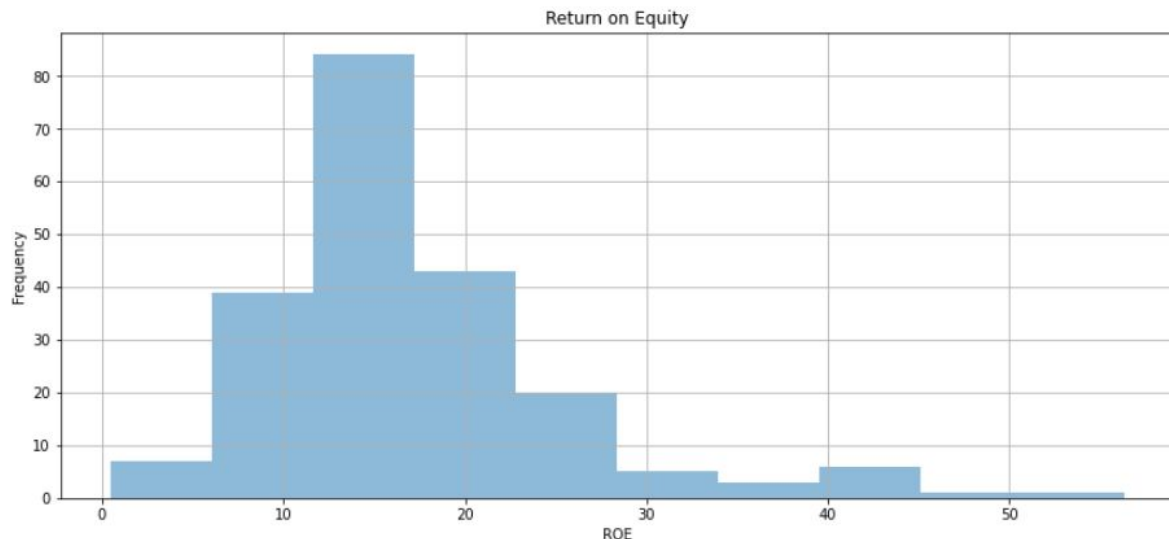
```
salary      False
pcsalary    False
sales        False
roe          False
pcroe       False
ros         False
indus       False
finance     False
consprod    False
utility     False
lsalary     False
lsales      False
dtype: bool
```



# Histograms in Matplotlib

- Histograms can be used to learn about a variable's skewness, outliers, range, and general shape
- Matplotlib makes creating histograms simple with *plt.hist(series)*
- A very quick histogram can be produced using *series.hist()*

```
1 # Set the figure size
2 plt.figure(figsize = (14, 6))
3
4 # Plot the data
5 plt.hist(salary["roe"], alpha = .5)
6
7 # add Labels
8 plt.title("Return on Equity")
9 plt.ylabel("Frequency")
10 plt.xlabel("ROE")
11
12 # add grid lines
13 plt.grid()
```



# Number of Bins

- The hist() function will automatically a number of bins that usually works well
- The bins argument can take an integer as an argument
- Two other rules of thumb are discussed in ECON 430:

## Sturges

$$k = 1 + \log_2(n)$$



```
1 #math module lets you use a custom base for log
2 import math
3 # Sturges method
4 bins = np.round(1 + math.log(len(salary),2))
5
6 #plotting
7 plt.figure(figsize = (16, 8))
8 plt.hist(salary.roe, alpha = .5, bins = bins)
9 plt.plot()
```

## Freedman's and Diaconis (FD)

$$\frac{n^{1/3}(\max - \min)}{2(Q_3 - Q_1)}$$

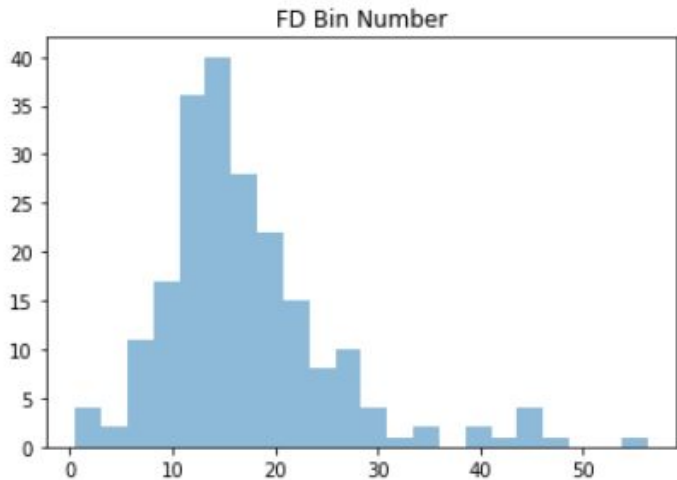


```
1 # create FD function
2 def bins_fd(series):
3     numerator = (len(series)**(1/3)*(series.max()-series.min()))
4     denominator = (2*(salary.roe.quantile(.75)-salary.roe.quantile(.25)))
5
6     return int(np.round(numerator/denominator))
7
8 plt.figure(figsize = (16, 8))
9 plt.hist(salary.roe, alpha = .5, bins = bins_fd(salary.roe))
10 plt.plot()
```

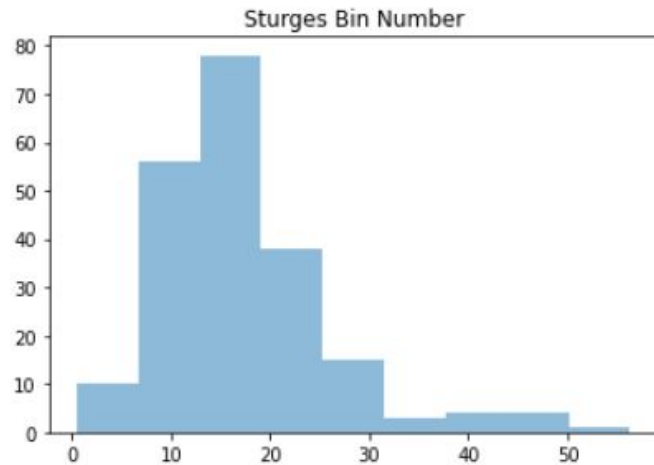
# Bins in Matplotlib

- The [bins argument also accepts](#) some strings that will produce results identical to the previous slide
- Sequences can also be used instead to specify the the bin edges

```
1 plt.hist(salary.roe, alpha = .5, bins = "fd")|
2 plt.title("FD Bin Number")
3 plt.show()
```



```
1 plt.hist(salary.roe, alpha = .5, bins = "sturges")
2 plt.title("Sturges Bin Number")
3 plt.show()
```



