DATA SOCIETY®

The premiere data science training for professionals

Forecasting

"Prediction is very difficult, especially of the future."

-Niels Bohr

Outline: Intro to Regression

1. Single variable linear regression
2. Multiple linear regression
3. Model selection
4. Measuring variance and error
5. Dealing with outliers
6. Checking for model validity
7. Interactions
8. Regression with categorical variables

Measuring errors: variance

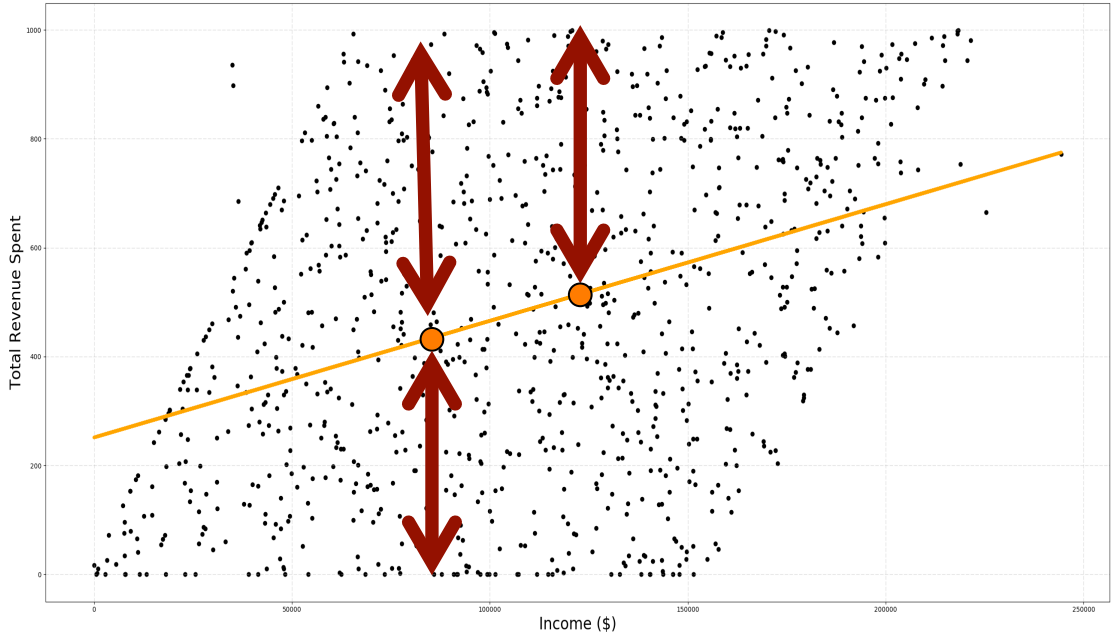
- Variance = $\frac{(\text{actual data point} - \text{expected data point})^2}{\text{Number of data points}}$ = average squared deviation from the mean

This can also be the average

- Variance is denoted by σ^2 "sigma" squared

- Variance = on average, how widely actual data is dispersed around [the predicted values, the mean, etc.]

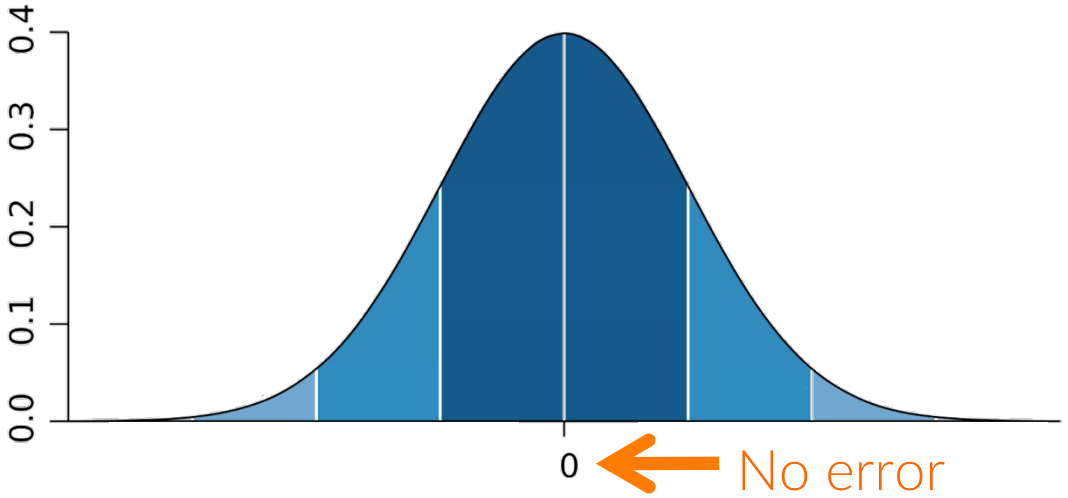
- *The higher the variance of the residuals the less accurate the model*

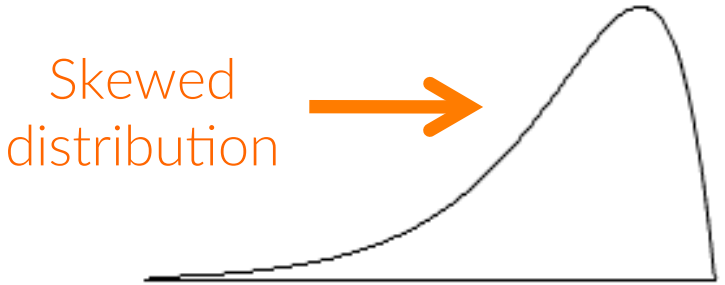


Measuring errors: randomness

"Normal distribution" of errors:

- In a non-biased model errors will be random
- If errors are not random it indicates that there is a "bias" in the model
- If errors are not random it means you're not taking something into account
- Random errors follow a bell curve
  - The bell curve is called a "normal" distribution



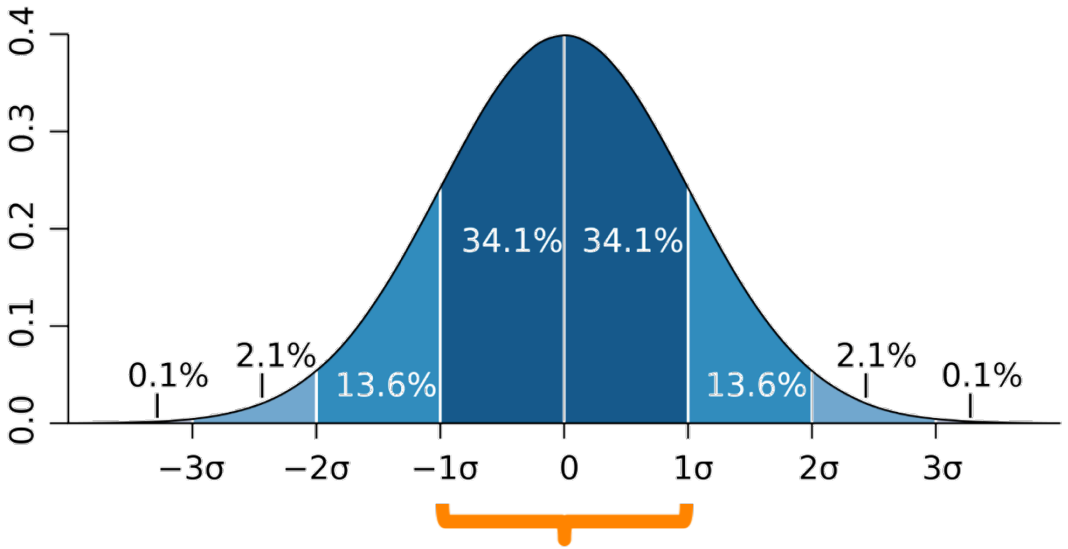


Measuring errors: standard deviation

"Normal distribution" of errors:

- Standard deviation = $\sqrt{\text{variance}} = \sigma$
- Standard deviation is a standardized measure of how dispersed data points are around the average or the expected value
- Standard deviation tells you what proportion of data points falls within a given range

The smaller the standard deviation of the residuals the more accurate the model



- 68.2% of errors are within 1σ away from the average or best fit line
- 95.4% of errors are within 2σ
- 99.6% of errors are within 3σ

Measuring errors: certainty

- Understanding standard deviation tells you
  the likelihood that a value will be within a given range!

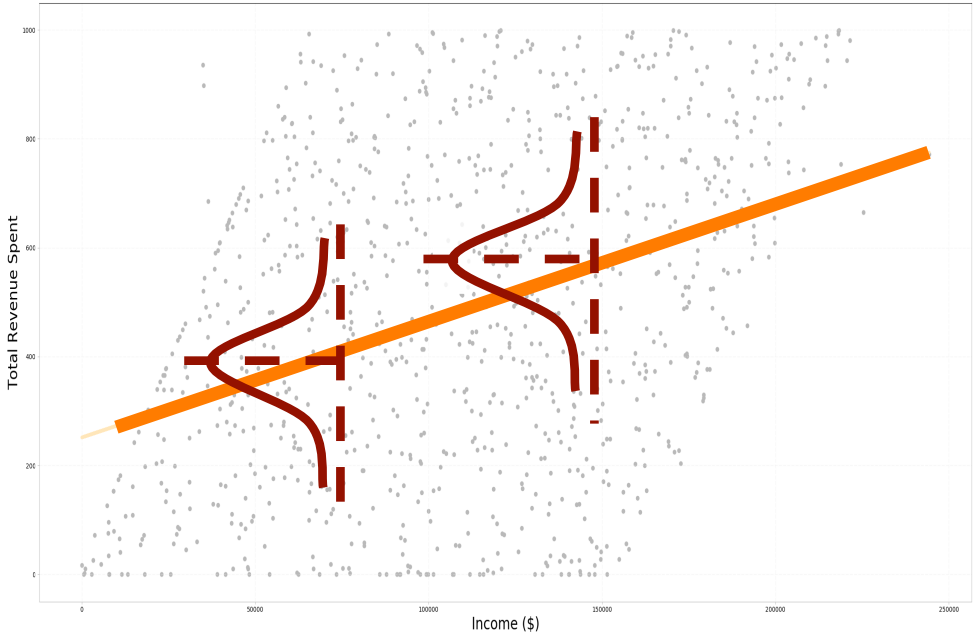
- A 95% confidence interval is within 2σ away from the average
  (or expected value)

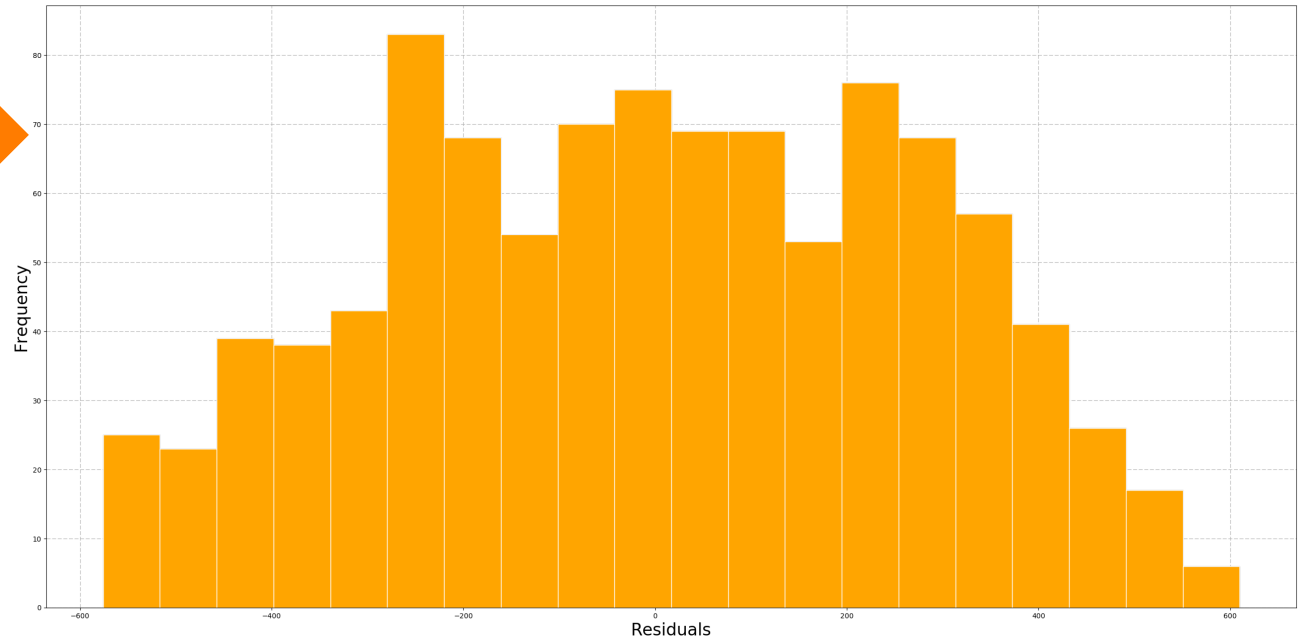
- The objective of a good model: decrease the standard deviation of the residuals
  - Put another way, to explain as much of the variance as possible

Understanding residuals

- We can check that residuals are normally distributed by plotting them as a histogram: no bias in our model!

Pooling the
residuals together





Plot the residuals

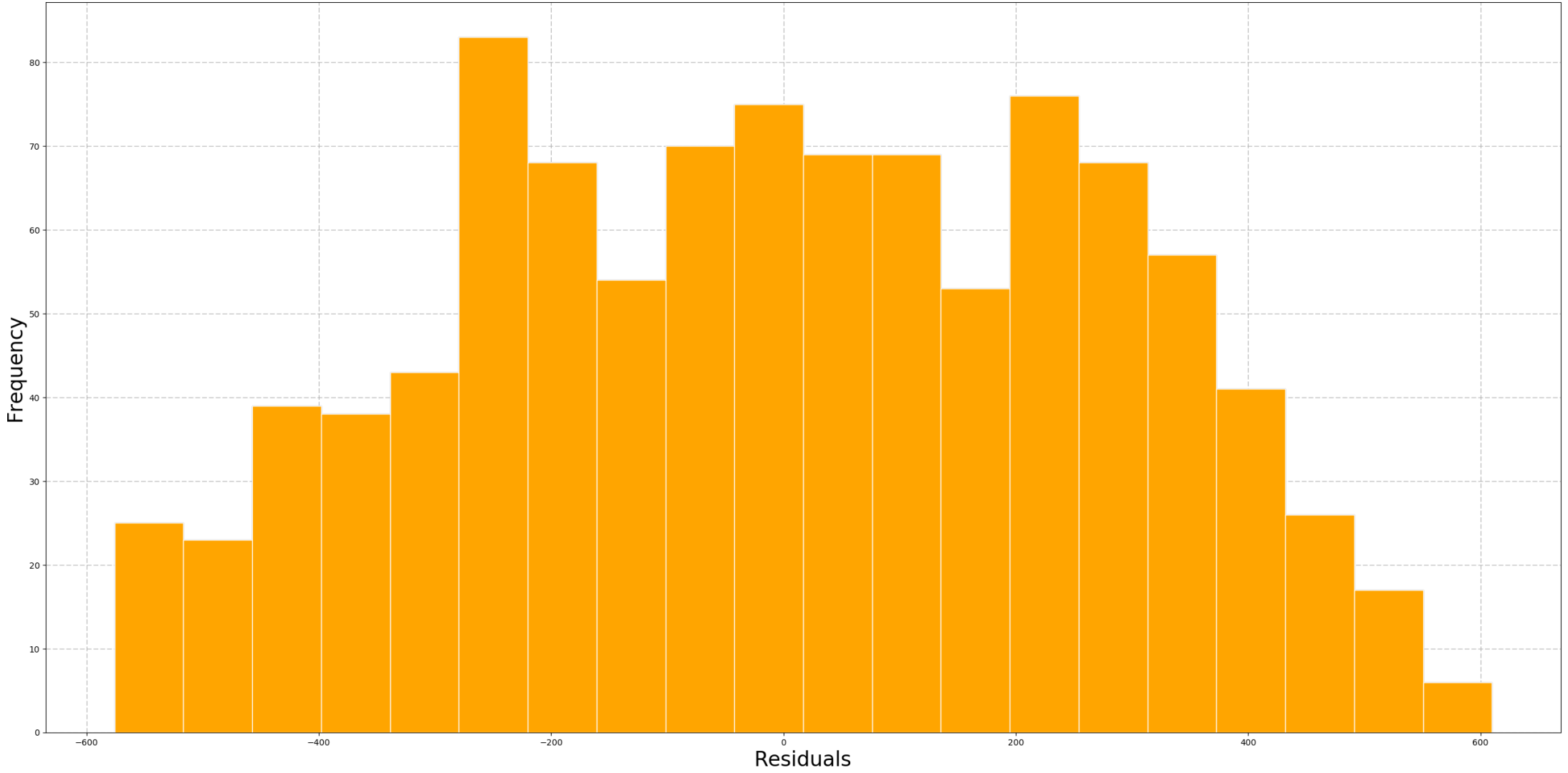
Script

```
# Create a new variable to represent residuals
slr_rev_income_stat_resid = slr_results.resid

# Plot the distribution of the residuals
plt.hist(slr_rev_income_stat_resid,
         histtype = 'bar',
         bins = 20,
         ec = 'white',
         color = 'orange',
         zorder = 3)
plt.xlabel('Residuals', fontsize = 24)
plt.ylabel('Frequency', fontsize = 24)
plt.grid(.25, linestyle = 'dashed', zorder = 0)
plt.savefig("lec2_slr_residuals.png", bbox_inches = 'tight')
```

1. Specify the number of bins
2. Make the outline of the bars white
3. Color the bars orange

4. X-axis label
5. Y-axis label

Plot the residuals



Calculate the variance of the residuals

Script

```
# var() method produces the variance of the dataset
slr_rev_income_stat_resid.var(ddof = 1) # ddof = degrees of freedom

# Let's check manually to see if we understand what Python is doing
slr_resid_avg = slr_rev_income_stat_resid.mean()
slr_resid_avg

# Subtract the average of the residuals from each residual
slr_resid_less_avg = slr_rev_income_stat_resid - slr_resid_avg

# Square this quantity, take the sum, divide by number of observations - 1
variance_check = ((slr_resid_less_avg**2).sum()) / (len(data) - 1)
variance_check

slr_rev_income_stat_resid.var(ddof = 1)
# Out[52]: 75185.41882196997

variance_check
# Out[53]: 75185.41882196997
```

Standard deviation of the residuals

Script

```
# The std() method calculates the standard deviation.
slr_rev_income_stat_resid.std()

# Take the square root of the variance to make sure you know what Python is
# doing. Note that there are many ways to do this.
import math
slr_rev_income_stat_resid.var(ddof = 1)**(1/2)
math.sqrt(slr_rev_income_stat_resid.var(ddof = 1))
np.sqrt(slr_rev_income_stat_resid.var(ddof = 1))
```