# Seaborn Layers

- Seaborn is a layer built on top of matplotlib
- Simplifies the process of creating many complex and visually appealing plots
- Features:
  - Beautiful out of the box plots with different themes
  - Built-in color palettes that can be used to reveal patterns in the dataset
  - Dataset-oriented interface
  - A high-level abstraction that still allows for complex visualization
- Seaborn code is compatible with matplotlib (so you can change seaborn plot elements with matplotlib functions)

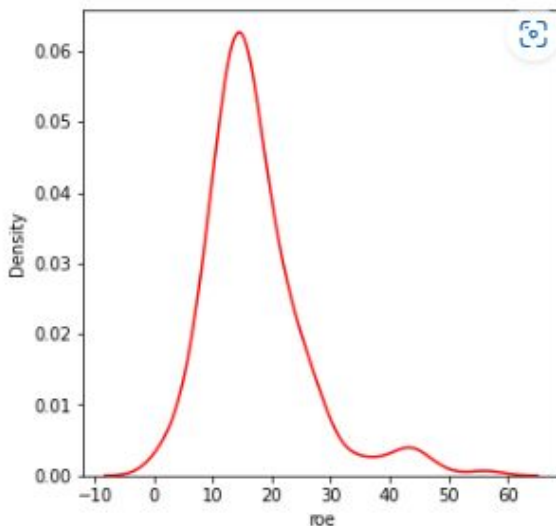


# Density Estimation

- The `kdeplot()` function from `seaborn` allows us to visualize a smoothed density plot
- The `rugplot()` function creates ticks for each observation on the x-axis

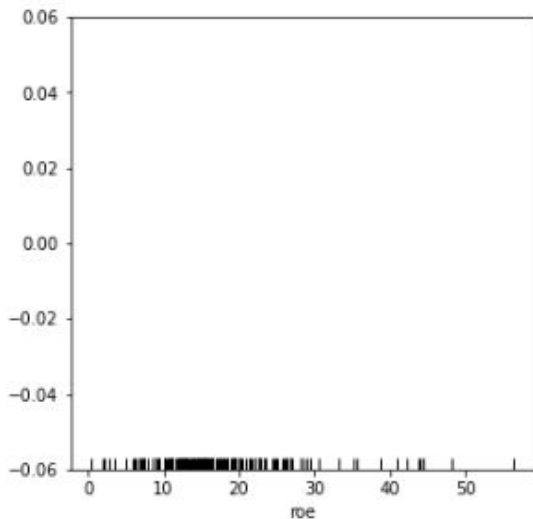
```
1 plt.figure(figsize=(5,5))
2 # Density plot
3 sns.kdeplot(salary.roe, color = "red")
```

<AxesSubplot:xlabel='roe', ylabel='Density'>



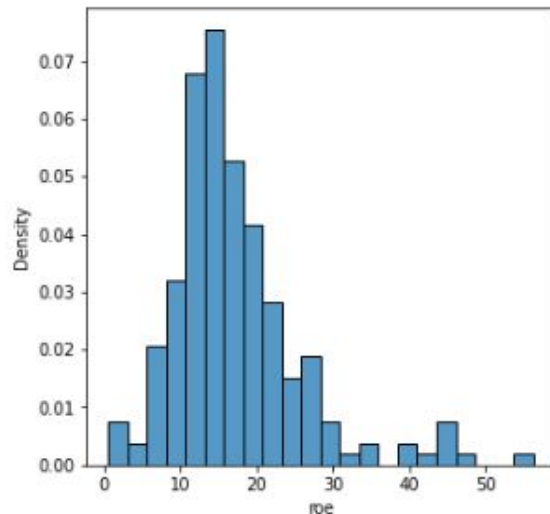
```
1 plt.figure(figsize=(5,5))
2 # Rug plot
3 sns.rugplot(salary.roe, color = "black")
```

<AxesSubplot:xlabel='roe'>



```
1 plt.figure(figsize=(5,5))
2 sns.histplot(salary.roe, stat = "density")
```

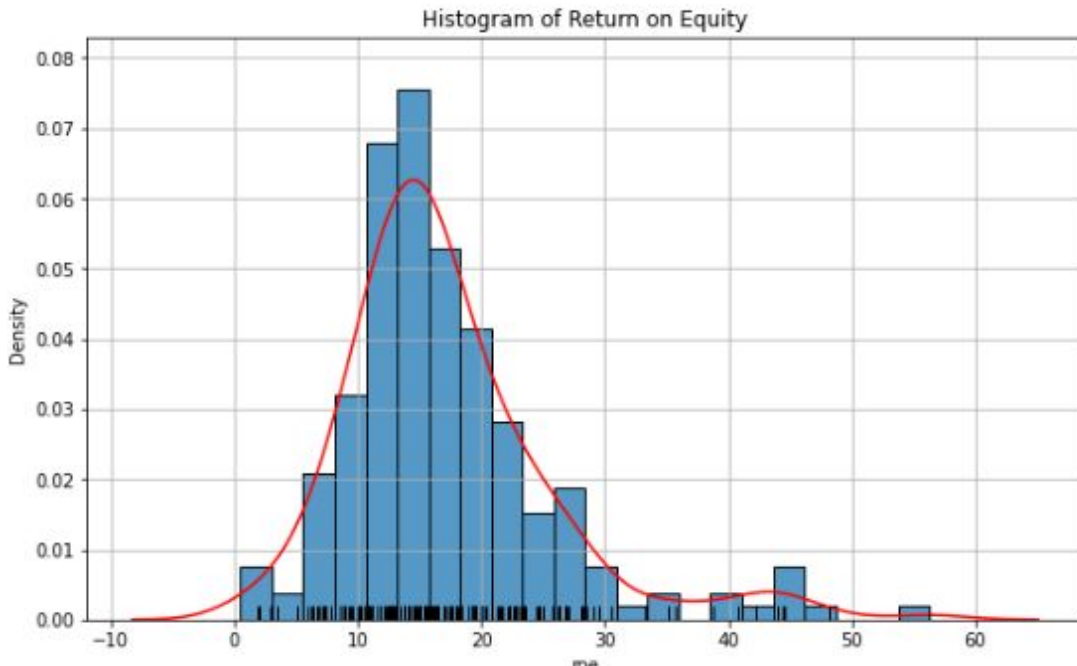
<AxesSubplot:xlabel='roe', ylabel='Density'>



# Density Estimation

- The plots can be merged by using each in the same Jupyter notebook cell
- Matplotlib functions can add labels and change the figure size
- Seaborn has its own methods for doing each of these things as well

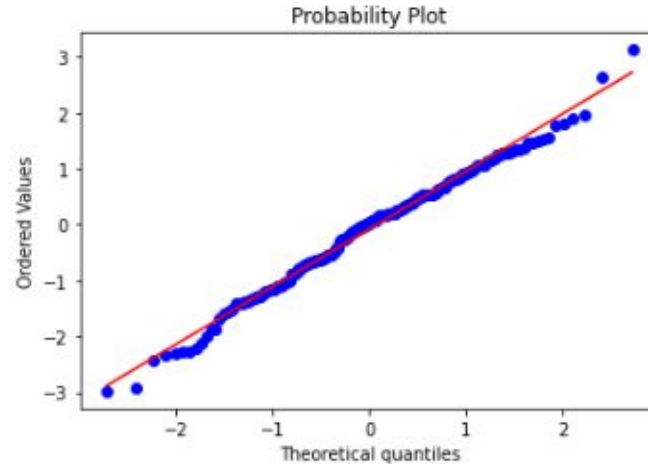
```
1 plt.figure(figsize=(10,6))
2 plt.title("Histogram of Return on Equity")
3 sns.histplot(salary.roe, stat = "density")
4 sns.kdeplot(salary.roe, color = "red")
5 sns.rugplot(salary.roe, color = "black")
6
7 plt.grid()
```



# Quantile Plots

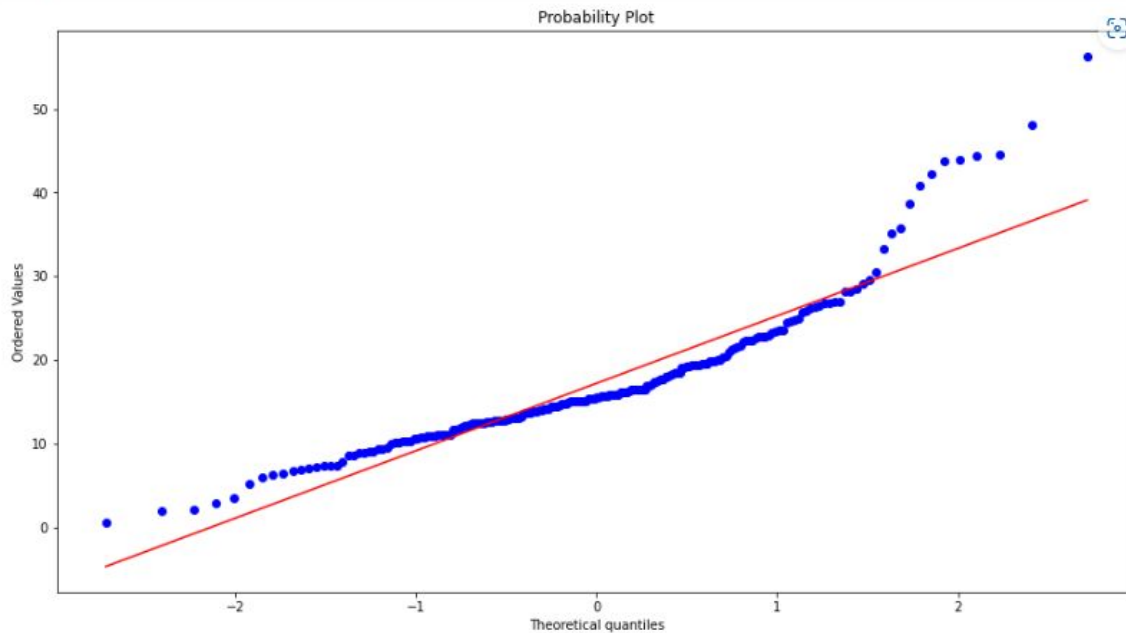
- Quantile plots are a quick visual method for comparing one variable's distribution to a theoretical distribution
- The `probplot()` function is a straightforward way of creating a q-q plot
- The function takes the data and comparison distribution as an argument

```
1 # create a sample of the same size as the salary dataset
2 # from the standard normal distribution
3 norm_sam = np.random.normal(0, 1, salary.shape[0])
4
5 # How a sample drawn from a normal distribution may look
6 # The plot argument says to use matplotlib to build the plot
7 stats.probplot(norm_sam, dist="norm", plot = plt)
8 plt.show()
```



# Quantile Plots (Example)

```
1 import numpy as np
2 import scipy.stats as stats
3
4 plt.figure(figsize=(15, 8))
5 stats.probplot(salary.roe, dist="norm", plot=plt)
6 plt.show()
```

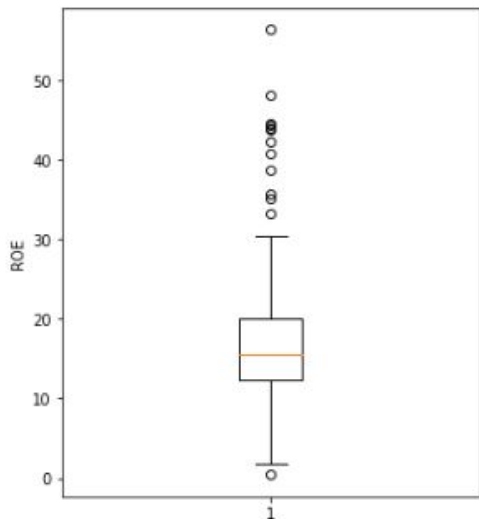




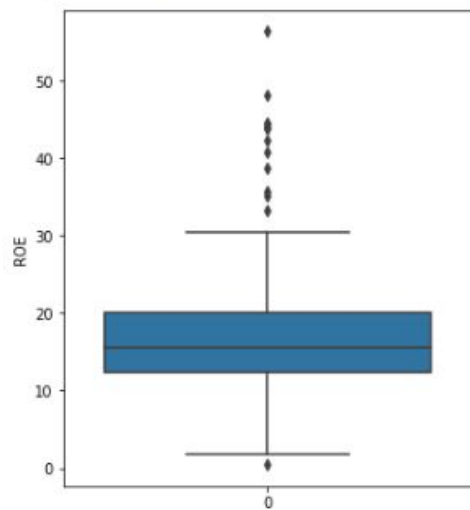
# Single Boxplots

- Boxplots helps us visualize the min, max, first, second, and third quartiles
- Matplotlib and Seaborn both have easy methods for creating boxplots
  - `sns.boxplot()`
  - `plt.boxplot()`

```
1 # matplotlib
2 plt.figure(figsize=(5,6))
3 plt.boxplot(salary.roe)
4 plt.ylabel("ROE")
5 plt.show()
```