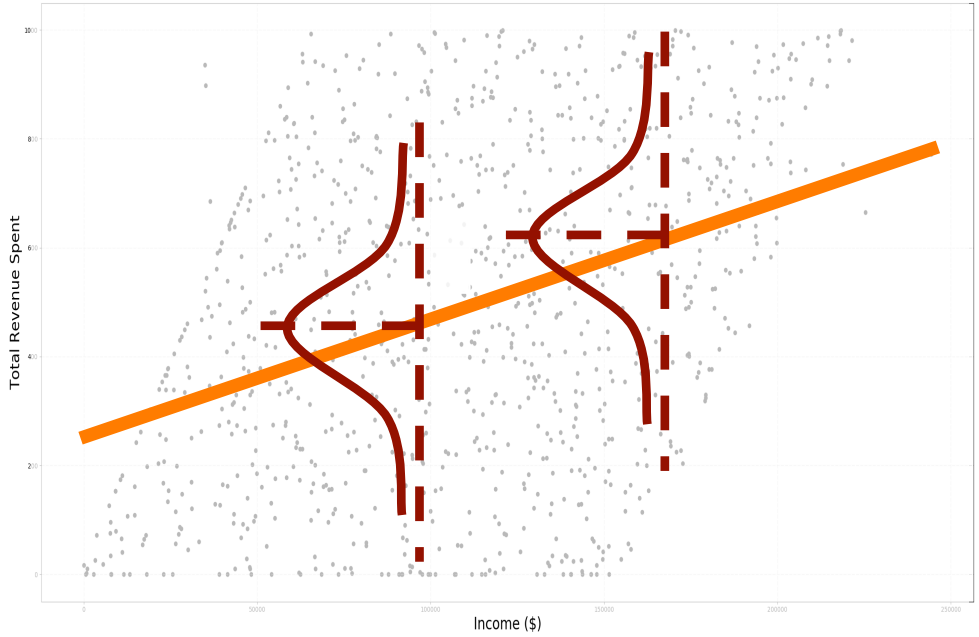
```
In [64]: slr_rev_income_stat_resid.std()
Out[64]: 274.19959668455016

In [65]: import math

In [66]: slr_rev_income_stat_resid.var(ddof = 1)**(1/2)
Out[66]: 274.19959668455016

In [67]: math.sqrt(slr_rev_income_stat_resid.var(ddof=1))
Out[67]: 274.19959668455016

In [68]: np.sqrt(slr_rev_income_stat_resid.var(ddof=1))
Out[68]: 274.19959668455016
```

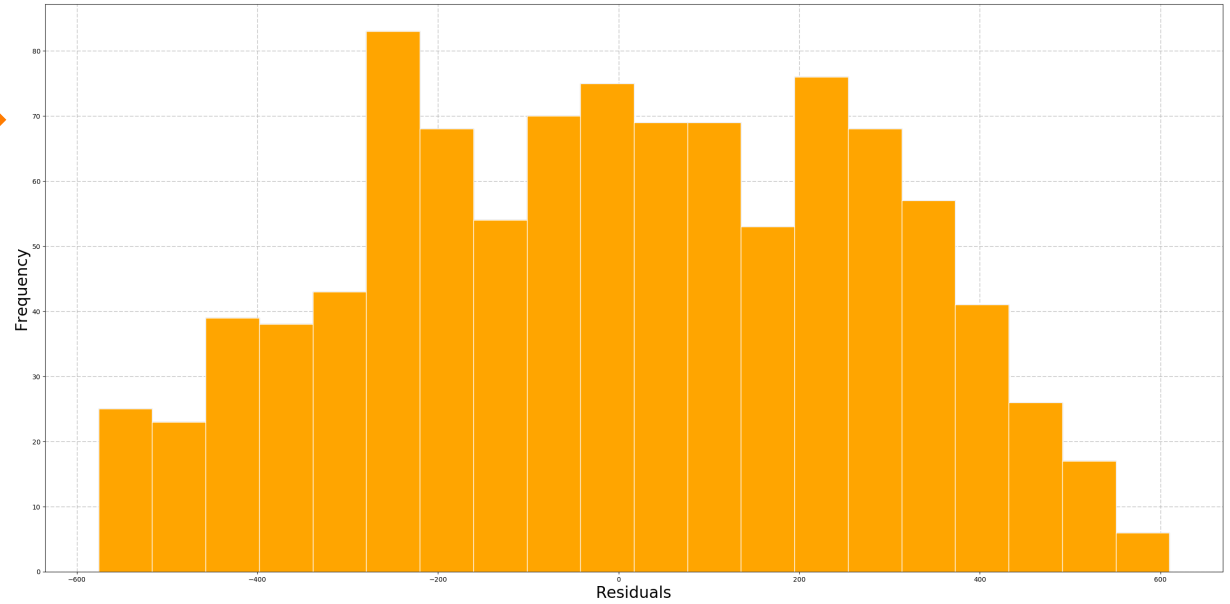
Exercise time!

Understanding residuals

- σ of total revenue spent: 296
- σ of income: 51,681
- σ of residuals: 274

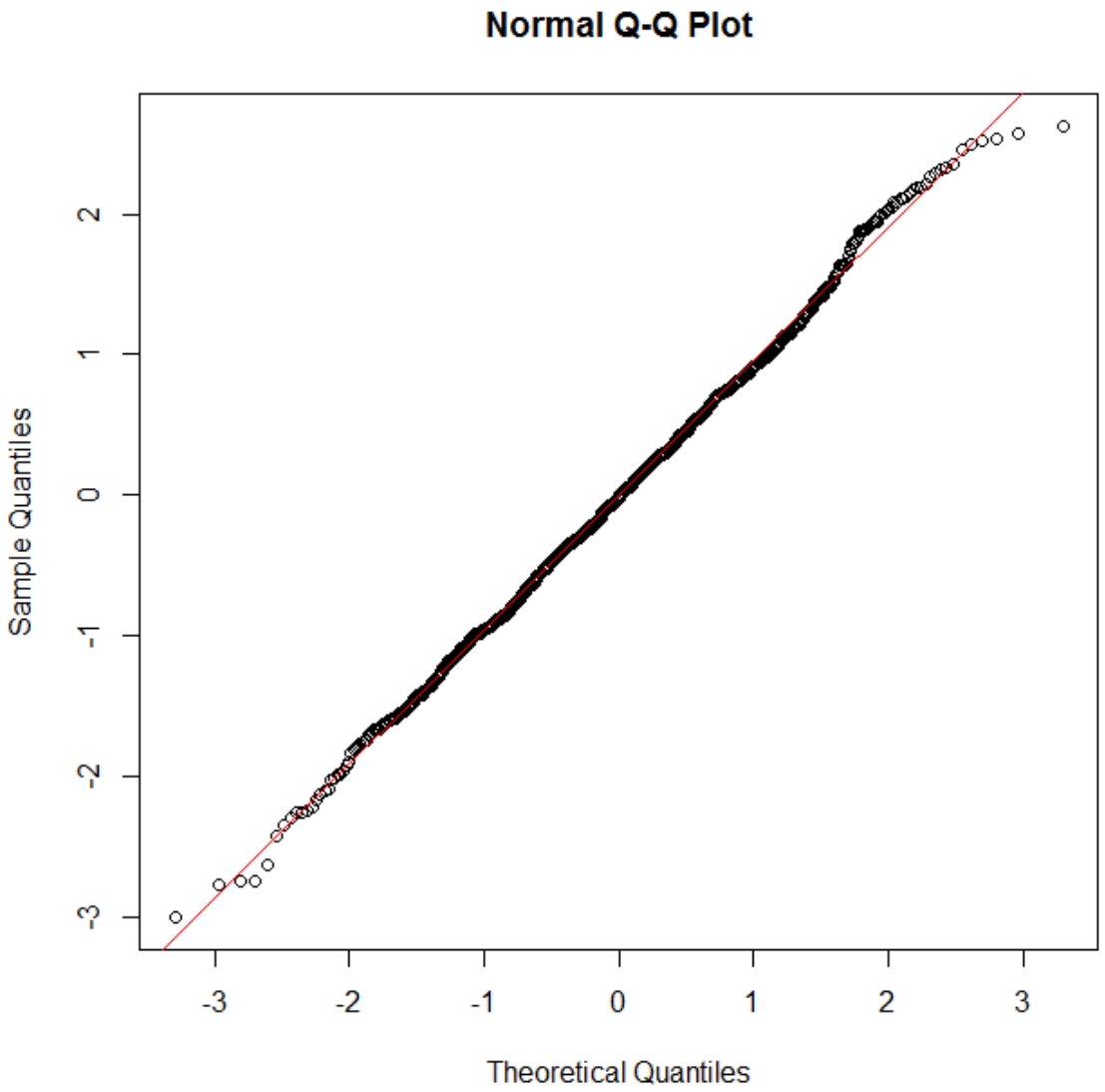
Pooling the residuals together





Quantile-quantile plot

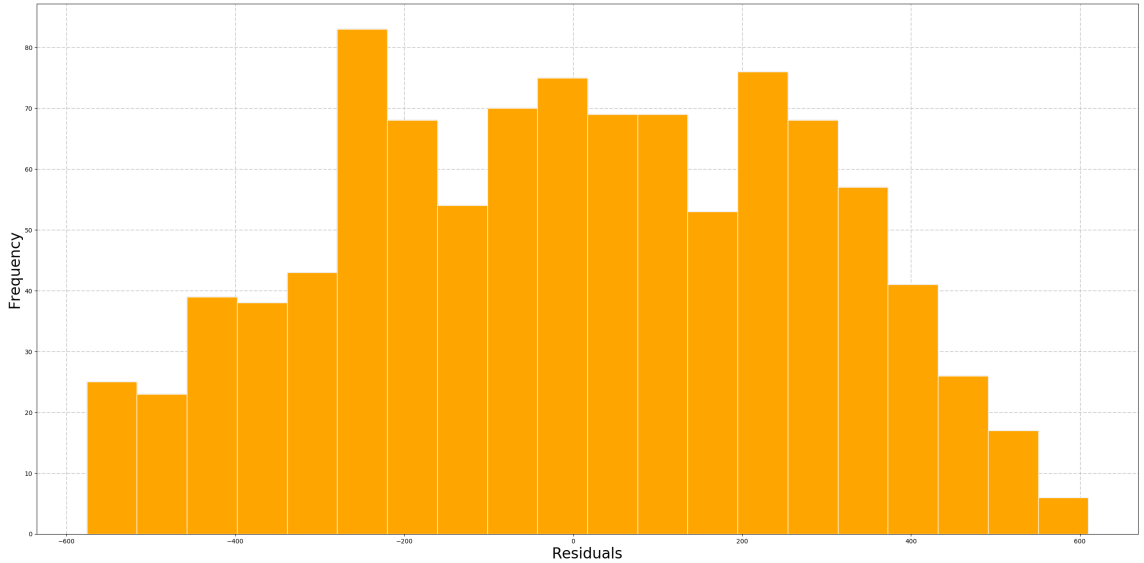
- Compares distributions between two sam
  normal distribution (standard deviation)
- This can be used to evaluate the residuals
  residuals to the expected normal distributi
- The closer the points follow the line, the n
- Can be used to compare samples from the
  world data, etc.

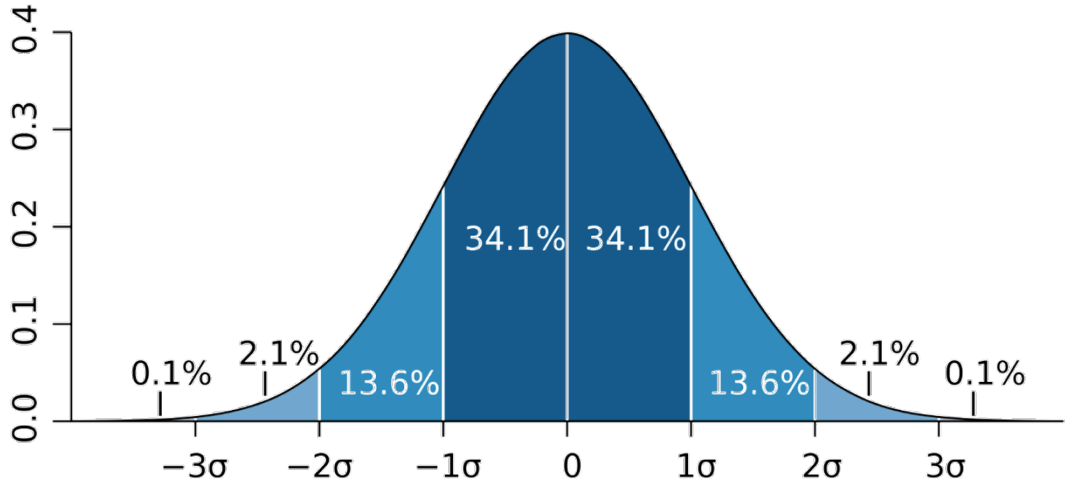


Quantile-quantile plot

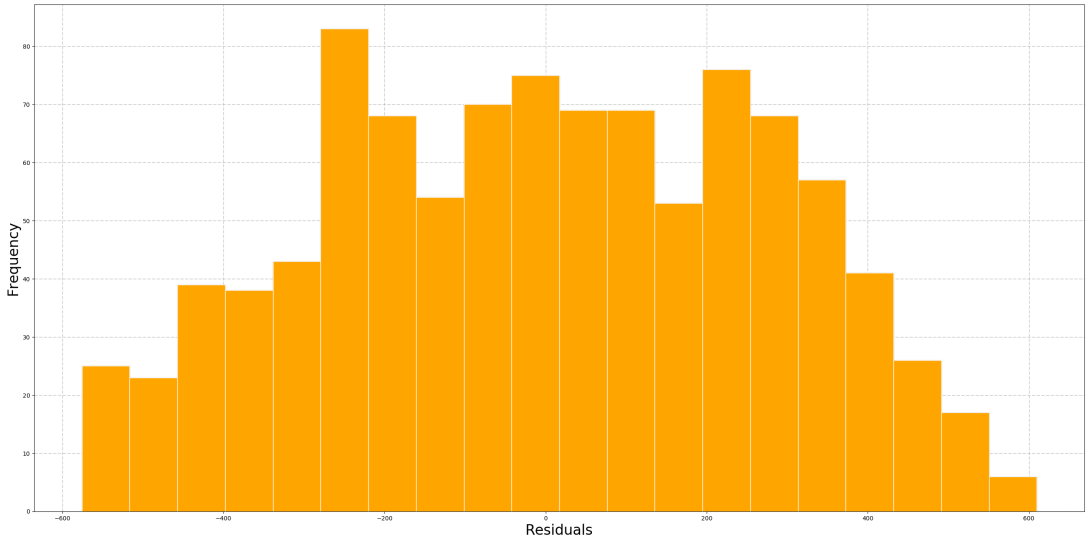
- Plots distribution of actual data vs. distribution of normally distributed data

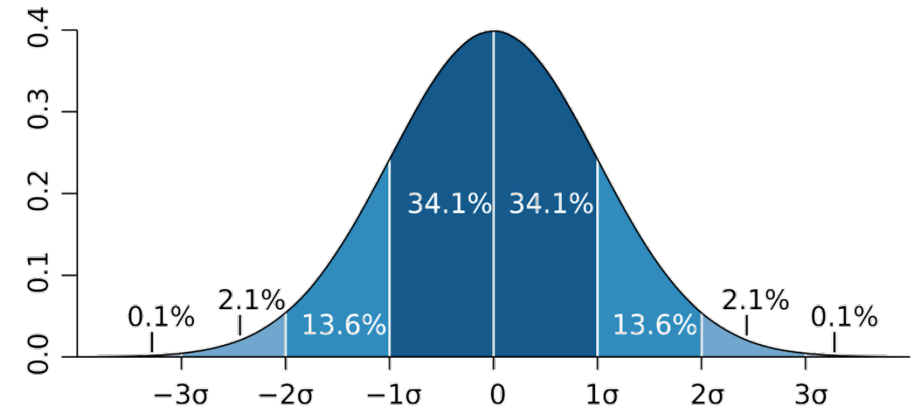
- Actual values are plotted on the y-axis and normally distributed values are plotted on the x-axis

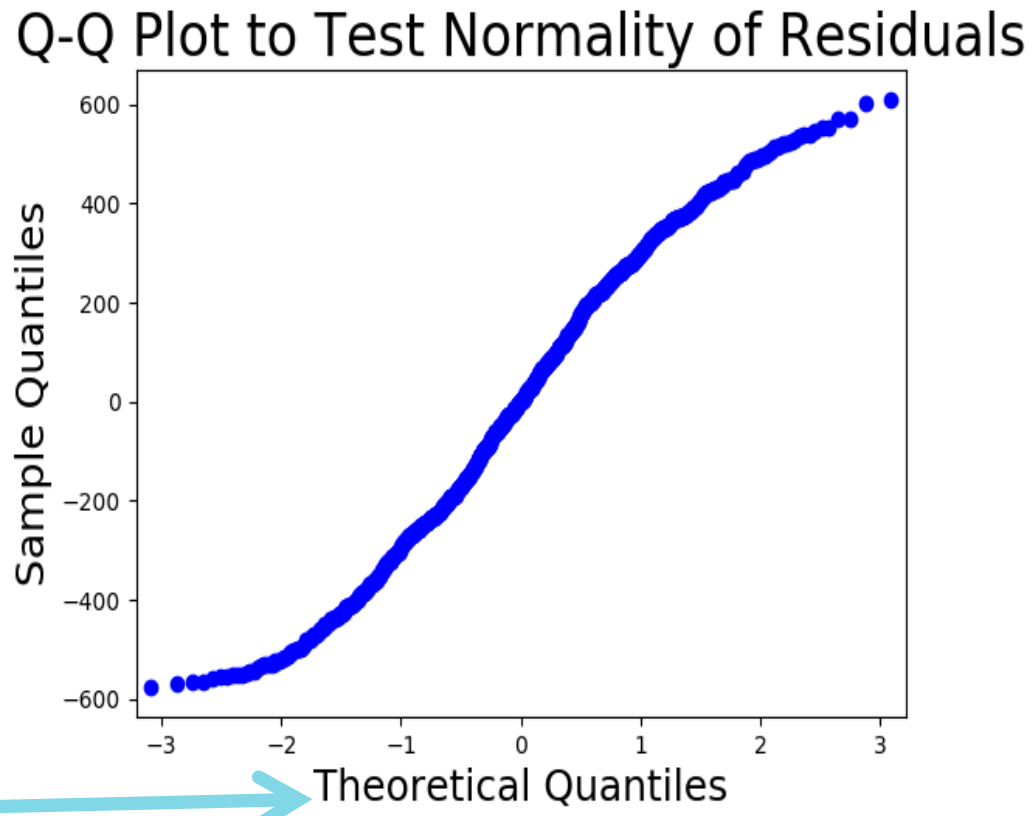




Quantile-quantile plot





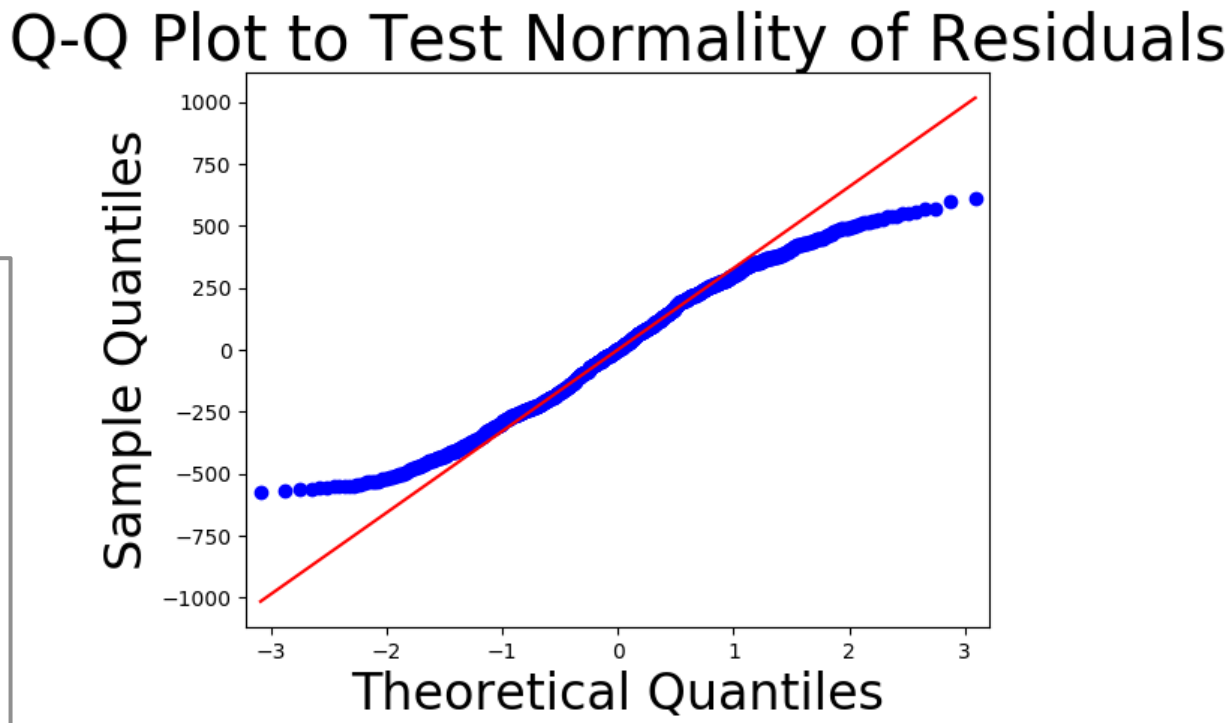


Q-Q plot: best fit line

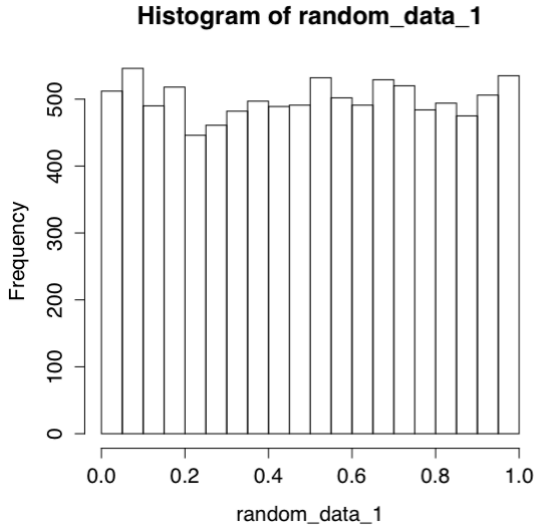
- To make the Q-Q plot easier to read, we can plot a best fit line through certain percentiles of the data, say 25th percentile and 75th percentile