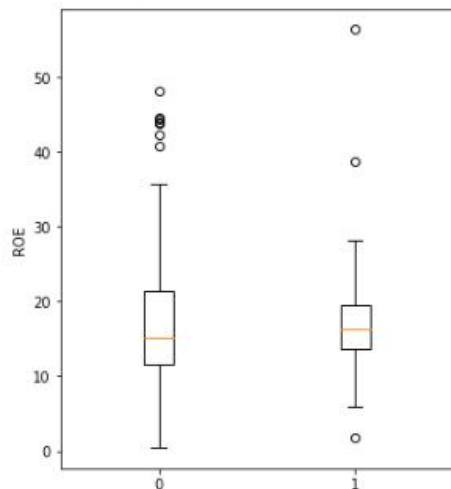


```
1 # seaborn
2 plt.figure(figsize = (5, 6))
3 sns.boxplot(data = salary.roe)
4 plt.ylabel("ROE")
5 plt.show()
```

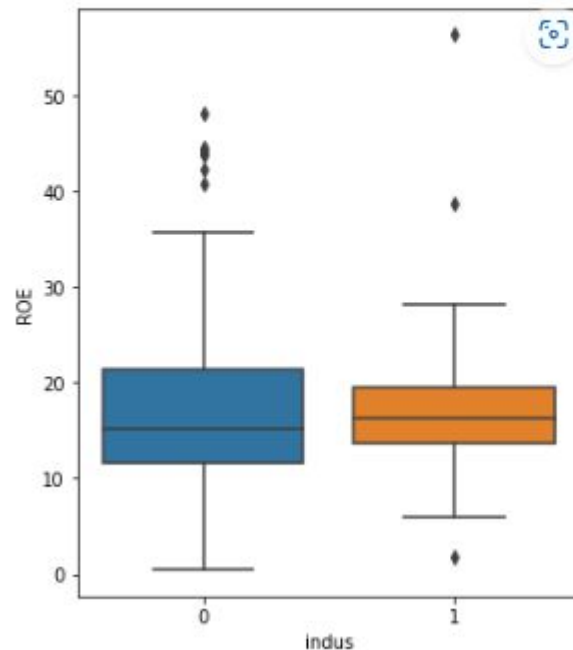


# Parallel Boxplots

```
1 # create a list containing the two series of people within different industries
2 data = [salary[salary.indus == 0].roe, salary[salary.indus == 1].roe]
3 plt.figure(figsize=(5,6))
4 plt.boxplot(data)
5 plt.ylabel("ROE")
6 plt.xticks([1,2], ["0", "1"])
7 plt.show()
```



```
1 # seaborn
2 plt.figure(figsize = (5, 6))
3 sns.boxplot(x = "indus", y = "roe", data = salary)
4 plt.ylabel("ROE")
5 plt.show()
```



# Univariate Data Characterization Exercise

A data.frame with 173 observations on 10 variables:

1. Import and use the vote1 data from wooldridge
2. Describe the dataset and check for NA values
3. Generate a histogram of voteA using Freedman's and Diaconis binning
4. Use a qq plot to decide whether voteA is normally distributed
5. Create a boxplot and use it to identify potential outliers in voteA

- **state:** state postal code
- **district:** congressional district
- **democA:** =1 if A is democrat
- **voteA:** percent vote for A
- **expendA:** camp. expends. by A, \$1000s
- **expendB:** camp. expends. by B, \$1000s
- **prtystrA:** percent vote for president
- **lexpendA:** log(expendA)
- **lexpendB:** log(expendB)
- **shareA:**  $100 * (\text{expendA} / (\text{expendA} + \text{expendB}))$

```
1 import wooldridge as woo
2 import numpy as np
3 import matplotlib.pyplot as plt
```

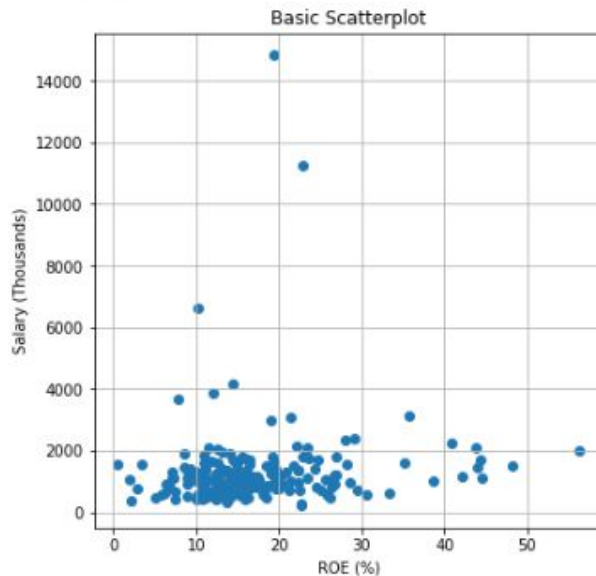
```
1 vote = woo.data('vote1')
2 vote
```

	state	district	democA	voteA	expendA	expendB	prtystrA	lexpendA	lexpendB	shareA
0	AL	7	1	68	328.295990	8.737000	41	5.793916	2.167567	97.407669
1	AK	1	0	62	626.377014	402.476990	60	6.439952	5.997638	60.881039

# Bivariate Data Characterization: Matplotlib Scatter Plot

- Pairwise associations can be plotted in scatterplots
- The `plt.scatter()` function takes two series of equal lengths as arguments
- Features all matplotlib customizations

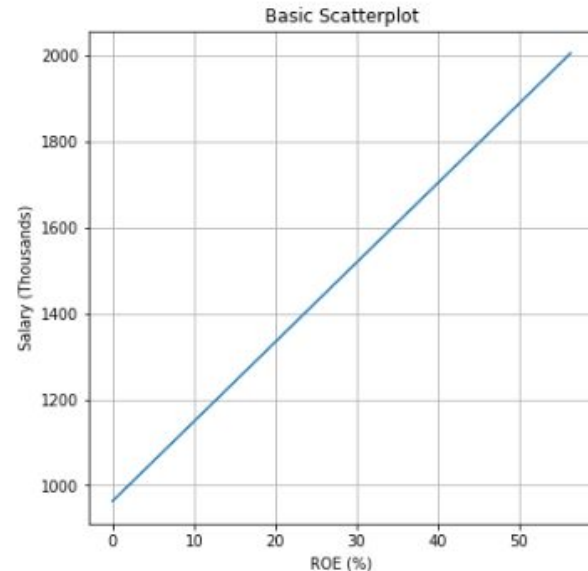
```
1 plt.figure(figsize = (6, 6))
2 # Create scatterplot between two variables
3 plt.scatter(salary["roe"], salary["salary"],)
4 plt.title("Basic Scatterplot")
5 plt.ylabel("Salary (Thousands)")
6 plt.xlabel("ROE (%)")
7 plt.grid()
```