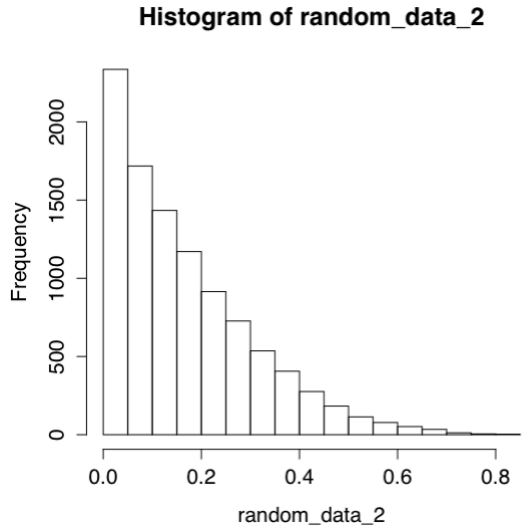
Script

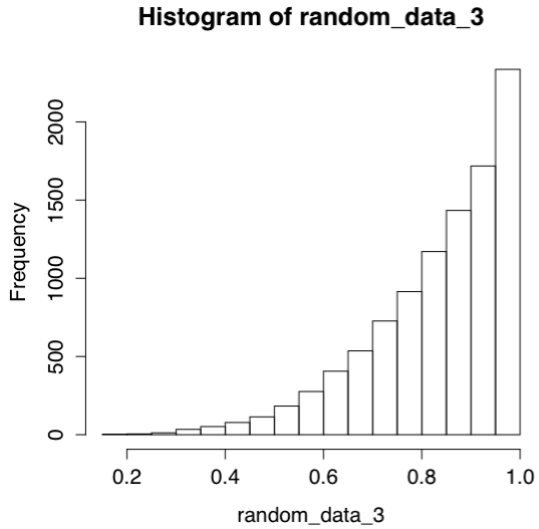
```
sm.qqplot(slr_rev_income_stat_resid,
          line = 'q')
plt.title('Q-Q Plot to Test Normality of
          Residuals', fontsize = 24)
plt.xlabel('Theoretical Quantiles',
           fontsize = 18)
plt.ylabel('Sample Quantiles',
           fontsize = 18)
plt.savefig("lec2_slr_qqplot_w_line.png",
            bbox_inches = 'tight')
plt.grid(.25)
```

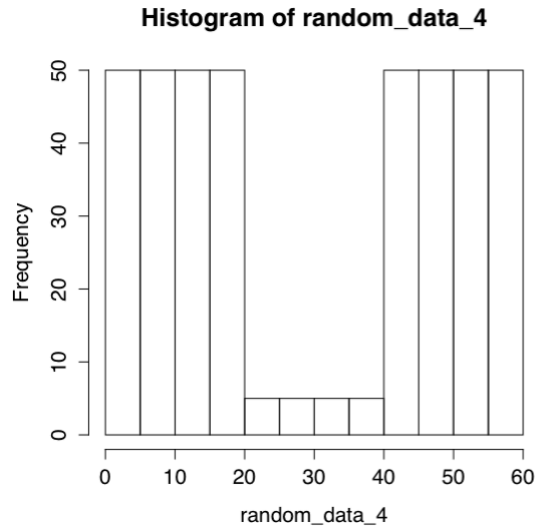


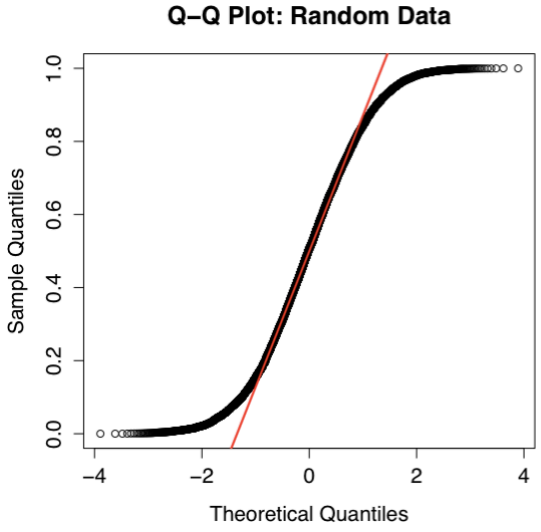
Abnormal Q-Q plots

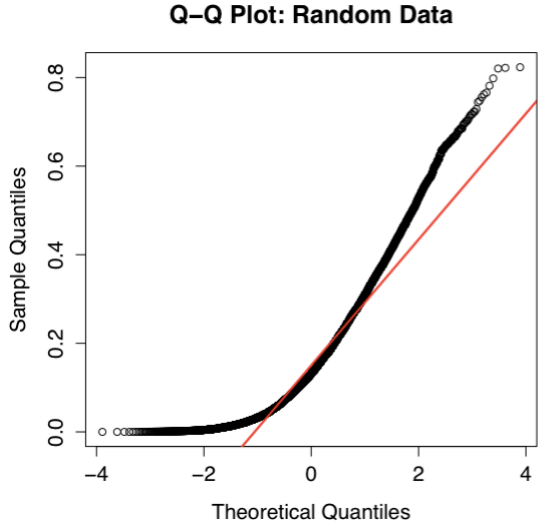


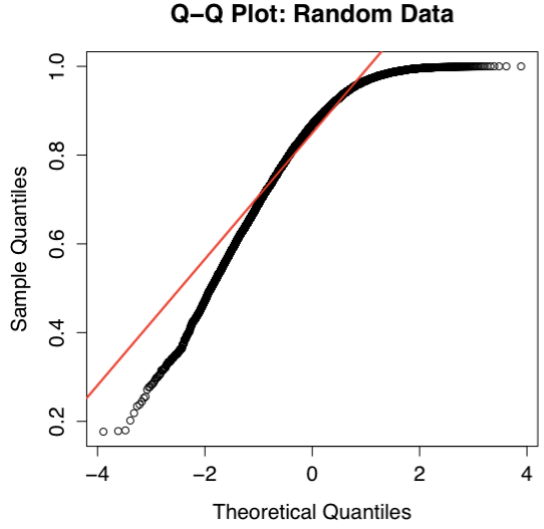


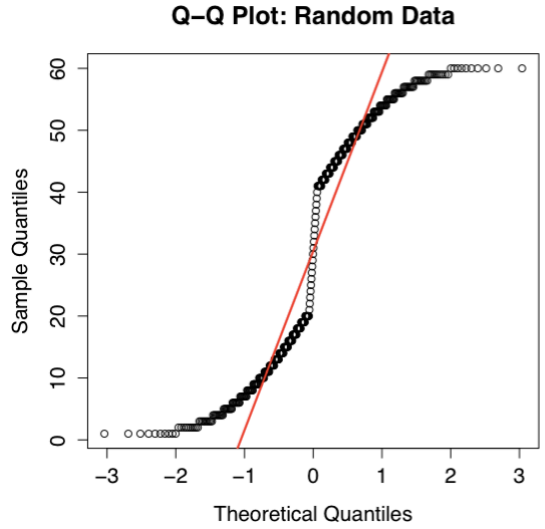












Standard error of a best fit line

- The standard error, or confidence interval of a regression line, tells you with a certain percentage certainty where the best fit line can be
- Recall that the best fit line is the "average expected value" of y for a given expected value of x
  - This value has a standard deviation as well that implies the distribution of the average expected value
- We won't go through the math here, but the arithmetic here is well understood

Explanation: http://stats.stackexchange.com/questions/44838/how-are-the-standard-errors-of-coefficients-calculated-in-a-regression

Confidence interval of regression model

Script

```
# Plot regression line and 95% confidence intervals with sns.lmplot (from seaborn)
sns.lmplot(x = 'Income',
           y = 'TotRevSpend',
           data = data,
           line_kws = {'color': 'orange',
                       'alpha': 5},
           scatter_kws = {'color': 'black',
                          'alpha': .15},
           ci = 95)
plt.xlabel('Income ($)', fontsize = 24)
plt.ylabel('Total Revenue Spent ($)', fontsize = 24)
```

1. Use linear regression
2. Make the best fit line orange
3. Thickness of the best fit line

4. Thickness of points on scatterplot
5. Degree of confidence (related to standard deviation concept)

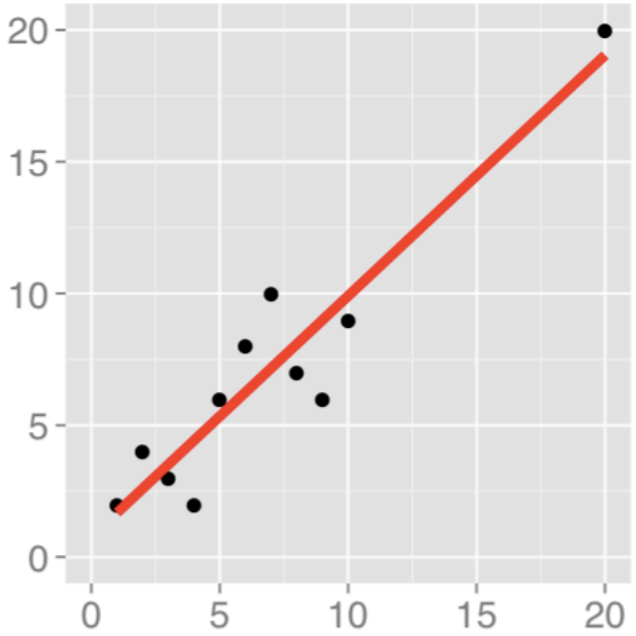
Exercise time!

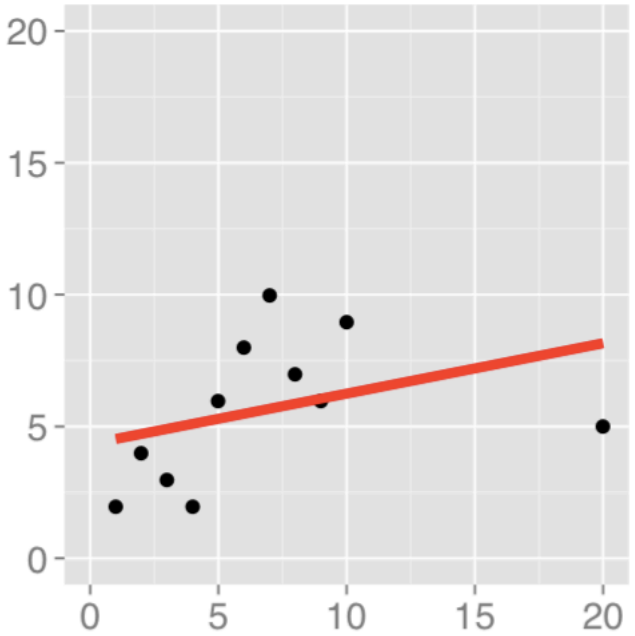
Outline: Intro to Regression

1. Single variable linear regression
2. Multiple linear regression
3. Model selection
4. Measuring variance and error
5. Dealing with outliers
6. Checking for model validity
7. Interactions
8. Regression with categorical variables

Outliers can spoil your regression model

- Outliers can have a very negative impact on linear regressions if they are not identified and handled properly:





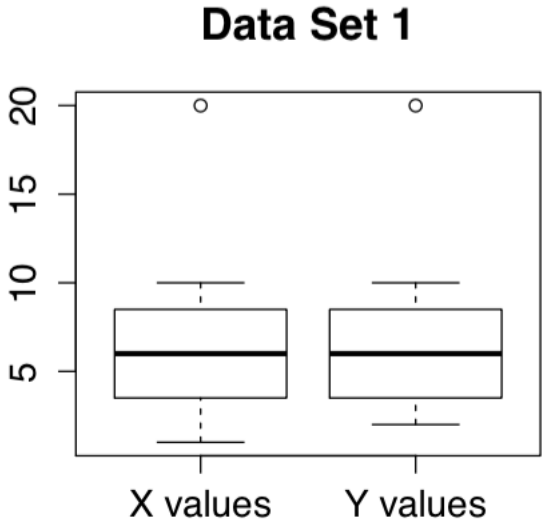
- 1 data point can completely change the predictions of your model

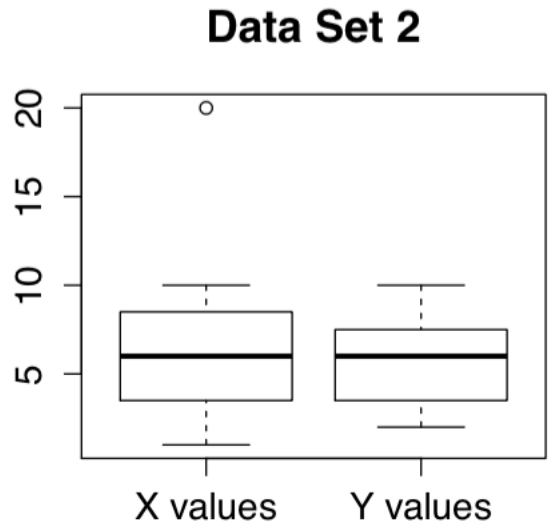
Identifying outliers

- Scatterplots
- Box-and-whisker plots
- Cook's distance
- Other methods exist and are covered in other courses

Box-and-whisker plots

- Box-and-whisker plots in R denote the median, 25th percentile and 75th percentile with the horizontal lines of the box
- The upper whisker is located at the smaller of the maximum x value and 75th percentile + 1.5 * IQR
- The lower whisker is located at the larger of the smallest x value and 25th percentile - 1.5 * IQR
- Outliers are represented by separate points



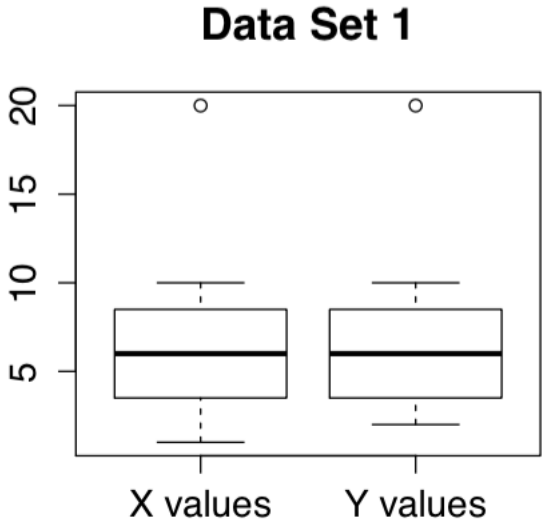


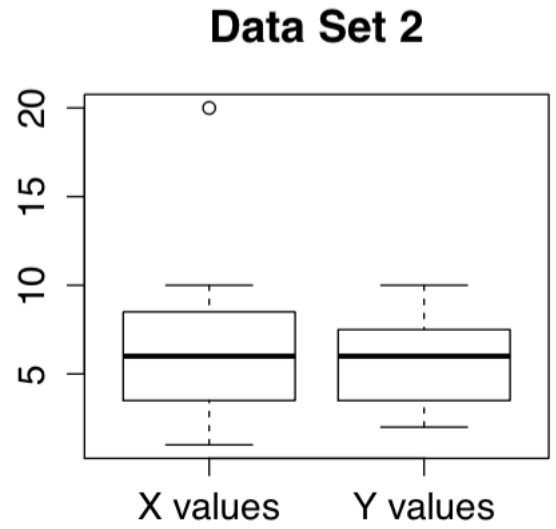
Box-and-whisker plots: IQR

- IQR = inter-quartile range
- IQR = 75th percentile

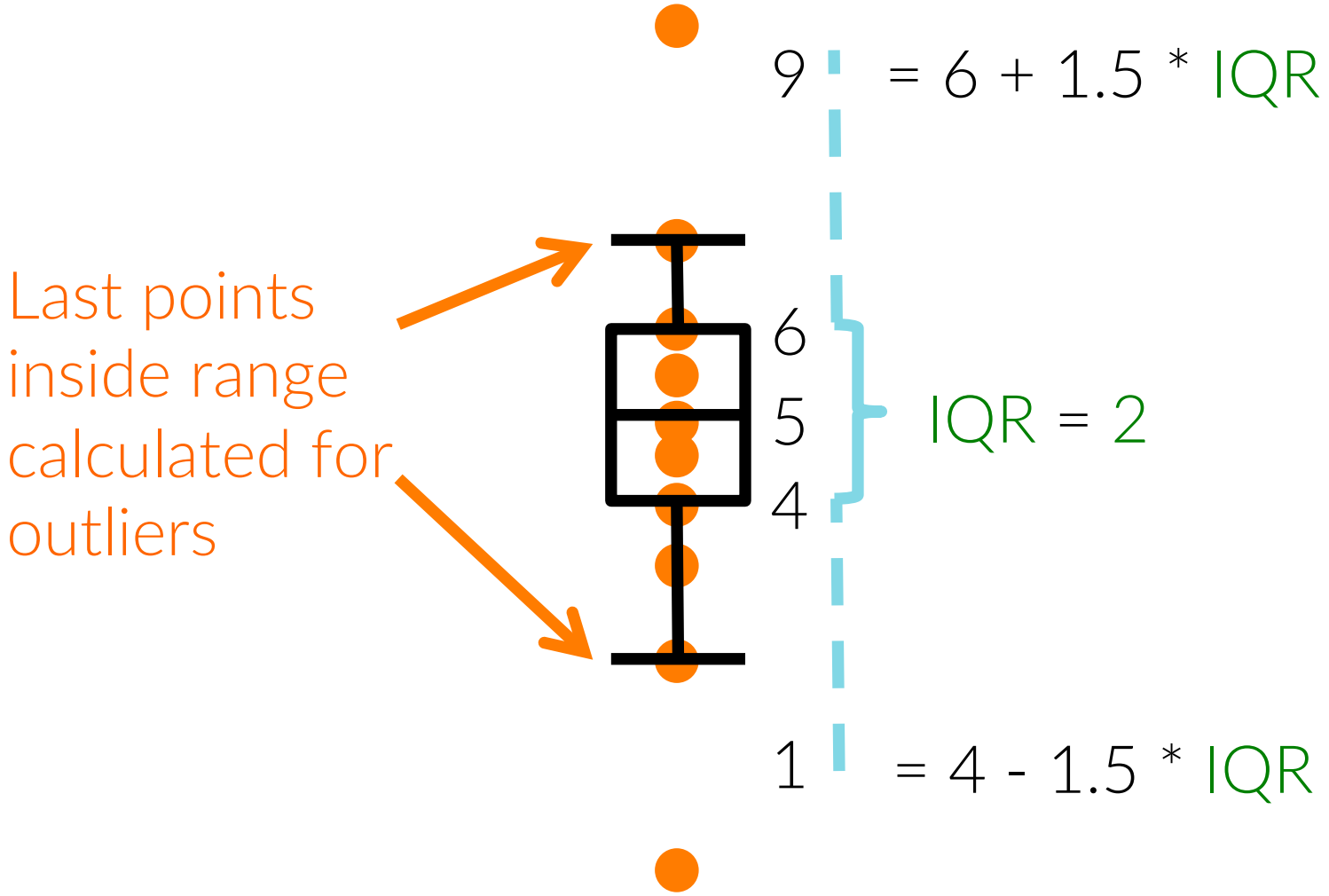
  –

  25th percentile
- When an outlier is detected, it is not included in the data set when calculating the values for the box limits and the whiskers





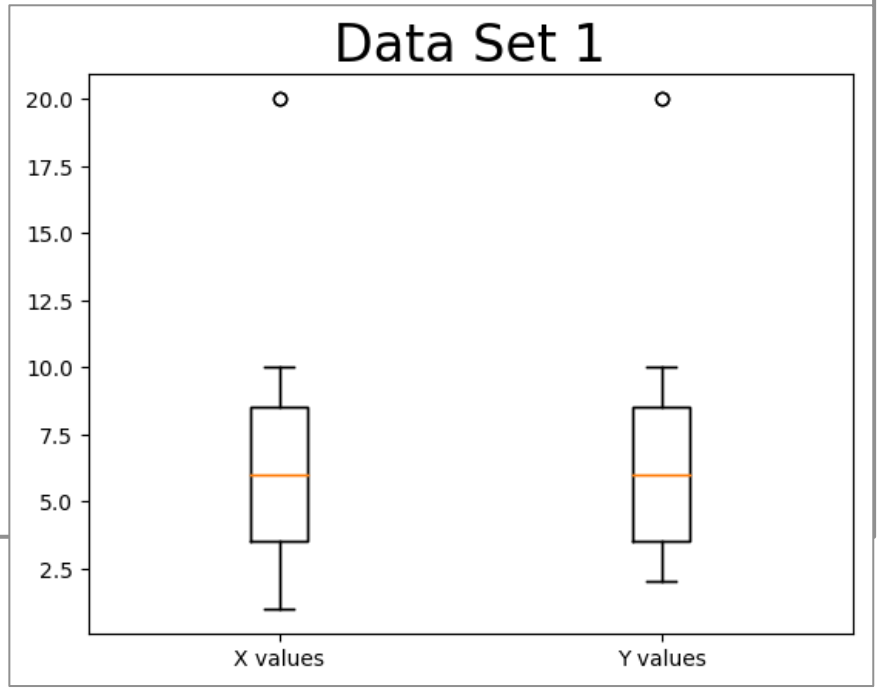
Box-and-whisker plots: example



Box-and-whisker plots

Script

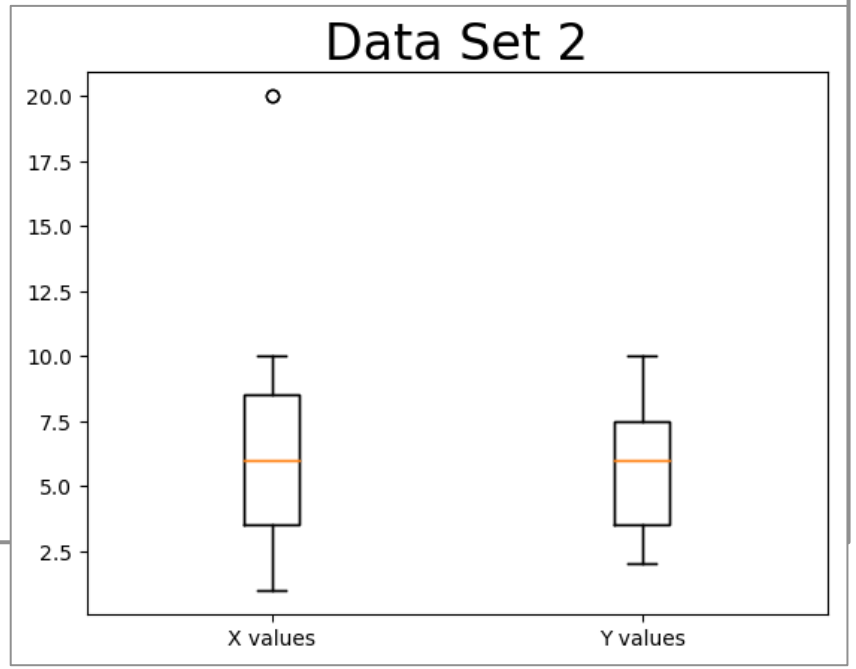
```
# Identify outliers with box-and-whisker plot.
outlier_data = np.column_stack(([1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 20],
                                [2, 4, 3, 2, 6, 8, 10, 6, 7, 9, 20]))

outlier_df = pd.DataFrame(outlier_data, columns = ['X values', 'Y values'])
bp = plt.boxplot(outlier_data)
plt.xticks([1, 2],['X values', 'Y values'])
plt.title('Data Set 1', fontsize = 24)

plt.savefig("boxplot1.png", bbox_inches = 'tight')

# Obtain data from the box plot
outliers = [flier.get_ydata() for flier in bp['fliers']]
outliers

# Out[88]: [array([20]), array([20])]
```



Box-and-whisker plots

Script

```
# Identify outliers with box-and-whisker plot for second data set.
outlier_data = np.column_stack(([1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 20],
                                [2, 4, 3, 2, 6, 8, 10, 6, 7, 9, 5]))

outlier_df2 = pd.DataFrame(outlier_data2, columns = ['X values', 'Y values'])
bp = plt.boxplot(outlier_data2)
plt.xticks([1, 2],['X values', 'Y values'])
plt.title('Data Set 2', fontsize = 24)

plt.savefig("boxplot2.png", bbox_inches = 'tight')

# Obtain data from the box plot
outliers = [flier.get_ydata() for flier in bp2['fliers']]
outliers2

# Out[95]: [array([20]), array([], dtype = int32)]
```



Cook's distance

- Measures the effect an observation has on a regression model
  - P_i = prediction from the full regression model for point i
  - P_{i2} = prediction from the regression model for point i if point i is excluded from the data
  - V = number of variables in the model
  - MSE = mean squared error of the regression model (mean square of the residuals)

- Cook's distance (CD) for point i:

$$CD_i = \frac{(P_i - P_{i2})^2}{V * MSE}$$

Cook's distance measures how the prediction of a point changes if that point is not included in the original data set.

Cook's distance

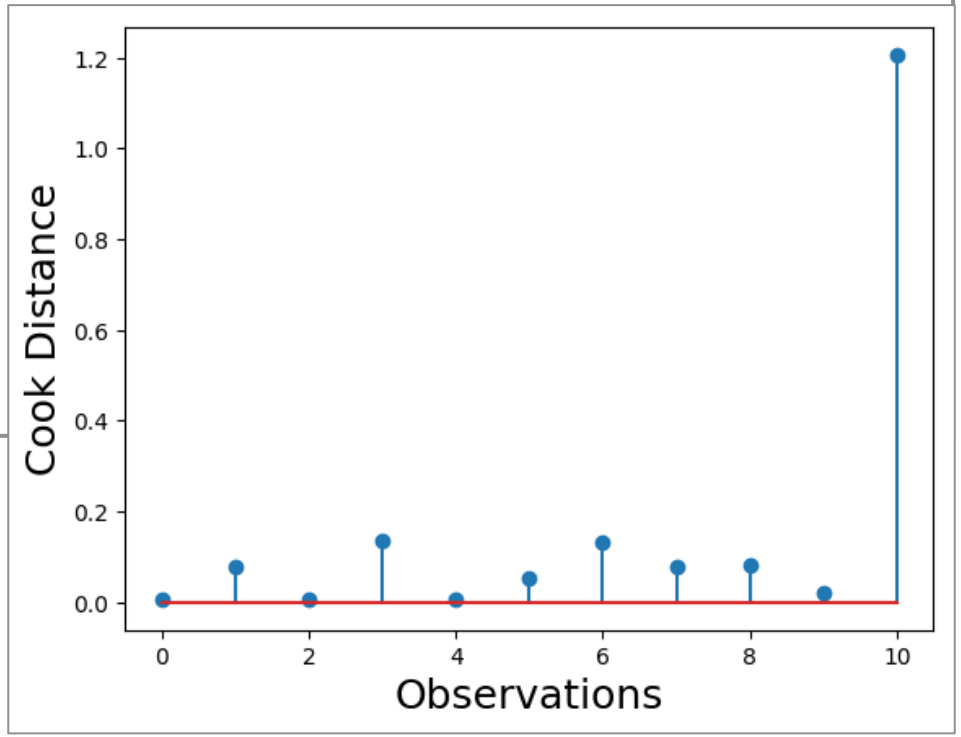
Script

```
# Create a sample data set.
outlier_data = np.column_stack(([1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 20],
                                [2, 4, 3, 2, 6, 8, 10, 6, 7, 9, 20]))

outlier_df = pd.DataFrame(outlier_data, columns = ['X values', 'Y values'])

# Reshape data for input into model
X_rs = outlier_df.iloc[:, 0].values.reshape((len(outlier_df), 1))
Y_rs = outlier_df.iloc[:, 1].values.reshape((len(outlier_df), 1))
X = outlier_df.iloc[:, 0]
Y = outlier_df.iloc[:, 1]

# Run a linear regression model
lm_outlier_data_stat = ols(formula = 'Y_rs ~ X_rs', data = outlier_df)
lm_outlier_fitted = lm_outlier_data_stat.fit()
influence = lm_outlier_fitted.get_influence()

# Outlier_data_CD is the distance and p is the p-value
(outlier_data_CD, p) = influence.cooks_distance
```

Cook's distance

Script

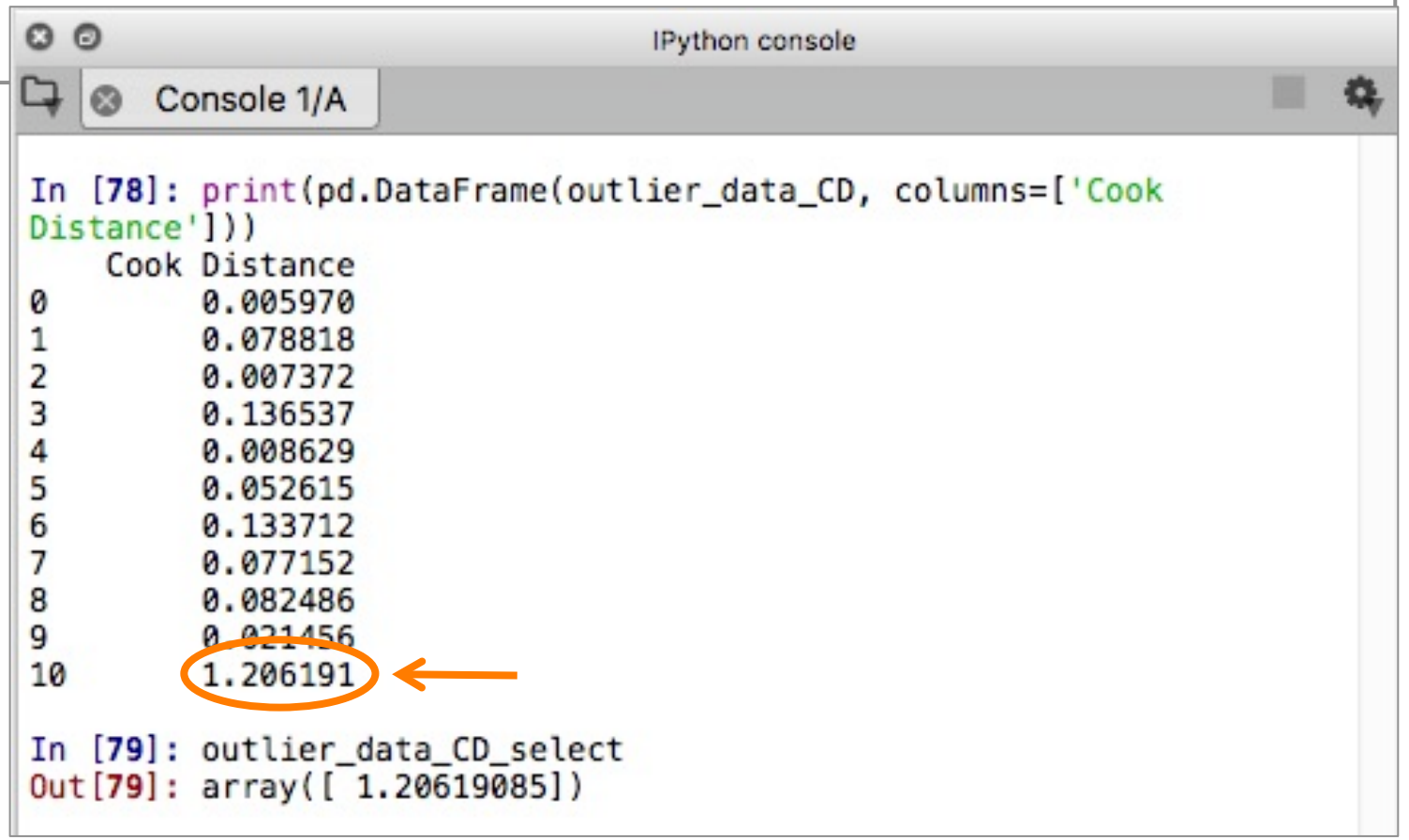
```
# A conventional cut-off point for Cook's distance is 4 / number of data points
# in the data set, all points where Cook's distance exceeds that value should
# be investigated as potential outliers
outlier_data_CD_select = outlier_data_CD[outlier_data_CD > 4 / 11]
outlier_data_CD_select

# Visualize Cook's distance
plt.stem(np.arange(len(outlier_data_CD)),
         outlier_data_CD)
plt.xlabel('Observations', fontsize = 18)
plt.ylabel('Cook Distance', fontsize = 18)
```



Cook's distance

Script

```
print(pd.DataFrame(outlier_data_CD, columns = ['Cook Distance']))

outlier_data_CD_select
```

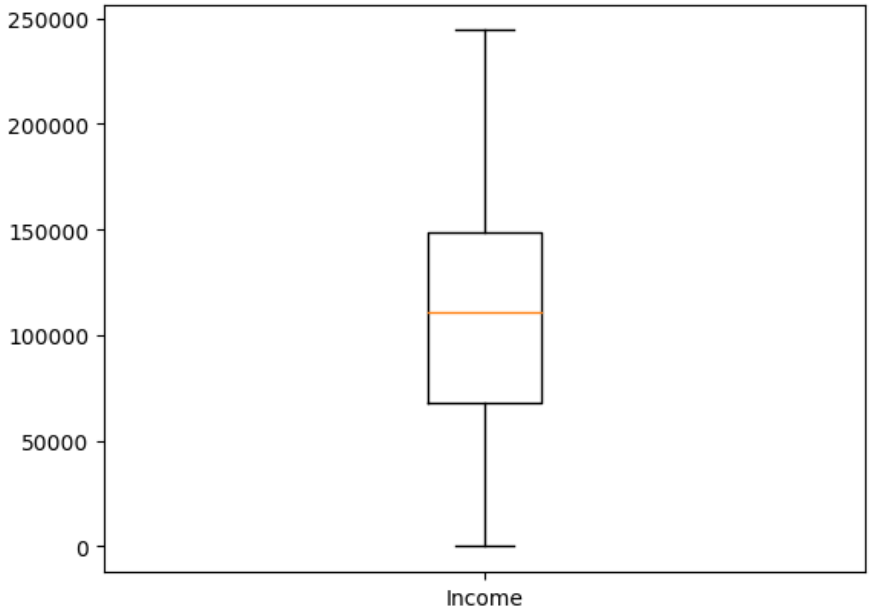


Outliers: NBA fan behavior

Script

```
# Let's see if our total fan revenue model comparing total revenue spent and
# fan income level has any outliers.

# Let's start with a boxplot.

# Matplotlib box plots can only take lists, tuples or numpy arrays
# not data frames.
rev_income_arr = np.array(data[['TotRevSpend', 'Income']])
plt.boxplot(rev_income_arr[:, 0:1])
plt.xticks([1], ['Total Revenue Spent']) # No outliers

plt.boxplot(rev_income_arr[:, 1:2]) # No outliers
plt.xticks([1], ['Income'])
```

Outliers: NBA fan behavior



Cook Distance

4	0.007575
50	0.006623
621	0.004109

Row number of data point

Cook's distance for each point

outliers_income
Out[32]: [array([], dtype=int64)]

outliers_rev
Out[30]: [array([], dtype=int64)]

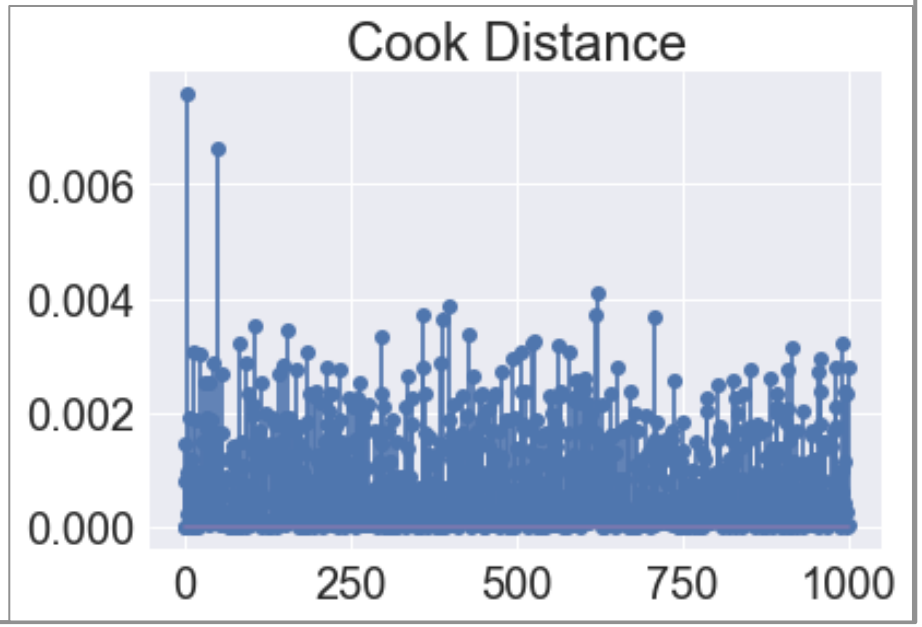
0 outliers according to the box plot

Note: Boxplots work best when the data is vaguely normally distributed (bell-curve shaped and symmetric)

Cook's distance: view outliers

Script

```
# Let's try Cook's Distance
# Remember our SLR model

slr_results_influence = slr_results.get_influence()

# slr_rev_income_CD is the distance and p is the p-value
(slr_rev_income_CD, p) = slr_results_influence.cooks_distance

plt.stem(np.arange(len(slr_rev_income_CD)),
         slr_rev_income_CD)
plt.title('Cook Distance', fontsize = 24)
plt.grid(.25)

slr_rev_income_CD_select =
           slr_rev_income_CD[slr_rev_income_CD >
           4 / len(data)]
data_CD_influencers =
           data[slr_rev_income_CD >
           4 / len(data)]
```