# Regression Lines in Matplotlib

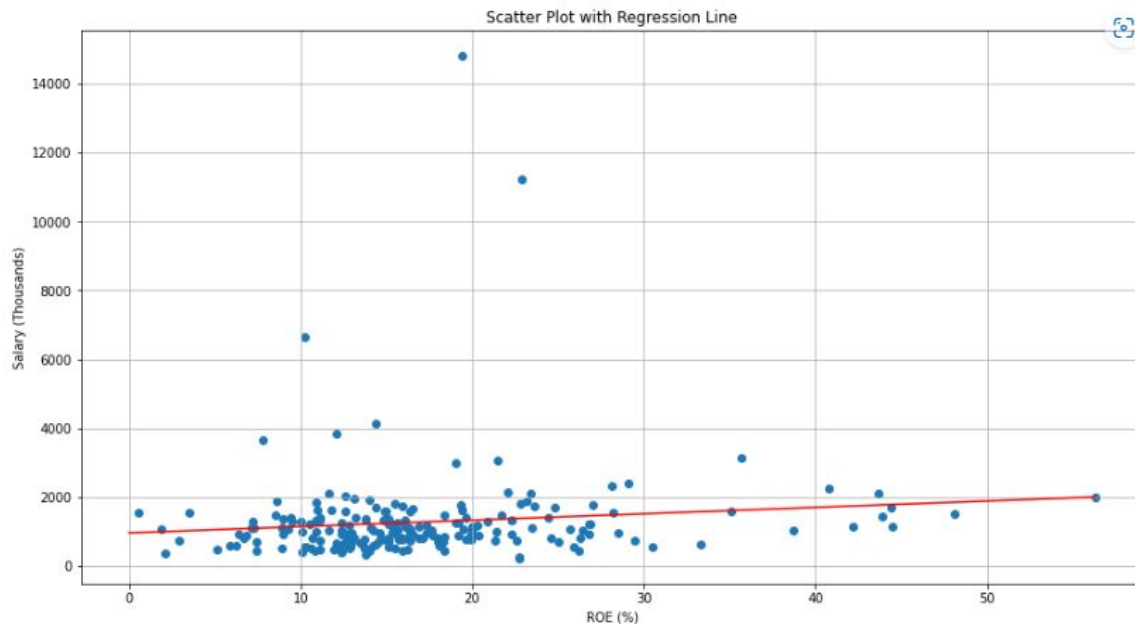
- Regression lines can be added to matplotlib scatter plots
- Need to estimate the regression coefficients (slope and intercept)
- `np.polyfit(x, y, deg = 1)` takes in arguments that define
  - X: The independent variable
  - Y: The dependent variable
  - Deg: The degree of polynomial being fit (one for a simple line)
- `polyfit()` returns an intercept (b) and slope coefficients (y)
- Then equally spaced points (x) are plotted against (mx+b)

```
1 plt.figure(figsize = (6, 6))
2
3 # Create a series of equally spaced values
4 x_range = np.linspace(0, salary.roe.max(), 100)
5
6 # Apply y=mx+b
7 plt.plot(x_range, m*x_range+b)
8 plt.title("Basic Scatterplot")
9 plt.ylabel("Salary (Thousands)")
10 plt.xlabel("ROE (%)")
11 plt.grid()
```



# Combining Regression Line and Scatter Plot

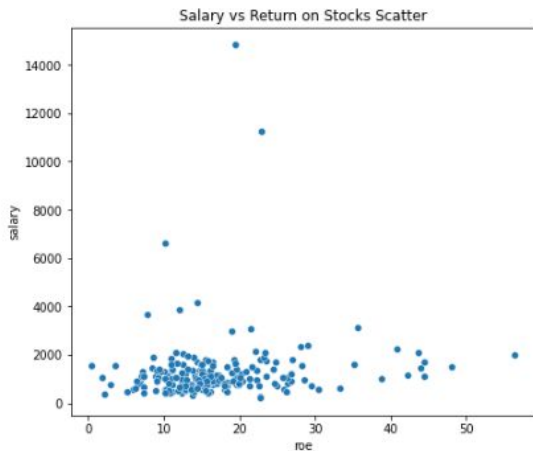
```
2  
3 # combining the two plots  
4 plt.scatter(salary["roe"], salary["salary"])  
5 plt.plot(x_range, m*x_range+b, color = "red")  
6 plt.title("Scatter Plot with Regression Line")  
7 plt.ylabel("Salary (Thousands)")  
8 plt.xlabel("ROE (%)")  
9 plt.grid()
```



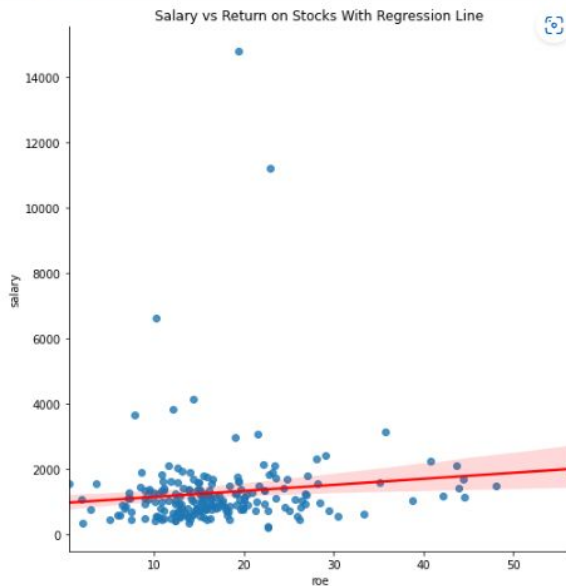
# Seaborn Scatterplot

- Similar plots can be produced easily in seaborn
- `scatterplot()` and `lmplot()` simplify their creation greatly
- In addition to the data and names of the x,y variables `regplot` also allows us to model nonlinear relationships with a lowess smoother

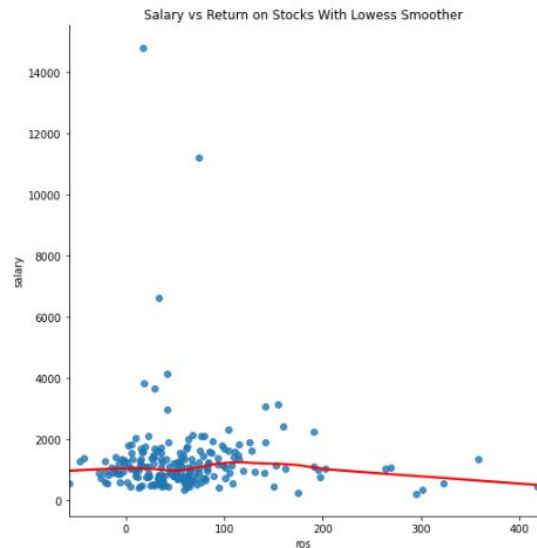
```
1 plt.figure(figsize = (7, 6))
2 sns.scatterplot(data = salary, x = "roe", y = "salary")
3 plt.title("Salary vs Return on Stocks Scatter")
4 plt.show()
```



```
1 # lmplot allows us to include different regression lines
2 sns.lmplot(data = salary, x = "roe", y = "salary",
3           line_kws= {"color": "red"}, height = 7, aspect = 1)
4 plt.title("Salary vs Return on Stocks With Regression Line")
5 plt.show()
```



```
1 # Lowess with return on stock
2 sns.lmplot(data = salary, x = "ros", y = "salary", lowess = True,
3           line_kws= {"color": "red"}, height = 7, aspect = 1)
4 plt.title("Salary vs Return on Stocks With Lowess Smoother")
5 plt.show()
```

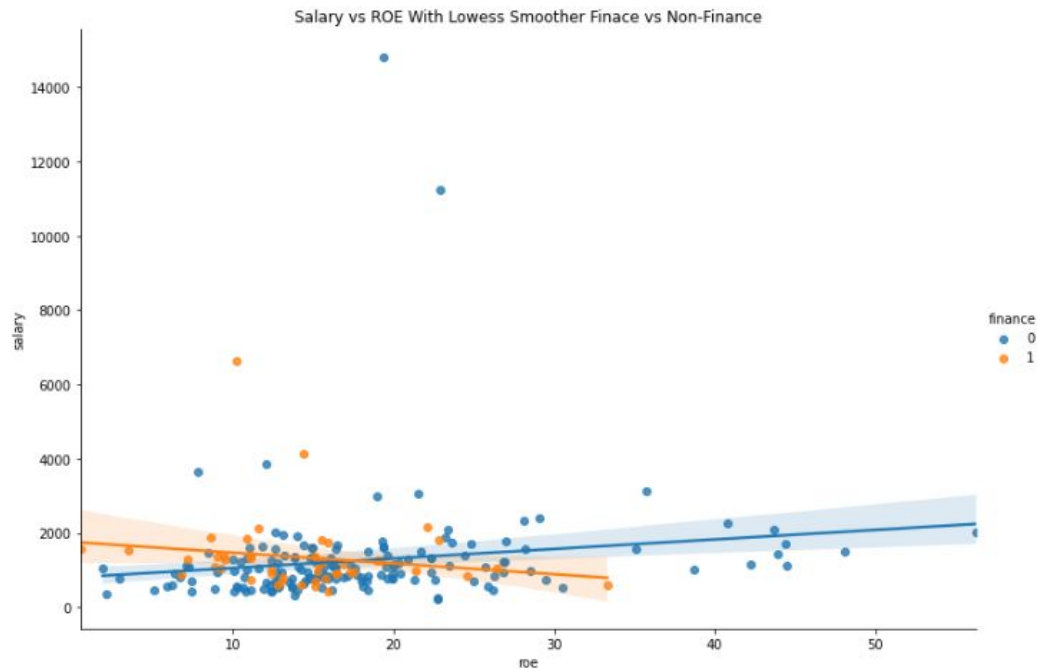


# Conditioned Scatterplot

- We can add a *hue = variable* argument that will condition our plot on a given variable
- Allows us to draw different regression lines for different groups
- This can help tell us whether any interactions between our variables occurs (e.g. the roe relationship is different for those in finance)

```
1 plt.figure(figsize= (10, 6))
2 sns.lmplot(data = salary, x = "roe", y = "salary",
3           hue = "finance", height = 7, aspect = 1.5)
4 plt.title("Salary vs ROE With Regression Line Finance vs Non-Finance")
5 plt.show()
```

<Figure size 720x432 with 0 Axes>





# Conditioned Scatterplot With Lowess Smoother

- An optional *lowess = True* argument may be passed to visualize non-linear relationships

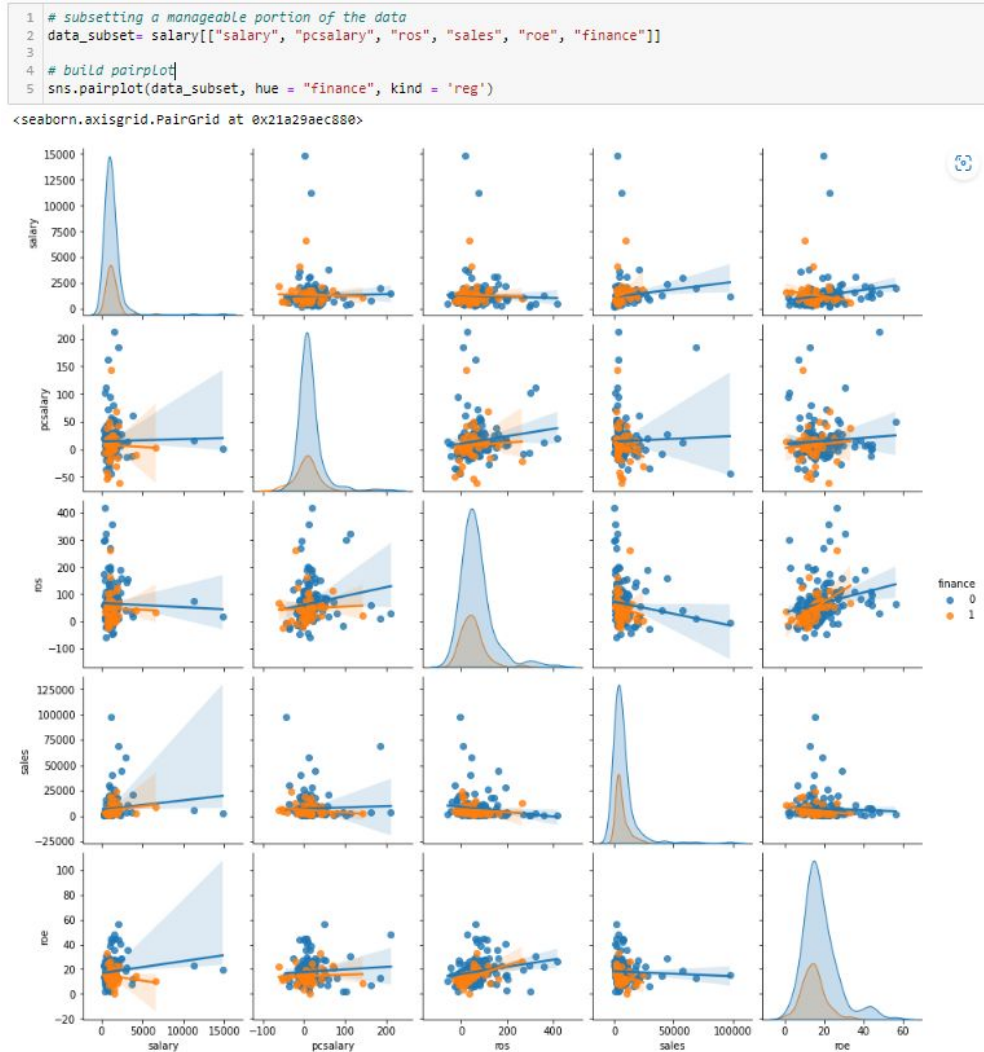
```
1 plt.figure(figsize= (10, 6))
2 sns.lmplot(data = salary, x = "roe", y = "salary",
3           hue = "finance", lowess = True, height = 7, aspect = 1.5)
4 plt.title("Salary vs ROE With Lowess Smoother Finace vs Non-Finance")
5 plt.show()
```