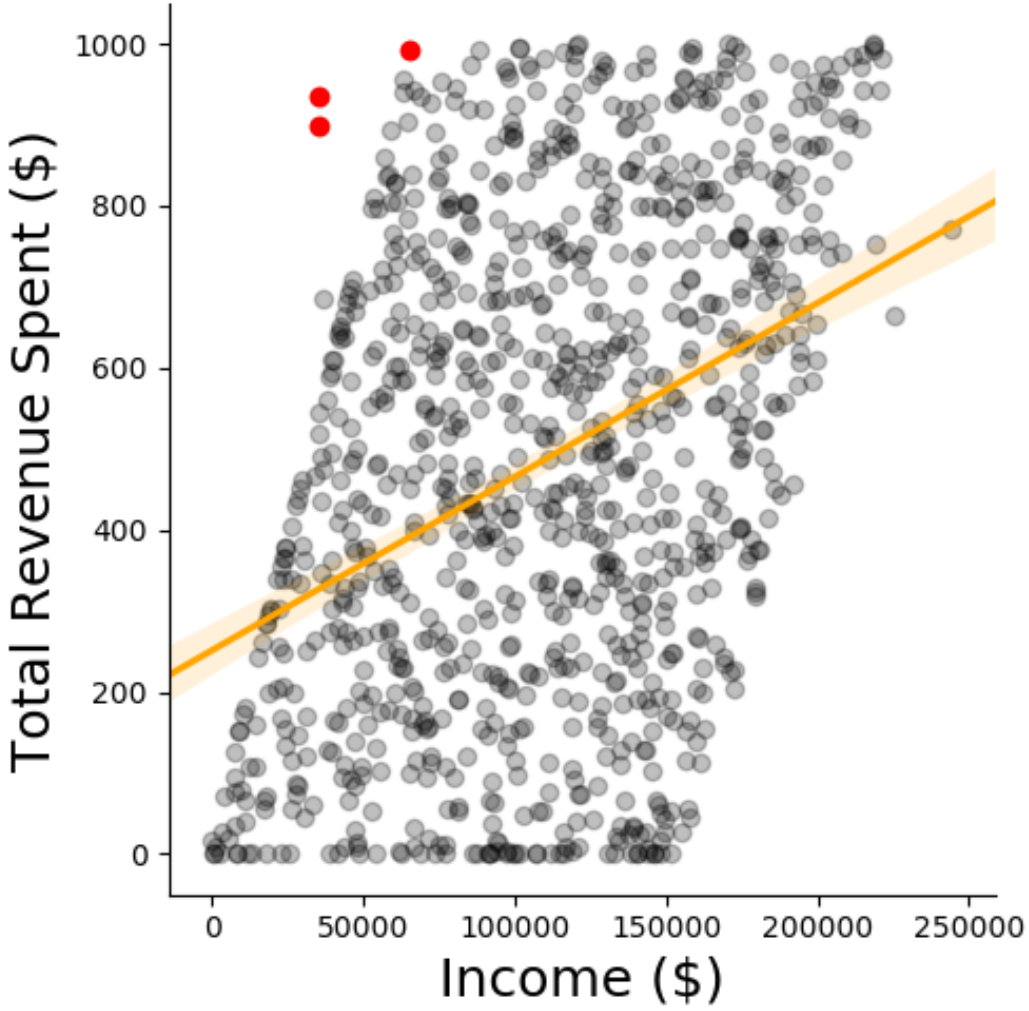


Cook's distance: view outliers

Script

```
# There are 3 potential outliers: observations 4, 50, 621
sns.lmplot(x = 'Income',
           y = 'TotRevSpend',
           data = data,
           line_kws = {'color': 'orange'},
           scatter_kws = {'color': 'black', 'alpha': .25})
plt.scatter(data_CD_influencers['Income'],
            data_CD_influencers['TotRevSpend'],
            color = 'red')
plt.xlabel('Income ($)', fontsize = 18)
plt.ylabel('Total Revenue Spent ($)', fontsize = 18)
```

Cook's distance: view outliers



Cook's distance: remove outliers

- Observation 621 seems to stick with the general pattern of the data, so let's keep it.
- Given incomes of around $35,000, it is out of the ordinary for these fans to spend $900 on NBA entertainment which range from $0 to about $600.
- However, it is totally possible that these fans just choose to allocate their wealth in this way.

Script

```
# Let's remove observations 4 and 50 and see how the analysis is affected.
data_clean = data.drop(data.index[[4, 50]])
y_rev_inc_clean = data_clean['TotRevSpend']
X_rev_inc_clean = data_clean['Income']
slr_rev_income_stat_clean = ols(formula = 'y_rev_inc_clean ~ X_rev_inc_clean',
                                data = data_clean)
slr_results_clean = slr_rev_income_stat_clean.fit()
slr_results_clean.summary()
```

Cook's distance: remove outliers

```
In [93]: slr_results_clean.summary()
Out[93]:
<class 'statsmodels.iolib.summary.Summary'>
"""
                            OLS Regression Results
==============================================================================
Dep. Variable:        y_rev_inc_clean   R-squared:                       0.145
Model:                            OLS   Adj. R-squared:                  0.144
Method:                 Least Squares   F-statistic:                     168.3
Date:                Wed, 29 Nov 2017   Prob (F-statistic):           1.11e-35
Time:                        14:46:02   Log-Likelihood:                -7014.5
No. Observations:                 998   AIC:                         1.403e+04
Df Residuals:                     996   BIC:                         1.404e+04
Df Model:                           1
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
Intercept    246.6058     20.225     12.193      0.000     206.917     286.295
X_rev_inc_clean     0.0022      0.000     12.974      0.000       0.002       0.003
==============================================================================
Omnibus:                      124.954   Durbin-Watson:                   2.006
Prob(Omnibus):                  0.000   Jarque-Bera (JB):               33.503
Skew:                          -0.054   Prob(JB):                     5.31e-08
Kurtosis:                       2.109   Cond. No.                     2.82e+05
==============================================================================
```

Y-intercept decreased from 251.37 to 246.61

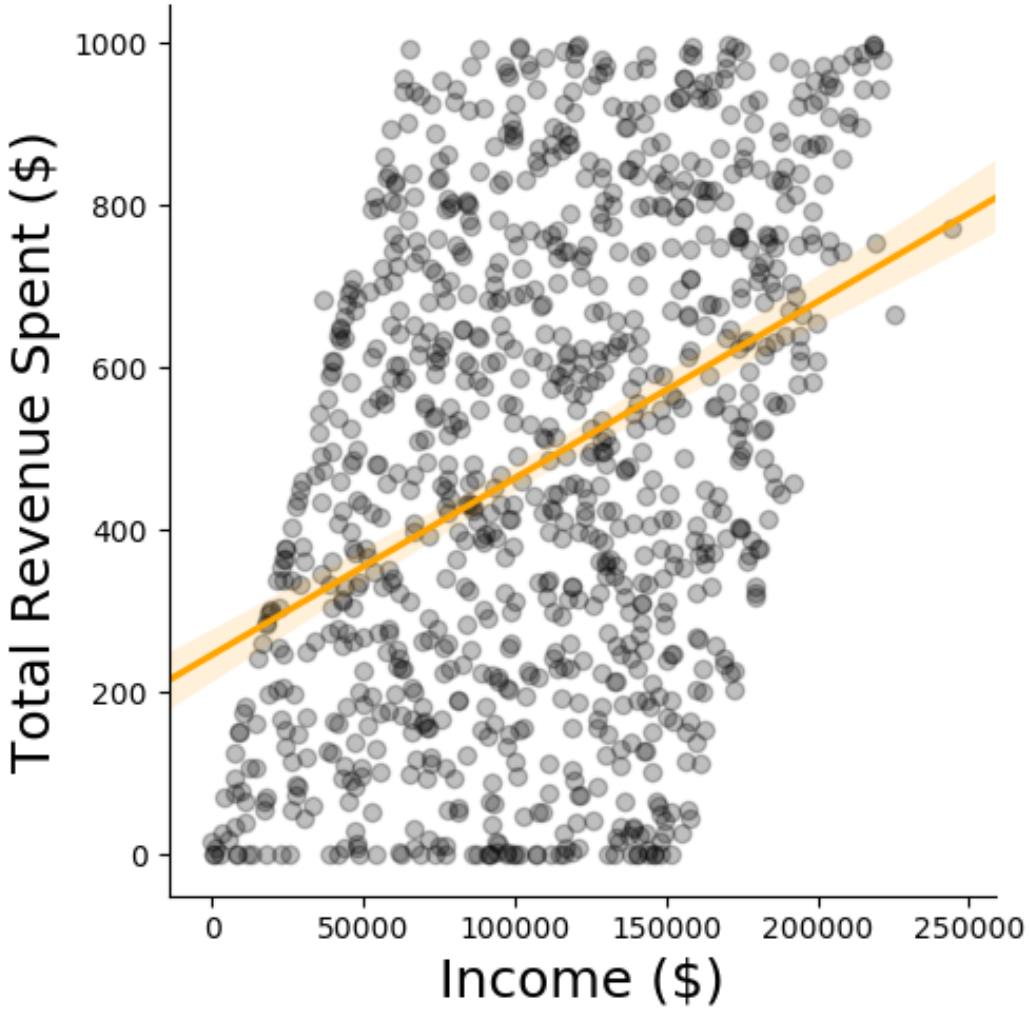
Slope increased from 0.0021 to 0.0022

Regression without outliers

Script

```
# Now let's plot the data without the outliers
sns.lmplot(x = 'Income',
           y = 'TotRevSpend',
           data = data_clean,
           line_kws = {'color': 'orange'},
           scatter_kws = {'color': 'black', 'alpha': .25})
plt.scatter(data_CD_influencers['Income'],
            data_CD_influencers['TotRevSpend'],
            color = 'red')
plt.xlabel('Income ($)', fontsize = 18)
plt.ylabel('Total Revenue Spent ($)', fontsize = 18)
```

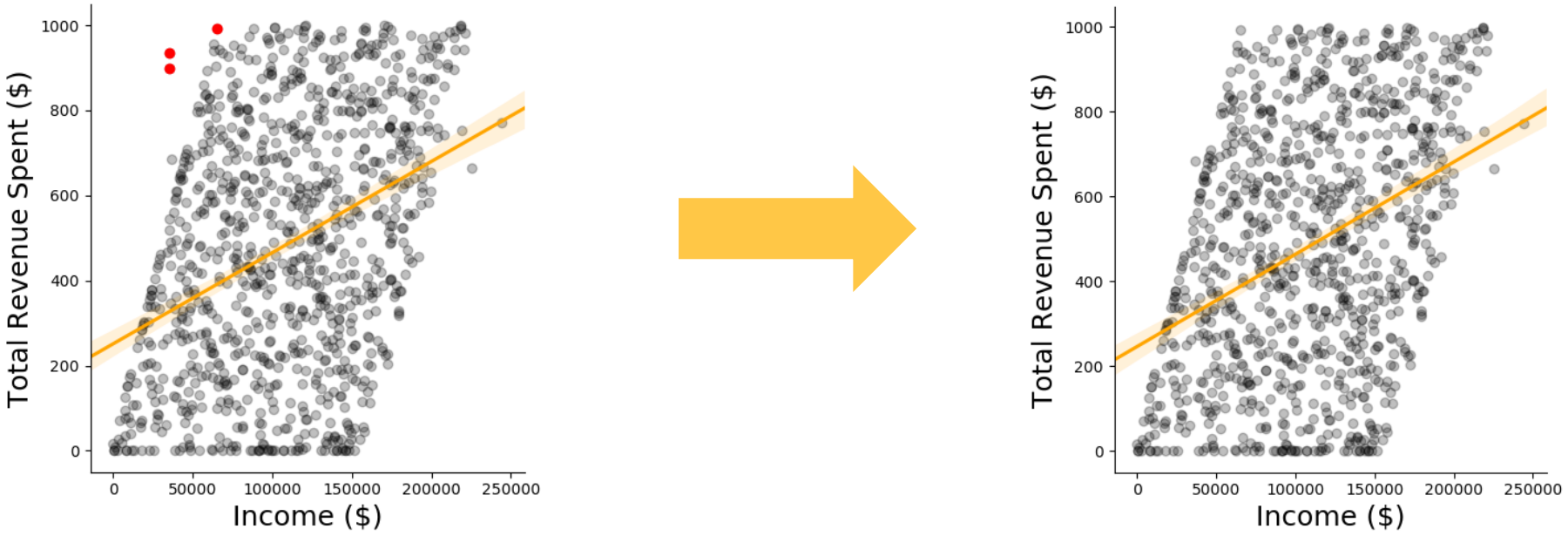
Regression without outliers



Outliers: watch out!

- Never remove outliers without understanding why they are present in the data

- Make sure you understand why removing outliers is the correct course of action for your analysis!



Regression without outliers

Script

```
# Let's consider our final model from lecture 1:
# DistToArena, GamesWatched, Income, YrsInDatabase
X_final2 = data[['DistToArena', 'GamesWatched', 'Income', 'YrsInDatabase']]
y_final2 = data['TotRevSpend']
mlr_final2_stat = ols(formula = 'y_final2 ~ X_final2', data = data)
mlr_final2_results = mlr_final2_stat.fit()
mlr_final2_results.summary()
```

Multiple regression: outliers

OLS Regression Results

Dep. Variable:	y_final2	R-squared:	0.559
Model:	OLS	Adj. R-squared:	0.557
Method:	Least Squares	F-statistic:	314.8
Date:	Wed, 29 Nov 2017	Prob (F-statistic):	5.82e-175
Time:	14:58:33	Log-Likelihood:	-6699.0
No. Observations:	1000	AIC:	1.341e+04
Df Residuals:	995	BIC:	1.343e+04
Df Model:	4		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
Intercept	285.4069	26.426	10.800	0.000	233.549	337.264
X_final2[0]	-0.5334	0.027	-19.942	0.000	-0.586	-0.481
X_final2[1]	6.6455	0.556	11.958	0.000	5.555	7.736
X_final2[2]	0.0011	0.000	8.659	0.000	0.001	0.001
X_final2[3]	18.3647	1.450	12.668	0.000	15.520	21.210

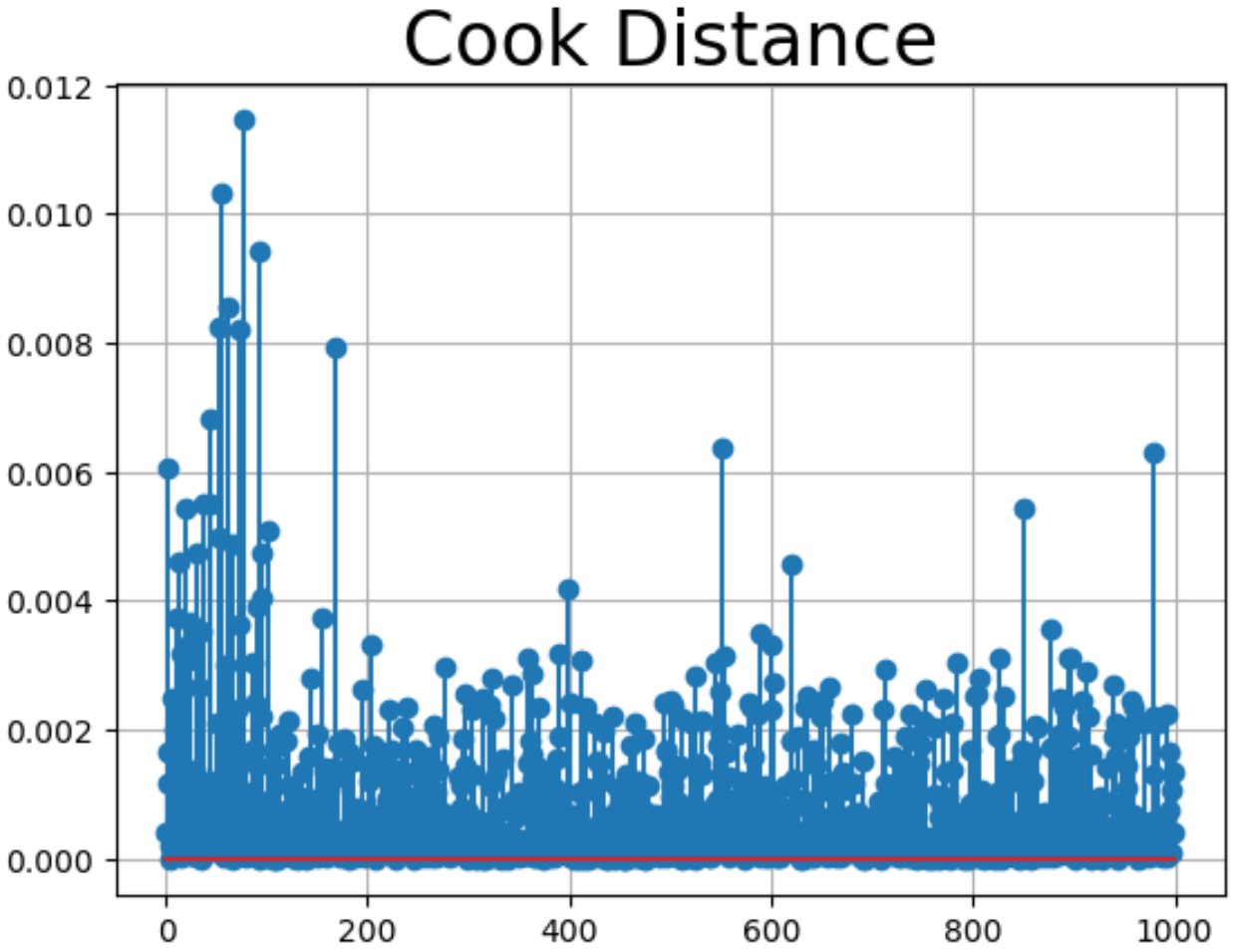
Omnibus:	8.419	Durbin-Watson:	1.858
Prob(Omnibus):	0.015	Jarque-Bera (JB):	5.953
Skew:	-0.036	Prob(JB):	0.0510
Kurtosis:	2.629	Cond. No.	5.12e+05

Multiple regression: outliers

Script

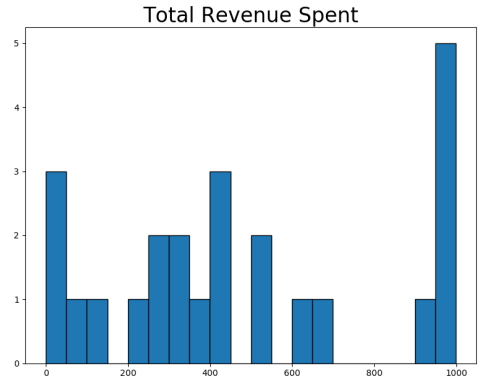
```
# Cook's Distance
mlr_results_influence = mlr_final2_results.get_influence()

# mlr_final2_CD is the distance and p is the p-value
(mlr_final2_CD, p) = mlr_results_influence.cooks_distance
plt.stem(np.arange(len(mlr_final2_CD)), mlr_final2_CD)
plt.title('Cook Distance', fontsize = 24)
plt.grid(.25)

mlr_final2_CD_select = mlr_final2_CD[mlr_final2_CD > 4 / len(data)]
data2_CD_influencers = data[mlr_final2_CD > 4 / len(data)]
```

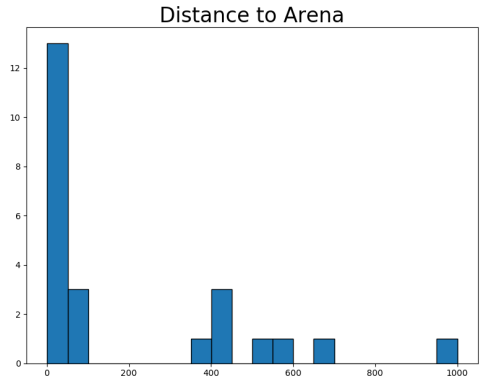
Multiple regression: outliers

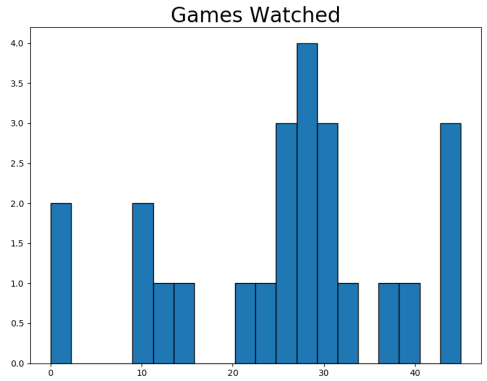


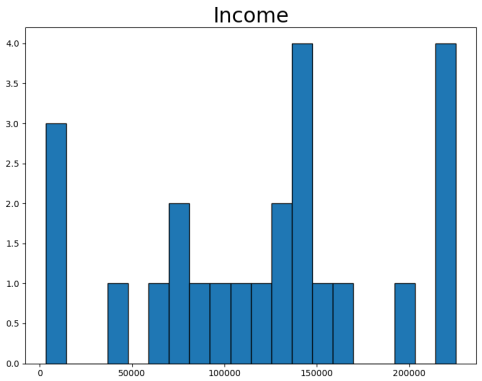
Multiple regression: outliers

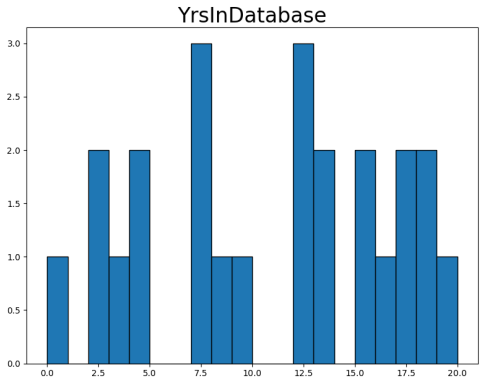
- According to Cook's distance and our threshold of 0.004, there are 24 possible outliers. Let's plot 24 outliers: 4 / 1000 threshold











Multiple regression: outliers

- It looks like many of these "outliers" live relatively close to the arena, watched about 30 games, have higher incomes and have been in the database for a number of years. These just seem like satisfied customers!

Script

```
# Let's try to limit the selected outliers to ensure that
# we don't exclude any points that may not be outliers. Instead of 4 in the
# numerator, we'll double it.
data2_CD_influencers2 = data[mlr_final2_CD > (4 * 2) / len(data)]

# View the data identified by Cook's
# method as potential outliers.
# Select only the rows and columns we're
# interested in.
data2_CD_influencers2

# Observations 52, 54, 61, 72, 77, 92
```

```
In [104]: data2_CD_influencers2
Out[104]:
    TotRevSpend  DistToArena  GamesWatched  Income  FanSatisfaction  \
52          443            8            10  166802                4
54          423           75            26  108875                3
61          311           75            29  141764                1
72          511           38            45  128211                3
77          504           42            43  131450                2
92          999           18             1  218371                6

    YrsInDatabase  FanComplaints
52             17              9
54             20              5
61             16             10
72             15              3
77             18              2
92              7              1
```

Multiple regression: outliers

Script

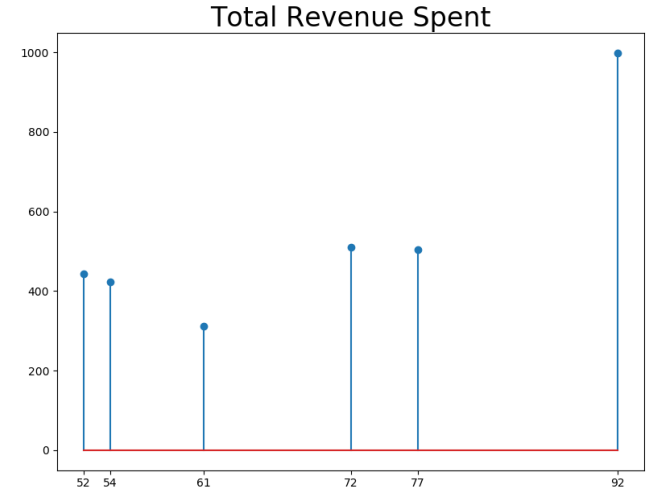
```
# Get observation numbers of outliers for the 8 / 1000 threshold
outlier_obs = []
for i in range(len(data2_CD_influencers2)):
    outlier_obs.append(data2_CD_influencers2.index[i])
outlier_obs

# Create plots for each variable with all the outliers at this new threshold
fig = plt.figure()
ax1 = fig.add_subplot(231)
ax1.stem(outlier_obs, data2_CD_influencers2['TotRevSpend'])
ax1.set_xticks(outlier_obs)
ax1.set_title('Total Revenue Spent')

ax2 = fig.add_subplot(232)
ax2.stem(outlier_obs, data2_CD_influencers2['DistToArena'])
ax2.set_xticks(outlier_obs)
ax2.set_title('Distance to Arena')
```

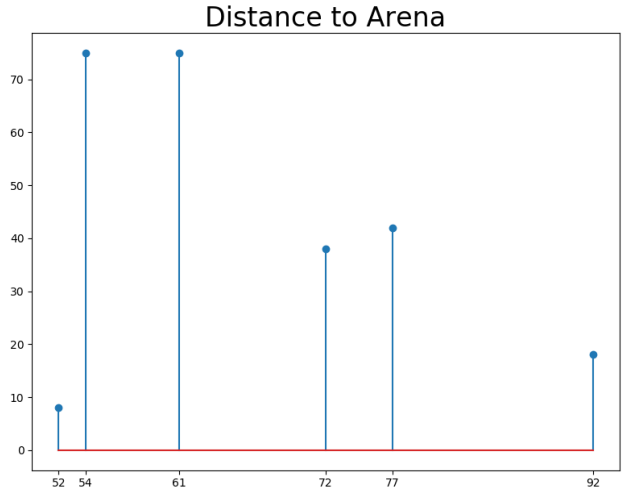
Multiple regression: outliers

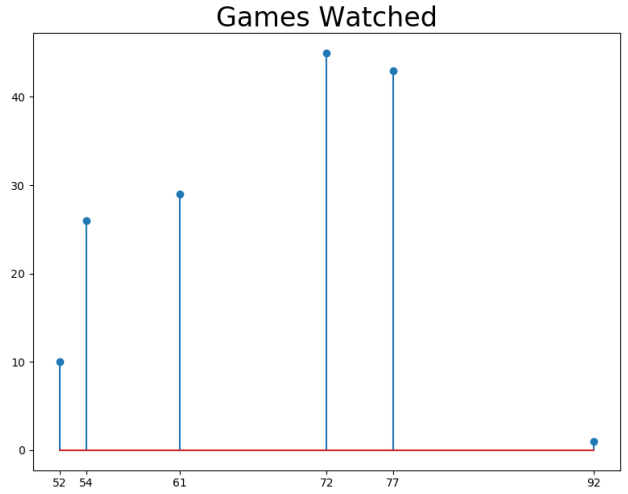
Script

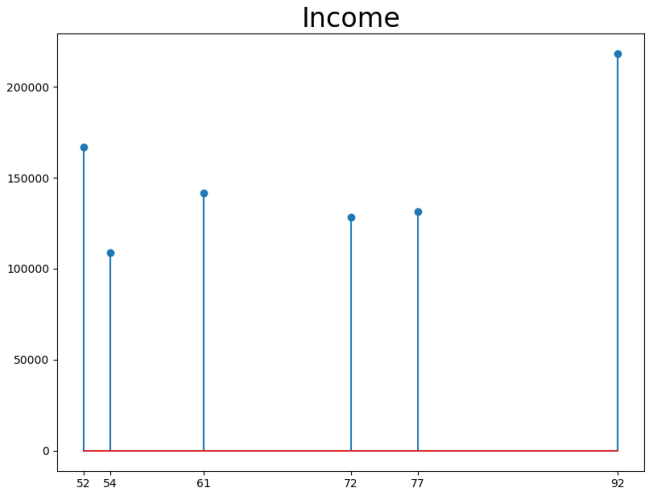
```
(continued…)

ax3 = fig.add_subplot(233)
ax3.stem(outlier_obs, data2_CD_influencers2['GamesWatched'])
ax3.set_xticks(outlier_obs)
ax3.set_title('Games Watched')

ax4 = fig.add_subplot(234)
ax4.stem(outlier_obs, data2_CD_influencers2['Income'])
ax4.set_xticks(outlier_obs)
ax4.set_title('Income')

ax5 = fig.add_subplot(235)
ax5.stem(outlier_obs, data2_CD_influencers2['YrsInDatabase'])
ax5.set_xticks(outlier_obs)
ax5.set_title('YrsInDatabase')
```

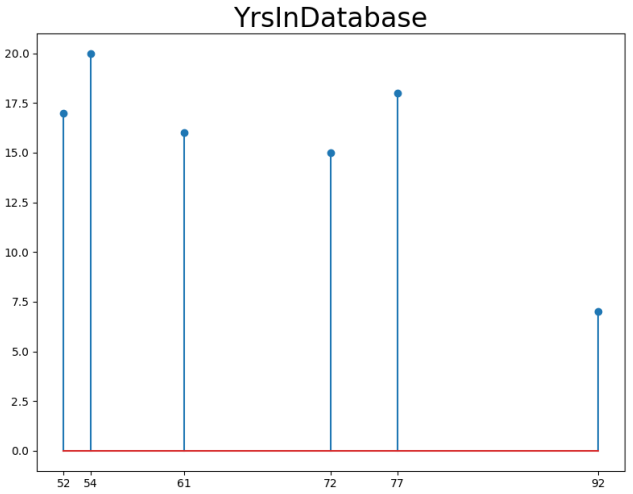
Multiple regression: outliers











Multiple regression: outliers

Script

```
# It looks like these 6 observations make more than $100,000 in income, have
# mostly spent at least 15 years in the database, and watched about 30 games.
# Observation 92 seems to be a newer NBA fan having watched only a few games
# but in the games the fan watched he/she spent a lot of money.

# If we remove these observations let's see how the regression analysis might change.
data2_clean = data.drop(data.index[outlier_obs])
X_final2_clean = data2_clean[['DistToArena', 'GamesWatched',
                              'Income', 'YrsInDatabase']]
y_final2_clean = data2_clean['TotRevSpend']
mlr_final2_stat_clean = ols(formula = 'y_final2_clean ~ X_final2_clean',
                            data = data2_clean)

mlr_final2_results_clean = mlr_final2_stat_clean.fit()
mlr_final2_results_clean.summary()
```

Multiple regression: outliers

Metric	Original model	Model without outliers
Adjusted R^2	0.557	0.571
Intercept	285.4069	283.6414
DistToArena	-0.5334	-0.5443
GamesWatched	6.6455	6.7674
Income	0.0011	0.0011
YrsInDatabase	18.3647	19.2615

Outline: Intro to Regression

1. Single variable linear regression
2. Multiple linear regression
3. Model selection
4. Measuring variance and error
5. Dealing with outliers
6. Checking for model validity
7. Interactions
8. Regression with categorical variables

Testing for multicollinearity: VIF

- Variance-inflation factors (VIFs) are a good test for multicollinearity
  - Especially helpful for regression with categorical data where `ggpairs()` and correlation measurement may not be meaningful
- VIF measures how much the variance of a regression coefficient is increased due to collinearity

$$VIF_i = \frac{1}{1 - R^2_i}$$

- R^2_i = the R^2 of the model if a regression model is run with i as the dependent variable and all other variables in the model as the independent variables
- Rule of thumb: if VIF > 10, then multicollinearity is high

Testing for multicollinearity: VIF

$$R^2 = 1 - \frac{\text{Randomness}}{\text{Variance}}$$

$$1 - R^2 = \frac{\text{Randomness}}{\text{Variance}}$$

so...

$$\text{VIF} = \frac{1}{\frac{\text{Randomness}}{\text{Variance}}}$$

$$\text{VIF}_i = \frac{1}{1 - R^2_i}$$

The more the rest of the variables in the model can explain the variance in variable `i`, the more the other variables in the model capture all the effects represented by variable `i`!

$$\text{VIF}_i = \frac{\text{Variance}}{\text{Randomness}}$$ = variance as a multiple of its unexplained component for the regression model of variable `i`

Validating: check for multicollinearity

Script

```
# We will use the variance_inflation_factor function from statsmodels package
# to check for multicollinearity.
from statsmodels.stats.outliers_influence import variance_inflation_factor

# Consider a model including all 6 predictor variables
X_all_multicollinearity = [variance_inflation_factor(data.iloc[:,1:].values, j)
    for j in range(data.iloc[:, 1:].shape[1])]
X_all_multicollinearity

# While none of the variables have egregious VIFs
# there is some in all and VIFs certainly over 5
# are worth looking into.

# Organize VIFs into a data frame.
data.iloc[:, 1:].corr()
```

Validating: check for multicollinearity

Script

```
# We will use the variance_inflation_factor function from statsmodels package
# to check for multicollinearity.
from statsmodels.stats.outliers_influence import variance_inflation_factor

# Consider a model including all 6 predictor variables
X_all_multicollinearity = [variance_inflation_factor(data.iloc[:,1:].values, j)
    for j in range(data.iloc[:,1:].shape[1])]
X_all_multicollinearity

# Let's look at the correlations between all the predictors
data.iloc[:, 1:].corr()
```

Validating: check for multicollinearity

Script

```
 Variables       VIF
0      DistToArena  3.465245
1     GamesWatched  5.022505
2           Income  5.296337
3  FanSatisfaction  5.053031
4    YrsInDatabase  4.913695
5  FanComplaints    2.450976
```

Fairly high VIFs

Correlation	DistToArena	GamesWatched	Income	FanSatisfaction	YrsInDatabase	FanComplaints
DistToArena	1	-0.145751	-0.19673	-0.232739	-0.215945	0.172823
GamesWatched	-0.145751	1	0.184111	0.072042	0.196412	-0.03677
Income	-0.196725	0.184111	1	0.11846	0.180637	-0.082256
FanSatisfaction	-0.232739	0.072042	0.11846	1	0.104565	-0.649276
YrsInDatabase	-0.215945	0.196412	0.180637	0.104565	1	-0.103144
FanComplaints	0.172823	-0.03677	-0.08226	-0.649276	-0.103144	1

Validating: check for multicollinearity

Script

```
# Since FanComplaints and FanSatisfaction are somewhat strongly negatively
# correlated, let's run a model without FanSatisfaction and examine its predictive
# power.
X_no_FanSatisfaction = data.iloc[:, 1:].drop('FanSatisfaction', axis = 1)
X_no_FanSatisfaction_multicollinearity = [variance_inflation_factor(
        X_no_FanSatisfaction.values, i)
        for i in range(X_no_FanSatisfaction.shape[1])]
X_no_FanSatisfaction_multicollinearity

# The VIFs decreased across the board to all below 5.
pd.concat([pd.DataFrame(X_no_FanSatisfaction.columns, columns = ['Variables']),
           pd.DataFrame(X_no_FanSatisfaction_multicollinearity, columns = ['VIF'])],
           axis = 1)
```

	Variables	VIF
0	DistToArena	3.114654
1	GamesWatched	4.665636
2	Income	4.681559
3	YrsInDatabase	4.484057
4	FanComplaints	1.922651

All now lower than 5 but there is still some multicollinearity

Validating: check for multicollinearity

Script

```
# Let's consider our final model from Lecture 1 which included:
# DistToArena, GamesWatched, Income, YrsInDatabase i.e. we've removed FanComplaints
X_final2_multicollinearity = [variance_inflation_factor(
                                X_final2.values, i)
                                for i in range(X_final2.shape[1])]
pd.concat([pd.DataFrame(X_final2.columns, columns = ['Variables']),
           pd.DataFrame(X_final2_multicollinearity, columns = ['VIF'])],
           axis = 1)
```

	Variables	VIF
0	DistToArena	2.725881
1	GamesWatched	4.625936
2	Income	4.663273
3	YrsInDatabase	4.478689

All now lower than 5
but there is still some
multicollinearity

```
# What if we standardize our data?
X_final2_scaled = preprocessing.scale(X_final2)
X_final2_scaled_multicollinearity = [variance_inflation_factor(
        X_final2_scaled, i) for i in range(X_final2_scaled.shape[1])]
```

Validating: check for multicollinearity

Original model

Variables	VIF
DistToArena	3.465245
GamesWatched	5.022505
Income	5.296337
FanSatisfaction	5.053031
YrsInDatabase	4.913695
FanComplaints	2.450976

Standardized and without FanSatisfaction and FanComplaints

Variables	VIF
DistToArena	1.085791
GamesWatched	1.073145
Income	1.083341
YrsInDatabase	1.096264

Standardizing the data fixed our multicollinearity issues bringing the VIFs of all remaining variables in our potential model to almost 1!

To keep in mind

- Multicollinearity doesn't affect the fit of the model
- Removing multicollinearity gives us more confidence in the reliability of our estimates since the standard errors of our estimates should decrease
- Multicollinearity can cause unexpected changes in the sign and significance of coefficient estimates

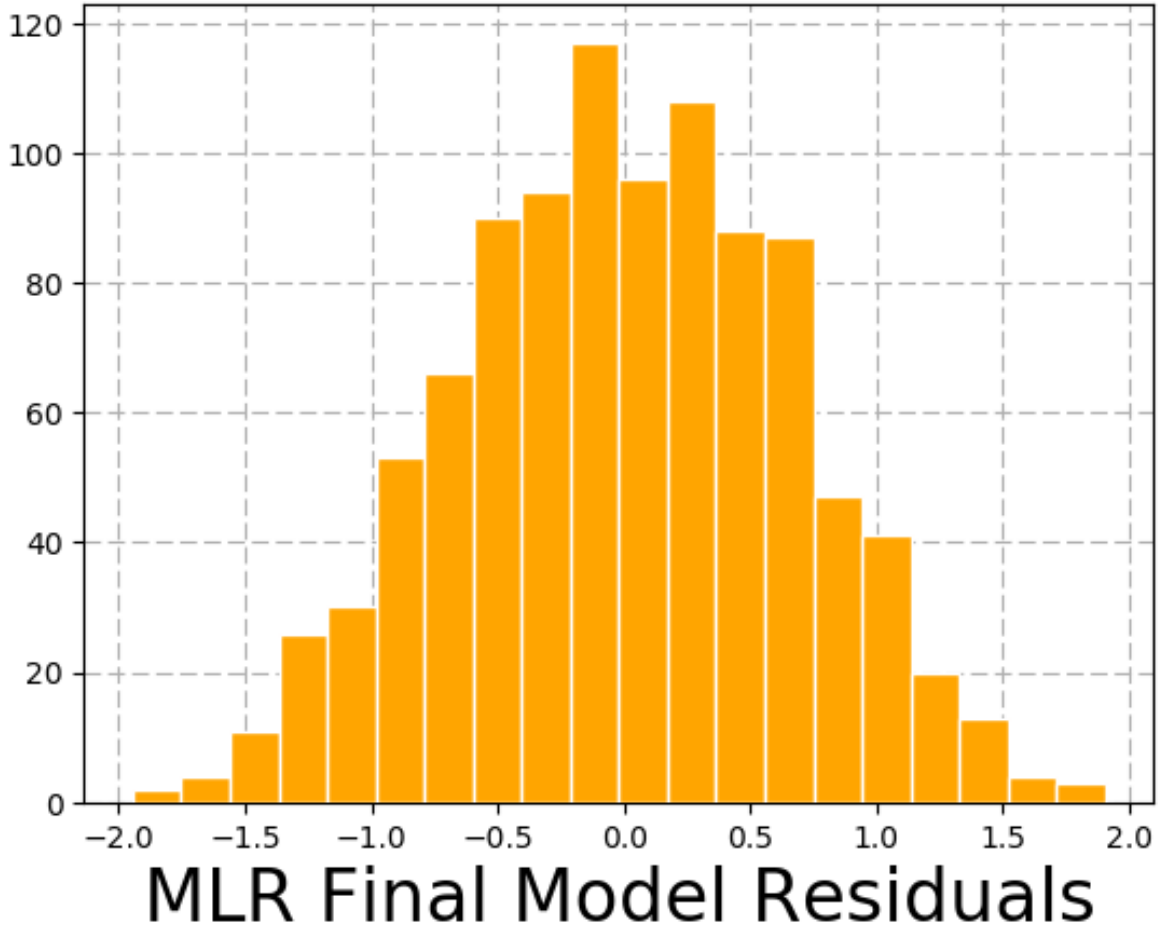
Validating: check the residuals

Script

```
# Let's consider the final model from lecture 1 with standardized data. This model
# includes: DistToArena, Income, GamesWatched, YrsInDatabase.

plt.hist(mlr_final_results.resid,
         bins = 20,              1. Specify number of bins desired
         color = 'orange',       2. Color the bins
         ec = 'white',           3. Make outline of each bin white
         zorder = 3)             4. zorder makes the bins on top of the grid lines in the back
plt.xlabel('MLR Final Model Residuals',
           fontsize = 24)
plt.grid(.25,
         linestyle = 'dashed',
         zorder = 0)
```