<Figure size 720x432 with 0 Axes>



# Pairplot

- Many different relationships may be visualized at once
- This can be helpful us for identifying potentially important relationships
- We can also condition on a variable to identify interactions
- We can also look at the conditional distributions of our variables in the density plots along the diagonal

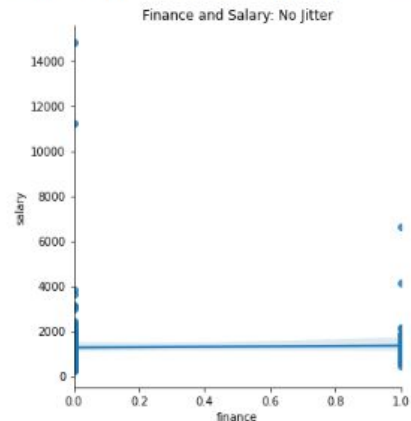


# Jittering Scatterplots

- Using categorical variables with scatterplots often means points will be stacked
- This can make the plots difficult to interpret
- “Jittering” adds random noise to your points to spread them out and make it easier to observe where observations are clustering
- Many seaborn plot types take *x\_jitter* = *float* and *y\_jitter* = *float* arguments that allow you to inject noise into your plot
- Higher numbers will add more noise

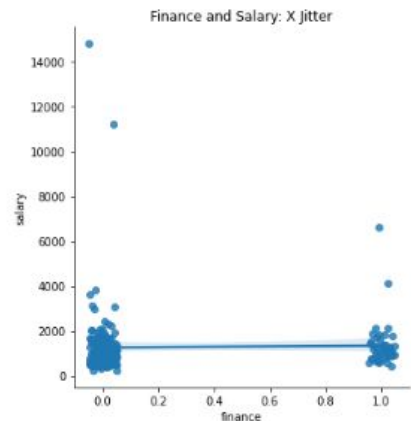
```
1 sns.lmplot(x = "finance", y = "salary", data = salary)
2 plt.title("Finance and Salary: No Jitter")
```

```
: Text(0.5, 1.0, 'Finance and Salary: No Jitter')
```



```
1 sns.lmplot(x = "finance", y = "salary", data = salary, x_jitter = .4)
2 plt.title("Finance and Salary: X Jitter")
```

```
: Text(0.5, 1.0, 'Finance and Salary: X Jitter')
```



# Bivariate Data Characterization Exercise

1. Import and use the vote1 data from wooldridge
2. Generate two scatterplots of  $x = \text{shareA}$  against  $y = \text{voteA}$  with regression lines:
  - a. No condition
  - b. Conditioning on democA
3. Generate a pairplot conditioning on DemocA

# Simple Linear Regression

- We are concerned with estimating the population parameters for each beta in a simple linear regression
- Helps us understand an association between two variables
  - There is almost never a causal interpretation
  - Requires x to be independent of the unobserved factors
- We can apply the simple linear regression to understand the association between roe and ceo salary

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

$$salary = \beta_0 + \beta_1 roe + u$$

# Simple Linear Regression Example (manual)

- Many modules are available for running regressions
  - The outputs will often also contain different diagnostics/metrics
  - Outputs may be compatible with other modules as well
- Linear regressions are simple to run manually as well

$$\hat{\beta}_0 = \bar{y} - \hat{\beta}_1 \bar{x}$$

$$\hat{\beta}_1 = \frac{\text{cov}(x, y)}{\text{var}(x)}$$

```
1 # subset the x and y variables
2 x = salary['roe']
3 y = salary['salary']
4
5 # calculate the covariance and variance
6 cov_xy = np.cov(x,y)[1,0] # returns a covariance matrix, we want the covariance between x and y
7 var_x = np.var(x, ddof = 1)
8
9 # Calculate coefficients
10 b1 = cov_xy/var_x
11 b0 = np.mean(y) - b1*np.mean(x)
12
13 print("The slope of the regression line is:", b1, "and the intercept is:", b0)
```

The slope of the regression line is: 18.50118634521492 and the intercept is: 963.191336472558

What is the interpretation of beta1? beta0?

# Statsmodels Linear Regression Setup

- Statsmodels has an even easier way to run a regression
- Import statsmodels.formula.api
- We specify a regression with arguments containing the data and formula
- The formula is specified by a string in the format:
  - 'y ~ x' where y is the dependent variable (salary) and x is the independent variable (roe)
- The calculation of the coefficients and other results is done when we call the fit() method in the second line
- This information is stored in results

```
import statsmodels.formula.api as smf
```

```
1 # syntax is model = sm.OLS('y~x', data = sample)  
2 reg = smf.ols('salary ~ roe', data= salary)  
3 results = reg.fit()
```

# Statsmodels Linear Regression Results

- The .params attribute allows us to access the estimated regression coefficients
- The values should match the manual calculation (differences may occur with rounding)

```
1 # syntax is model = sm.OLS('y~x', data = sample)
2 reg = smf.ols('salary ~ roe', data= salary)
3 results = reg.fit()
4 results.params
```

$\hat{\beta}_0$  → Intercept    963.191336  
 $\hat{\beta}_1$  → roe        18.501186  
dtype: float64

$$\widehat{salary} = 963.1913 + 18.50119 \cdot roe$$



# Plotting Regression Results With Statsmodels

- The results output can now be used to plot our regression line alongside our observations
- We can do this by supplying the regressor **roe** and the fitted values from the regression
- The fitted values are the y values predicted by our regression for each x value in our sample

```
# 1 # the fitted values attribute contains 209
# 2 # predicted y values for each observation
# 3 results.fittedvalues

: 0      1224.058071
  1      1164.854261
  2      1397.969216
  3      1072.348338
  4      1218.507712

    ...
204     1129.702014
205     1249.959725
206     1187.055698
207     1216.657586
208     1229.608413
Length: 209, dtype: float64
```