Validating: check the residuals

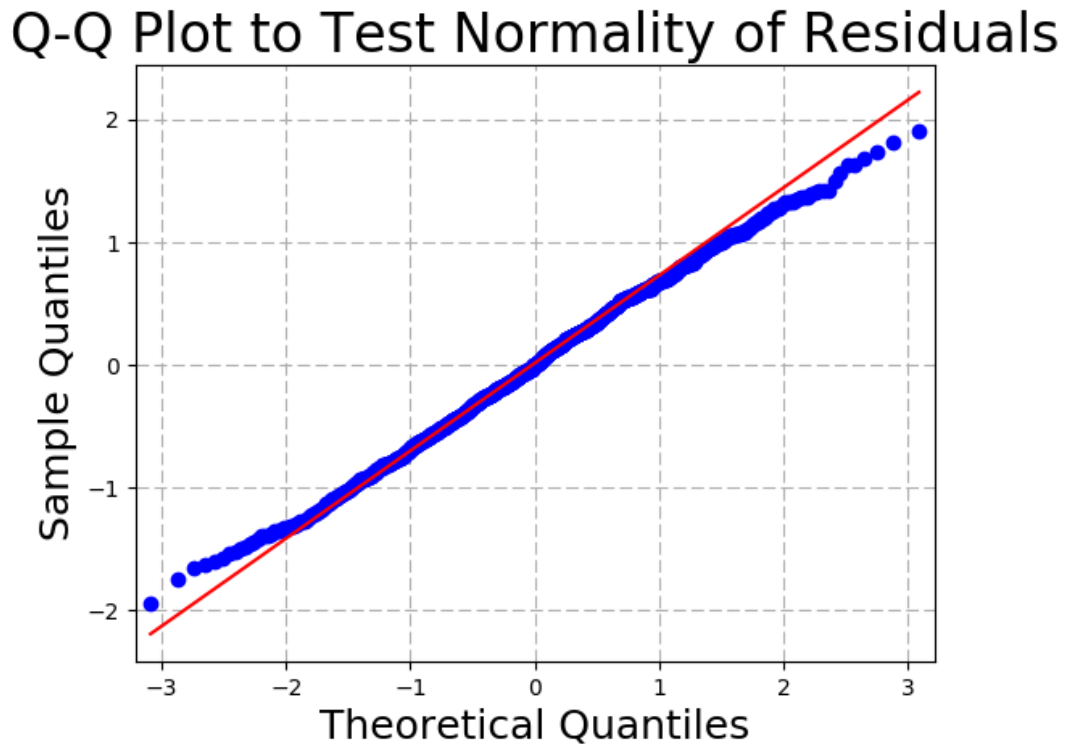
The residuals seem normal, let's check the Q-Q plot!



Validating: check the residuals

Script

```
# Now let's check the residuals for normality of distribution using the Q-Q plot
sm.qqplot(mlr_final_results.resid, line = 's')
plt.title('Q-Q Plot to Test Normality of Residuals', fontsize = 24)
plt.xlabel('Theoretical Quantiles', fontsize = 18)
plt.ylabel('Sample Quantiles', fontsize = 18)
plt.grid(.25, linestyle = 'dashed')
```



Looks like our model has slightly heavier tails than a normal distribution. Since the difference is only slight, these are probably good enough!

Validating: check the residuals

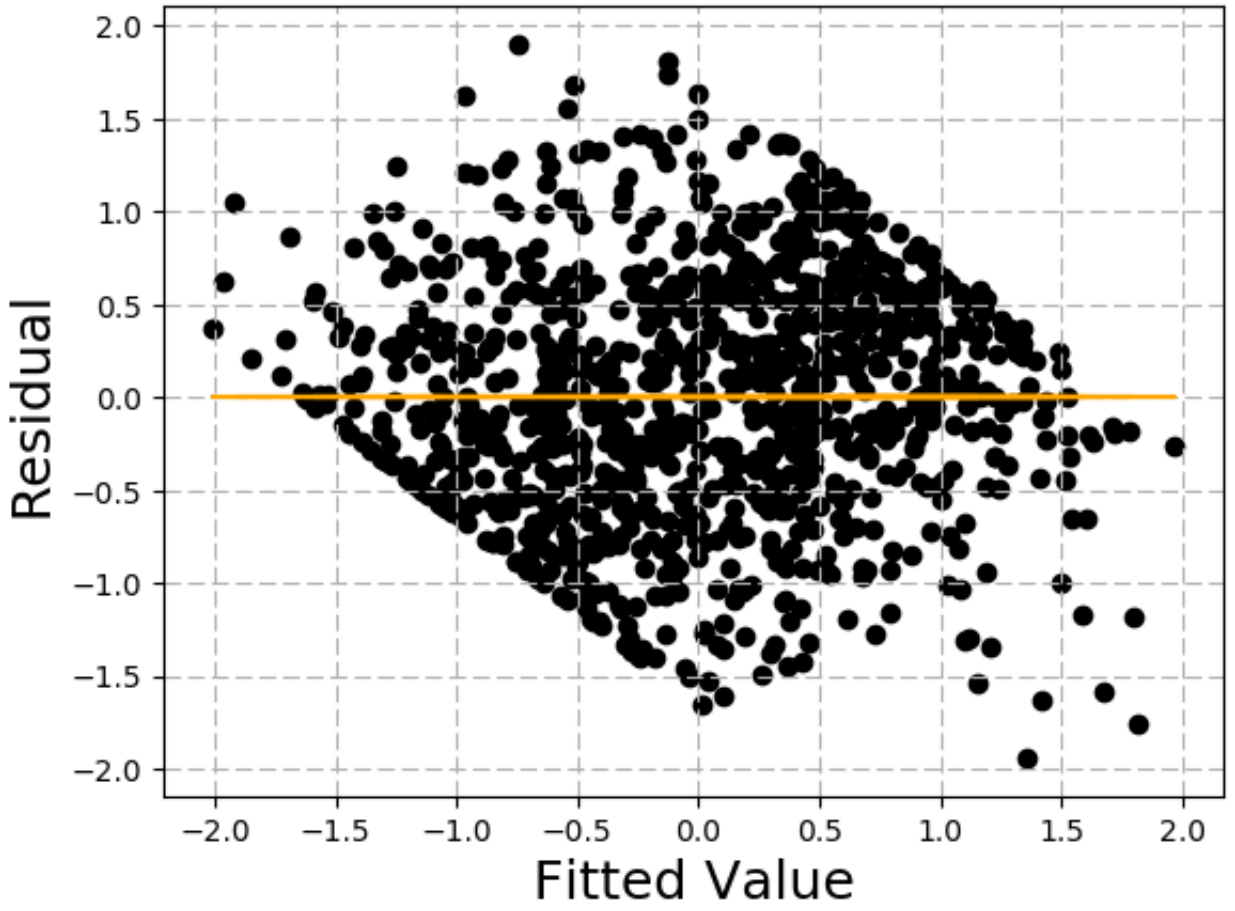
Script

```
# Does the variance of the residuals change with the predicted value?
plt.scatter(mlr_final_results.fittedvalues,
            mlr_final_results.resid,
            color = 'black')
plt.plot(mlr_final_results.fittedvalues,
         np.zeros((1000, )), color = 'orange')
plt.xlabel('Fitted Value', fontsize = 18)
plt.ylabel('Residual',
           fontsize = 18)
plt.grid(.25, linestyle = 'dashed')

plt.scatter(mlr_final_results.predict(), mlr_final_results.resid.abs(), color = 'black')
plt.plot(mlr_final_results.predict(), np.zeros((1000, )), color = 'orange')
plt.xlabel('Fitted Value', fontsize = 18)
plt.ylabel('Absolute Value of Residual', fontsize = 18)
```

There should not be a pattern but there is clearly a triangular shape to the absolute value of the residuals. This means we're not taking something into account.

Validating: check the residuals

Looks like the residuals follow somewhat of a parallelogram pattern, which means that our model is not taking something into account.



Testing for heteroscedasticity

- A good explanatory model will have residuals whose variance does not depend on the independent (predictor) variables

- If the variance of the residuals is affected by the magnitude of the independent variables then heteroscedasticity is present
  - This means we have to refine our model further as there is an effect that we're not taking into account

- To test for this, run a regression solving for the error terms using the independent variables in the model

$$y = mx^2 + b + e_{rror}$$

Testing for heteroscedasticity

- $y = mx^2 + b + e_{rror}$
- $e_{rror} = a + m_1x_1 + m_2x_2 + m_3x_3 + m_4x_4 \ldots m_nx_n$
- If there is no heteroscedasticity, then the coefficients m will not be material, they will all be equal to 0 or close to it
- $e_{rror} = 0 + 0x_1 + 0x_2 + 0x_3 + 0x_4 \ldots 0x_n = \sim 0$

Testing for heteroscedasticity

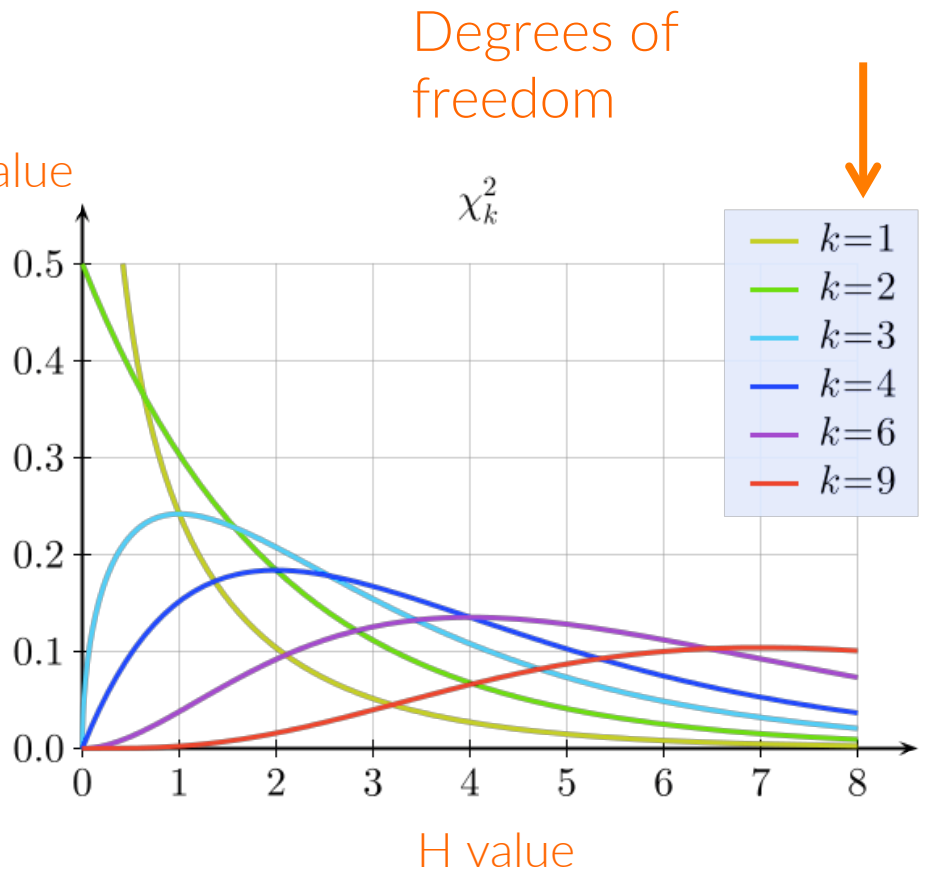
- $e_{rror} = a + m_1x_1 + m_2x_2 + m_3x_3 + m_4x_4 \ldots m_nx_n$
- N = number of points in the data set
- R^2 = the R^2 of the regression equation for the e_{rror} term

- Heteroscedasticity (Breusch-Pagan) test:

  $H = N^*R^2$

- H is tested against a X^2 distribution (chi-squared) given the degrees of freedom (number of data points – number of model parameters)

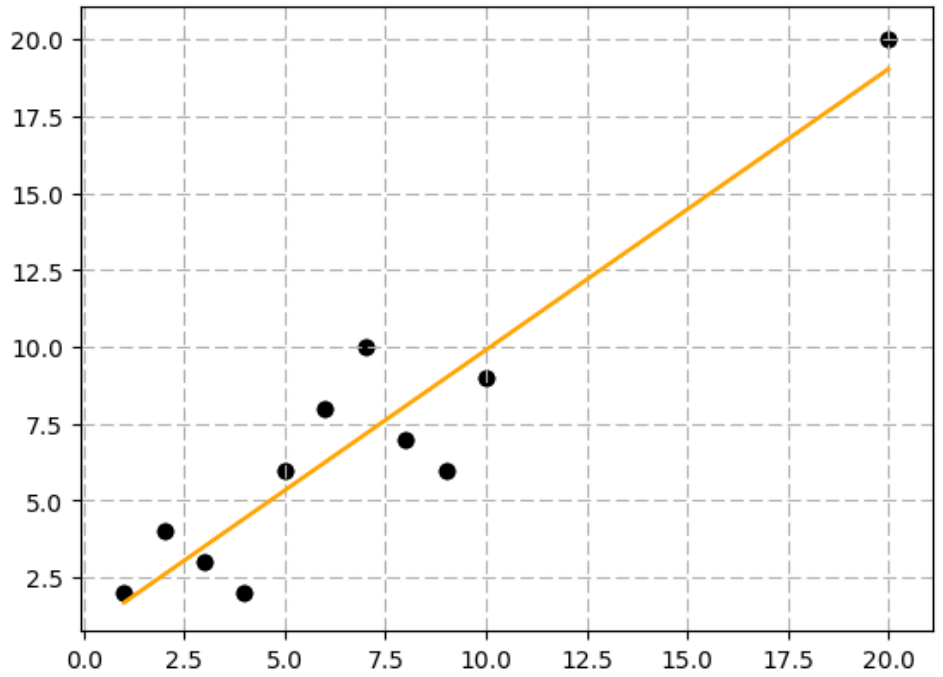
- The p-value of the distribution tells you if heteroscedasticity is likely present (if the regression model for the errors explains them)



Homoscedasticity example

Script

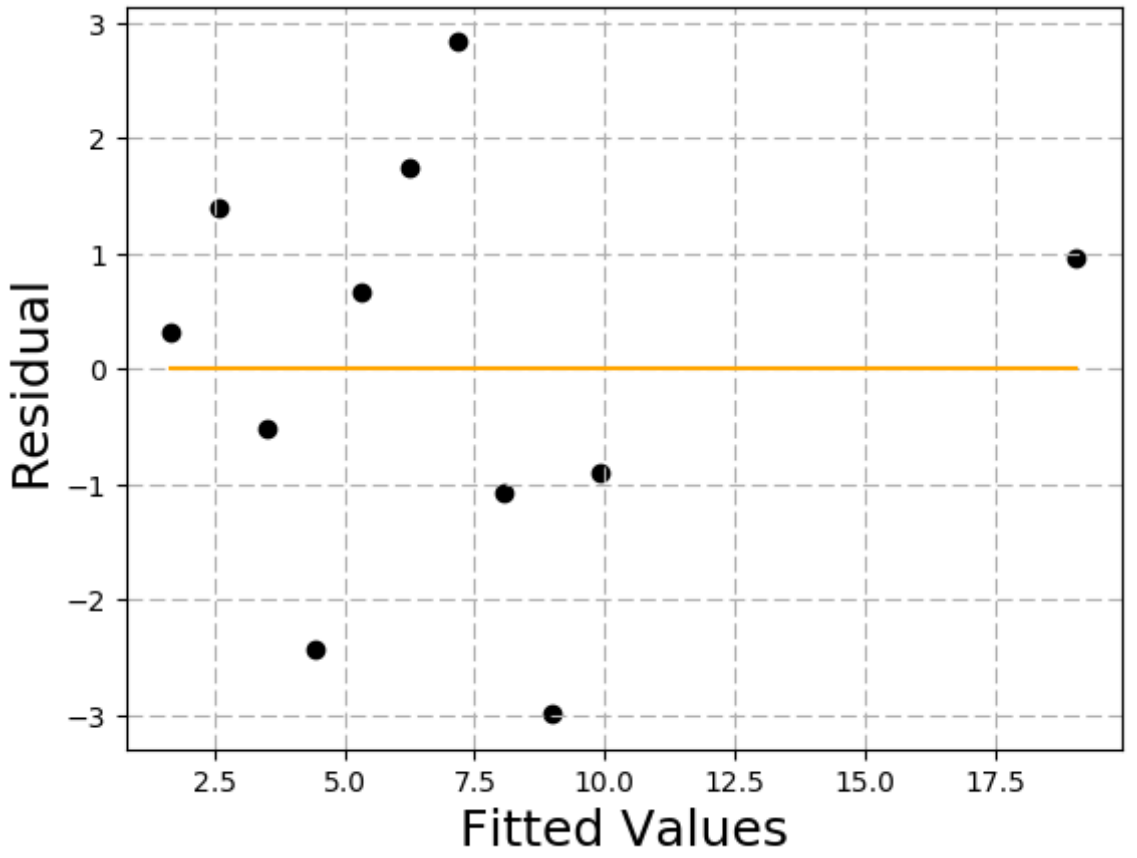
```
# Example with homoscedasticity (no heterscedasticity present)
sample_data = np.column_stack(([1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 20],
                               [2, 4, 3, 2, 6, 8, 10, 7, 6, 9, 20]))
sample_df = pd.DataFrame(sample_data, columns = ['x_data', 'y_data'])
sample_lm = ols('y_data ~ x_data',
                data=sample_df).fit()

plt.scatter(sample_df['x_data'],
            sample_df['y_data'],
            color = 'black')
plt.plot(sample_df['x_data'],
         sample_lm.fittedvalues,
         color = 'orange')
```



Homoscedasticity example

Script

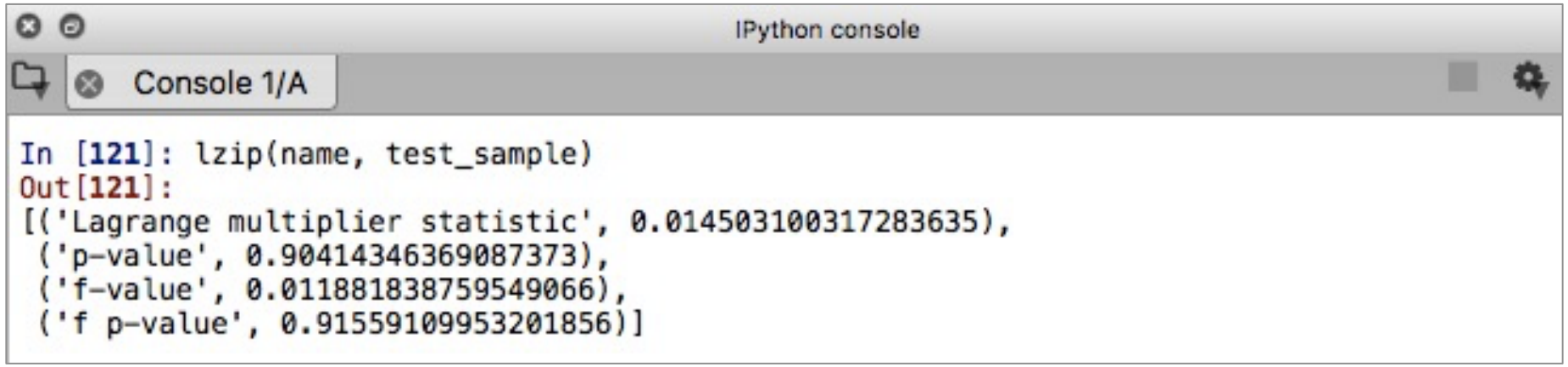
```
# Plot residuals vs. fitted values.
plt.scatter(sample_lm.fittedvalues,
            sample_lm.resid,
            color = 'black')
plt.plot(sample_lm.fittedvalues,
         np.zeros((11, )),
         color = 'orange')
plt.xlabel('Fitted Values',
           fontsize = 18)
plt.ylabel('Residual',
           fontsize = 18)
plt.grid(.25,
         linestyle = 'dashed')
```



Breusch-Pagan test

Script

```
# You can run the Breusch-Pagan test with het_breuschpagan function in the
# statsmodels.stats.api module.
import statsmodels.stats.api as sms
from statsmodels.compat import lzip

# Labels for output of test
name = ['Lagrange multiplier statistic', 'p-value', 'f-value', 'f p-value']
test_sample = sms.het_breuschpagan(sample_lm.resid, sample_lm.model.exog)
lzip(name, test_sample) # pairs each output from test with its meaning
```



Breusch-Pagan test

Script

```
# Both p-values are very large for the sample data, which means that the
# regression model for the error terms has poor explanatory power, which in
# turn means that there is NO heteroscedasticity.
test_mlr = sms.het_breuschpagan(mlr_final_results.resid, mlr_final_results.model.exog)
lzip(name, test_mlr)

# While the p-value is not as high as the sample data, it is still above
# our 5% significance level. This means that our regression model for the
# error terms has some explanatory power, but NOT enough to say that there is
# heteroscedasticity present.
```

```
In [121]: lzip(name, test_sample)
Out[121]:
[('Lagrange multiplier statistic', 0.014503100317283635),
 ('p-value', 0.90414346369087373),
 ('f-value', 0.011881838759549066),
 ('f p-value', 0.91559109953201856)]
```

```
In [123]: lzip(name, test_mlr)
Out[123]:
[('Lagrange multiplier statistic', 4.1206437207430557),
 ('p-value', 0.38992464876882049),
 ('f-value', 1.0292513034553478),
 ('f p-value', 0.390976680592237)]
```

Exercise time!

Potential remedies

- If you would rather trust the residual plot rather than the Breusch-Pagan test, there are
- various ways to deal wth heteroscedasticity, including:

- Transformation of the dependent or independent variables

- Instead of ordinary least squares, try weighted least squares

- Add polynomial or interaction terms

- Choose a non-linear model

Outline: Intro to Regression

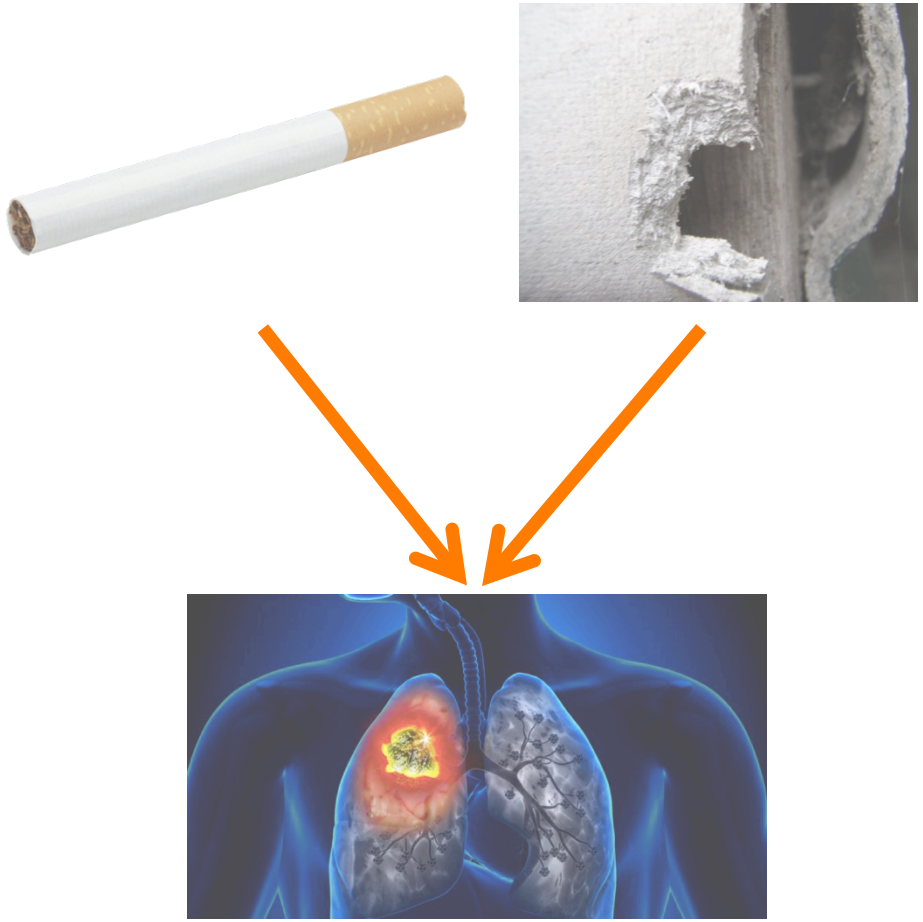
1. Single variable linear regression
2. Multiple linear regression
3. Model selection
4. Measuring variance and error
5. Dealing with outliers
6. Checking for model validity
7. Interactions
8. Regression with categorical variables

Variable interactions

- Some variables may represent different things but nonetheless may amplify one another

- Example:
  - Smoking and asbestos are different things, both cause lung cancer, but when combined the risk of getting lung cancer ***multiplies***!

- To capture this additional risk of both risk factors being present together, an interaction term should be added to the model



Variable interactions

- In the context of our NBA fan behavior analysis:
  - Both income and games watched (up to a certain point) increase total revenue spent.
  - It would make intuitive sense that a fan with more income who watched a lot of games would further increase the expected total revenue spent!
    - Therefore, we should consider testing the interaction term of income and games watched!
- Combining variables can decrease interpretability, but increase predictive accuracy – keep your objectives in mind!

Variable interactions

Script

```
data.loc[data['Income'] <= 50000, 'IncomeLevel'] = 1
data.loc[(data['Income'] > 50000) & (data['Income'] <= 125000), 'IncomeLevel'] = 2
data.loc[data['Income'] > 125000, 'IncomeLevel'] = 3

# Build interaction model
rev = data['TotRevSpend']
gw = data['GamesWatched']
il = data['IncomeLevel']
rev_il_gw_interaction_lm = ols(
        formula = 'rev ~ gw*il', data = data).fit()

# If you want il encoded with dummy variables, use C(il) in the formula.
rev_il_gw_interaction_lm.summary()
```

Variable interactions

Console

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                    rev   R-squared:                       0.229
Model:                            OLS   Adj. R-squared:                  0.227
Method:                 Least Squares   F-statistic:                     98.76
Date:                Fri, 24 Nov 2017   Prob (F-statistic):           5.79e-56
Time:                        15:25:23   Log-Likelihood:                -6977.6
No. Observations:                1000   AIC:                         1.396e+04
Df Residuals:                     996   BIC:                         1.398e+04
Df Model:                           3
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
Intercept     52.4691     58.907      0.891      0.373     -63.127     168.066
gw             8.6731      2.457      3.531      0.000       3.852      13.494
il            93.1757     25.858      3.603      0.000      42.434     143.917
gw:il          0.3719      1.038      0.358      0.720      -1.665       2.408
```

Variable interactions

Script

```
# To do this in scikit-learn:
# Not one hot encoded
poly = preprocessing.PolynomialFeatures(2, interaction_only = True,
                                        include_bias = False)
il_gw_interaction_sklearn = poly.fit_transform(data[['GamesWatched', 'IncomeLevel']])
rev_il_gw_interaction_lm_sklearn = LinearRegression()
rev_il_gw_interaction_lm_sklearn.fit(il_gw_interaction_sklearn, rev)
rev_il_gw_interaction_fitted =
        rev_il_gw_interaction_lm_sklearn.predict(il_gw_interaction_sklearn)

r2_score(rev, rev_il_gw_interaction_fitted)
# 22.9%

# If a categorical variable has n levels, then a one-hot encoding of that variable will
create n dummy variables.
```

Variable interactions

Script

```
# To do this in scikit-learn:
# One-hot encoding
from sklearn.preprocessing import OneHotEncoder

integer_encoded = data['IncomeLevel'].values.reshape(len(data['Income']), 1)
onehot_encoder = OneHotEncoder(sparse = False)
onehot_encoded = onehot_encoder.fit_transform(integer_encoded)
poly_onehot = preprocessing.PolynomialFeatures(2, interaction_only = True,
                                               include_bias = False)
il_gw_interaction_lm_sklearn_onehot =
poly_onehot.fit_transform(data['GamesWatched'].values.reshape(-1, 1), onehot_encoded)
rev_il_gw_interaction_lm_sklearn_onehot = LinearRegression()
rev_il_gw_interaction_lm_sklearn_onehot.fit(il_gw_interaction_lm_sklearn_onehot, rev)
rev_il_gw_interaction_onehot_fitted =
rev_il_gw_interaction_lm_sklearn_onehot.predict(il_gw_interaction_lm_sklearn_onehot)
r2_score(rev, rev_il_gw_interaction_onehot_fitted)

# 17.2% - Performed worse!
# Using this encoding we lose information on the ordering of income levels!
```

Variable interactions

Script

```
## Let's include interaction terms in our model selection process from lecture 1.
rev_scaled = y_scaled
dist_scaled = X_scaled[:, 0:1]
gw_scaled = X_scaled[:, 1:2]
inc_scaled = X_scaled[:, 2:3]
fs_scaled = X_scaled[:, 3:4]
yid_scaled = X_scaled[:, 4:5]
fc_scaled = X_scaled[:, 5:6]

mlr_interaction = ols(formula = 'rev_scaled ~ dist_scaled + gw_scaled + inc_scaled +
                      fs_scaled + yid_scaled + fc_scaled + gw_scaled * il + fs_scaled *
                      il',
                      data = pd.DataFrame(data_scaled)).fit()
mlr_interaction.summary()
```

Variable interactions

Script

OLS Regression Results

Dep. Variable:	rev_scaled	R-squared:	0.568
Model:	OLS	Adj. R-squared:	0.564
Method:	Least Squares	F-statistic:	144.6
Date:	Fri, 24 Nov 2017	Prob (F-statistic):	1.18e-173

	coef	std err	t	P>|t|	[0.025	0.975]
Intercept	0.3615	0.161	2.245	0.025	0.045	0.678
dist_scaled	-0.4169	0.022	-18.723	0.000	-0.461	-0.373
gw_scaled	0.2702	0.073	3.716	0.000	0.128	0.413
inc_scaled	0.2875	0.051	5.690	0.000	0.188	0.387
fs_scaled	0.0929	0.074	1.248	0.212	-0.053	0.239
yid_scaled	0.2735	0.022	12.471	0.000	0.230	0.317
fc_scaled	-0.0175	0.028	-0.634	0.526	-0.072	0.037
il	-0.1602	0.071	-2.266	0.024	-0.299	-0.021
gw_scaled:il	-0.0045	0.031	-0.147	0.883	-0.065	0.056
fs_scaled:il	-0.0079	0.030	-0.262	0.793	-0.067	0.051

Variable interactions

Script

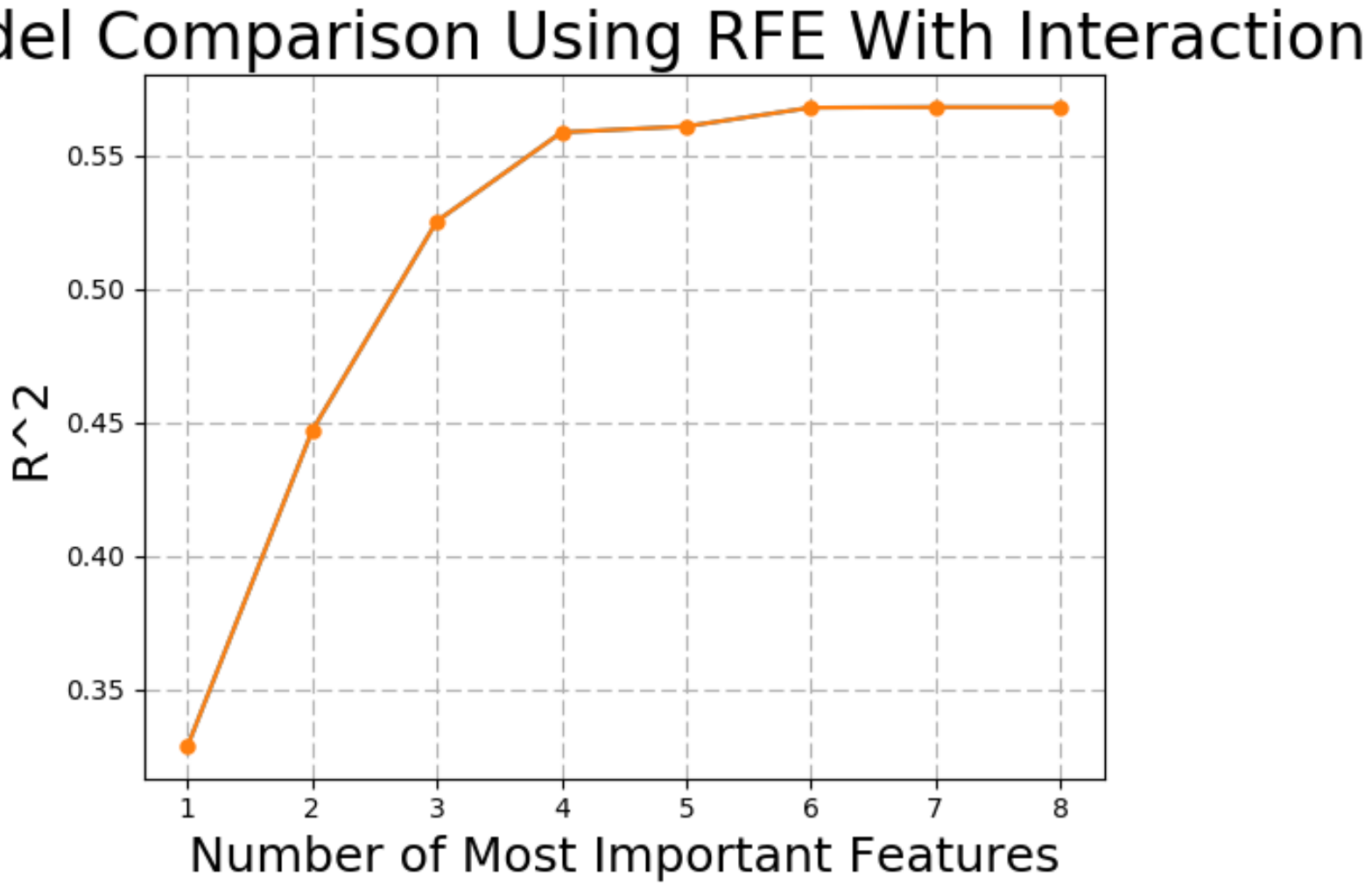
```
# To use RFE for model selection we need a LinearRegression object of sklearn
mlr_poly = preprocessing.PolynomialFeatures(2, interaction_only = True,
                                            include_bias = False)
gw_scaled_il_to_transform = np.column_stack([gw_scaled, data['IncomeLevel'].values])
gw_scaled_il_interaction = mlr_poly.fit_transform(gw_scaled_il_to_transform)

# Add the il and interaction column to our already scaled matrix of predictors
X_scaled_interaction = np.column_stack([X_scaled, gw_scaled_il_interaction[:, 1:]])


# Drop income column now that we have
mlr_interaction_sklearn = LinearRegression()
mlr_interaction_sklearn.fit(X_scaled_interaction, y_scaled)
```

Variable interactions

Script

```
R_squared_comparison = []

for i in range(1, X_scaled_interaction.shape[1] + 1):
    rfe = RFE(mlr_interaction_sklearn, i)
    rfe_fit = rfe.fit(X_scaled_interaction, np.ravel(y_scaled))
    X = X_scaled_interaction[:,np.ravel(np.where(rfe_fit.ranking_ == 1))]
    mlr = LinearRegression()
    mlr.fit(X, y_scaled)
    y_mlr_fitted = mlr.predict(X)
    R_sq = r2_score(y_scaled, y_mlr_fitted)
    R_squared_comparison.append(R_sq)

plt.plot(range(1, X_scaled_interaction.shape[1] + 1),
         R_squared_comparison,
         marker = '.',
         markersize = 10)
plt.title('Model Comparison Using RFE With Interaction', fontsize = 24)
plt.xlabel('Number of Most Important Features', fontsize = 18)
plt.ylabel('R^2', fontsize = 18)
plt.grid(.25, linestyle = 'dashed')
```

Variable interactions



Variable interactions

Script

```
# Do any of the interaction terms show up in our most important features?
# Similar to lecture 1, 4 features seems to be the optimal number without too much
# model complexity.
rfe_interaction = RFE(mlr_interaction_sklearn, 4)
rfe_interaction_fit = rfe_interaction.fit(X_scaled_interaction, np.ravel(y_scaled))
rfe_interaction_fit.ranking_
```

```
In [133]: print(rfe_interaction_fit.ranking_)
[1 1 1 3 1 4 2 5]
```

- Our best model according to RFE even with the possibility of an interaction between games watched and income level did not change.
- Perhaps the relationships in the real world are simply non-linear?

Utilize the model for prediction

Script

```
# If you decide to use this model and need to make a prediction based on
# some input variables, here is how you would do that:
# Get your testing data
test = xls.parse('Prediction')
del test['Unnamed: 0']

# Standardize test data
new_data = preprocessing.scale(test.iloc[:, 2:])

# Select DistToArena, GamesWatched, Income, YrsInDatabase
# Delete other variables
X_scaled_for_pred = np.delete(X_scaled, [3, 5], axis = 1)
new_data_for_pred = np.delete(new_data, [3, 5], axis = 1)

# First fit the model to the data then predict revenue spent with the new data.
mlr = LinearRegression()
mlr.fit(X_scaled_for_pred, y_scaled)
mlr_prediction = mlr.predict(new_data_for_pred)
```

These are your predictions for the new data!
Remember that these are standardized predictions!

Exercise time!

Other models to consider

- Should you build a separate model for each level of fan satisfaction?
- Should you build a model to understand what drives both the number of games watched and satisfaction level?
- Should you build separate models to understand what drives spending behavior for both casual and die-hard fans?
  - Could leverage this information to understand which factors don't matter to fans and don't affect their behavior?

Key things to check!

1. Outliers
2. Multicollinearity and correlation among the variables
3. Adjusted R squared
4. Model bias and distribution of residuals (Q-Q plot)
5. Standard deviation of residuals to assess model fit
6. Heteroscedasticity / pattern of residuals vs. fitted values

Prediction: use cases

- As much as 40% of trading on the London Stock Exchange is estimated to be driven by trading algorithms
- Ski manufacturers predict demand for skis each winter, stocking up on supplies
- Life insurance companies predict the age of death in order to approve policies and set pricing
- Energex (Australian utility) predicts 20 years of electricity demand growth to direct infrastructure investment
- Harrah's Hotel and Casino in Las Vegas predicts how much a customer will spend over the years, estimating their lifetime value to the casino

Source: Predictive Analytics by Eric Siegel

Recap: what we know so far

	Metric	Purpose
1	Variance	Measure of how dispersed the data is
2	Standard deviation	Standardized measure of how dispersed the data is
3	Q-Q plot / distribution of errors	Check if there is bias in the data or the model
4	Covariance	Measure of linear relationship between variables (positive / negative)
5	Correlation	Measure of strength of linear relationship between variables (positive / negative)
6	Slope	How a change in variable x will affect variable y
7	R^2	% of variation in y that can be explained by the variation in x
8	Adjusted R^2	Modified R^2 when there are many independent variables in the model
9	p-values	The probability that the pattern exists through random chance
10	VIF	Test for multicollinearity and independent variable interaction
11	Breusch-Pagan test	Check the residuals for heteroscedasticity (pattern contingent on fitted values)
12	AIC	Check for information loss when selecting the right model for your data