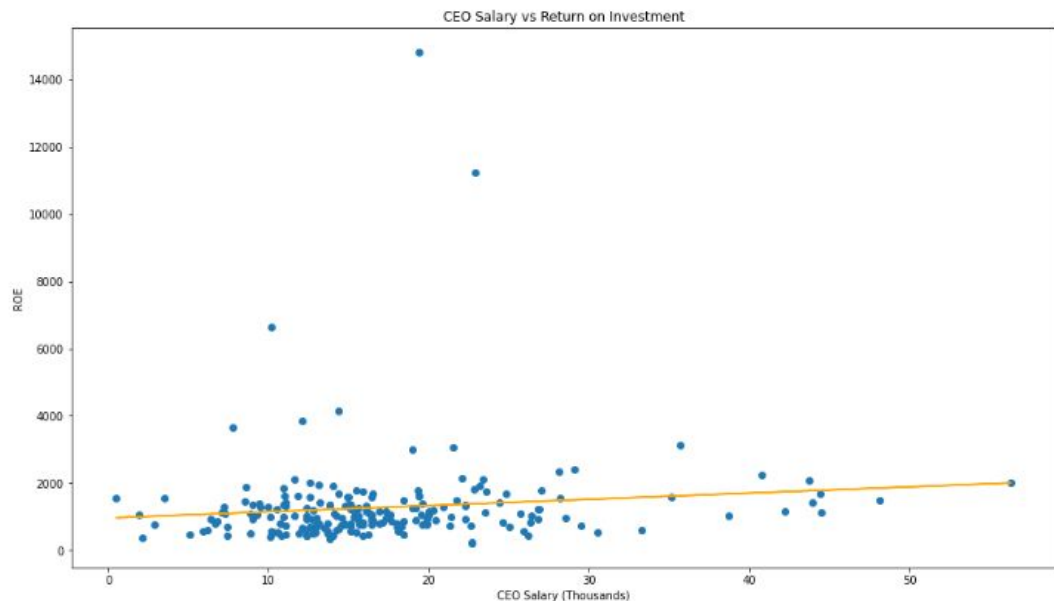
---

# Plotting Regression Results With Statsmodels (cont.)

- Plotting the fitted values against **roe** forms our regression line
- We also include our observed values in the scatterplot

```
1 plt.figure(figsize = (16, 9))
2
3 # plot the observed values
4 plt.scatter(salary['roe'], salary['salary'])
5
6 # plot roe vs the predicted or fitted values
7 plt.plot(salary['roe'], results.fittedvalues, color = "orange")
8
9 # add labels
10 plt.title("CEO Salary vs Return on Investment")
11 plt.xlabel("CEO Salary (Thousands)")
12 plt.ylabel("ROE")
13
14 plt.plot()
```

: []

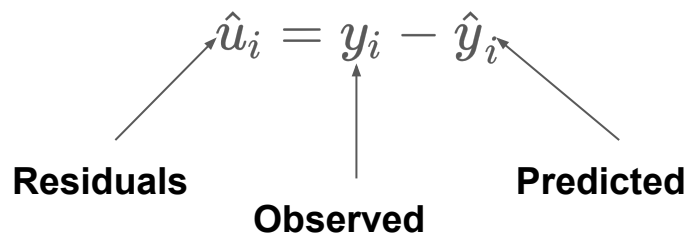


# Getting Residuals

- The regression residuals are defined as the deviations of our observations from the predicted values, i.e:

$$\hat{u}_i = y_i - \hat{y}_i$$

Residuals                      Observed                      Predicted



```
1 # calculate the fitted values from our estimated coefficients
2 y_hat = b0 + b1 * salary['roe']
3
4 # calculate the residuals
5 u_hat = salary["salary"] - y_hat
6
7 u_hat
```

```
0 -129.058071
1 -163.854261
2 -275.969216
3 -494.348338
4 149.492288
...
204 -199.702014
205 -724.959725
206 -529.055698
207 -661.657586
208 -603.608413
Length: 209, dtype: float64
```

```
1 reg = smf.ols('salary ~ roe', data= salary)
2 results = reg.fit()
3
4 # residuals are also stored in results from statsmodels
5 results.resid
```

```
0 -129.058071
1 -163.854261
2 -275.969216
3 -494.348338
4 149.492288
...
204 -199.702014
205 -724.959725
206 -529.055698
207 -661.657586
208 -603.608413
Length: 209, dtype: float64
```

# Properties of Residuals

- The following are algebraic properties of OLS statistics
- We now have the tools to confirm each of these properties in python

$$\sum_{i=1}^n \hat{u}_i = 0 \quad \Rightarrow \quad \bar{\hat{u}}_i = 0$$

$$\sum_{i=1}^n x_i \hat{u}_i = 0 \quad \Rightarrow \quad \text{Cov}(x_i, \hat{u}_i) = 0$$

$$\bar{y} = \hat{\beta}_0 + \hat{\beta}_1 \cdot \bar{x}$$

# Confirming the properties in Python

- Python does calculations with “double precision”, implying accuracy for up to 15 digits
- Therefore rounding errors will cause us to find our results are not *exactly* zero, but for all intents and purposes they *are* zero for propositions 1 and 2
- The third proposition is also confirmed by equality

```
1 # confirm property 1
2 prop1 = u_hat.mean()
3
4 # confirm property 2
5 prop2 = np.cov(x, u_hat)
6
7 #confirm property 3
8 y_bar_p3 = b0+b1*np.mean(x)
9 y_bar = np.mean(y)
```

```
1 print("The mean of our residuals is: ", prop1)
2 print("The covariance between x and u: ", prop2[0,1])
3 print("The mean of y is", y_bar, "and the mean calculated from the parameters is:", y_bar_p3)
```

The mean of our residuals is: -4.351649290779561e-15

The covariance between x and u: 4.197667854336592e-13

The mean of y is 1281.1196172248804 and the mean calculated from the parameters is: 1281.1196172248804

## Regression Exercise

1. Using the vote1 data from wooldridge
2. Using statsmodels.formula.smf estimate the regression parameters for:

$$voteA = \beta_0 + \beta_1 shareA + u$$

3. Plot the regression line against the observed values
4. Confirm the three algebraic properties of OLS statistics:

$$\sum_{i=1}^n \hat{u}_i = 0 \Rightarrow \bar{\hat{u}} = 0$$

$$\sum_{i=1}^n x_i \hat{u}_i = 0 \Rightarrow \text{Cov}(x_i, \hat{u}_i) = 0$$

$$\bar{y} = \hat{\beta}_0 + \hat{\beta}_1 \cdot \bar{x}$$