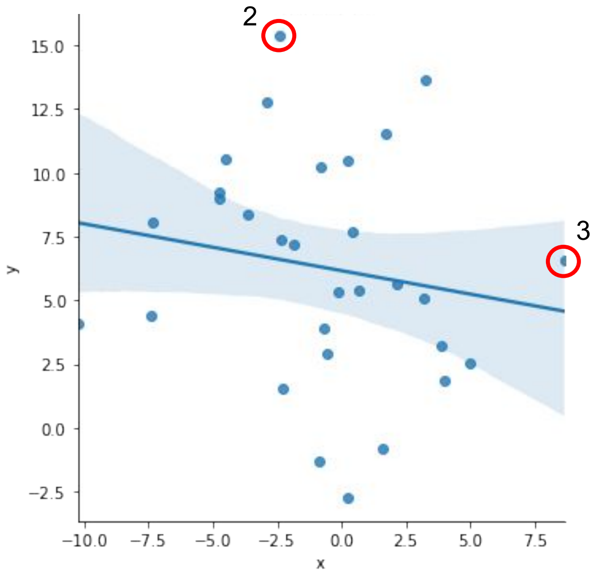
Unusual Observations

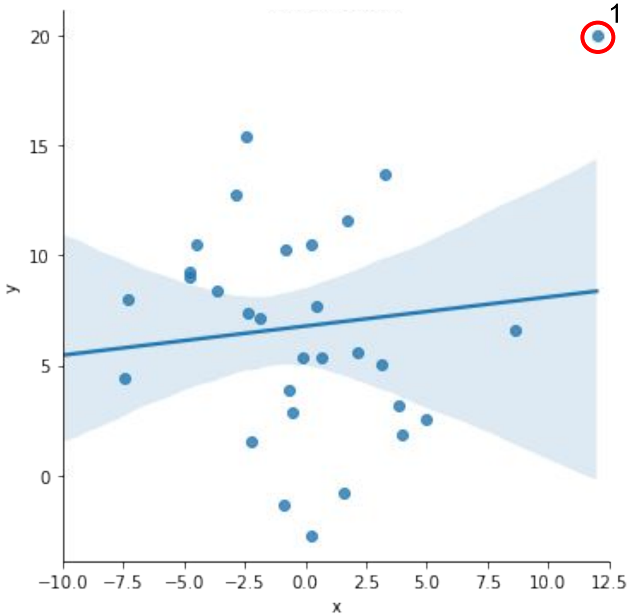
Today's Outline

- Unusual Observations
  - Leverage/Influence
  - Residual Plots
  - Studentized Residuals
  - Cook's Distance
  - DFFITS and DFBETAS
  - Removing observations

Outliers, Leverage, and Influence

- Outliers: Observations with extreme Y values
- Leverage: Observations with extreme x values have high ‘leverage’
- Influential points: Observations that drastically change our estimates of the regression coefficients
- Which is which?



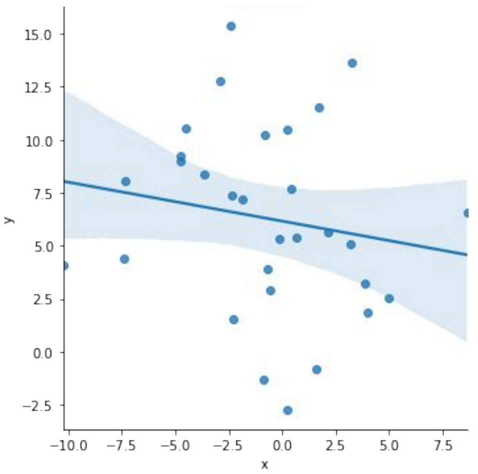


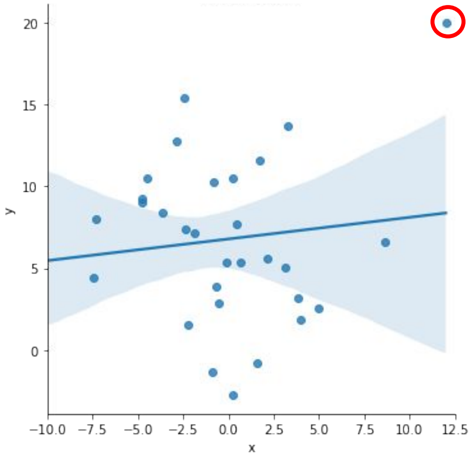
Outlier Simulation

- Outliers and otherwise influential observations (outliers with high leverage) make regression difficult
- Possible for these values to significantly change our sample estimates
- Below I simulate how a single outlier with high leverage can change results drastically

```
# create a synthetic linear random variable
# dependent
x = np.random.normal(0, 4, 30)

# error
u = np.random.normal(0,4, 30)

# slope and intercept
b0 = 5
b1 = -.25

# equation
y = b0+b1*x +u

# merge into data to create no outlier data
synth_data = pd.DataFrame(np.array([x,y]).T, columns = ["x", "y"])
```

```
# manually add an outlier to the data
new_obs = {"x": 12, "y": 20}
synth_data = synth_data.append(new_obs, ignore_index = True)
```

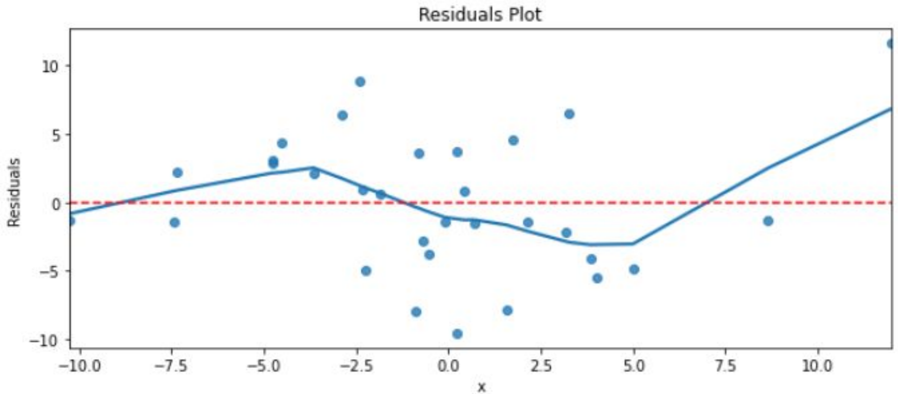




Residuals Plot (manual)

- We would like to find some reliable ways to identify influential observations that may unduly influence our estimators
- Seaborn's regplot() function can be used to generate a residuals plot with a lowess curve to identify problems
- What are we looking for in this plot?

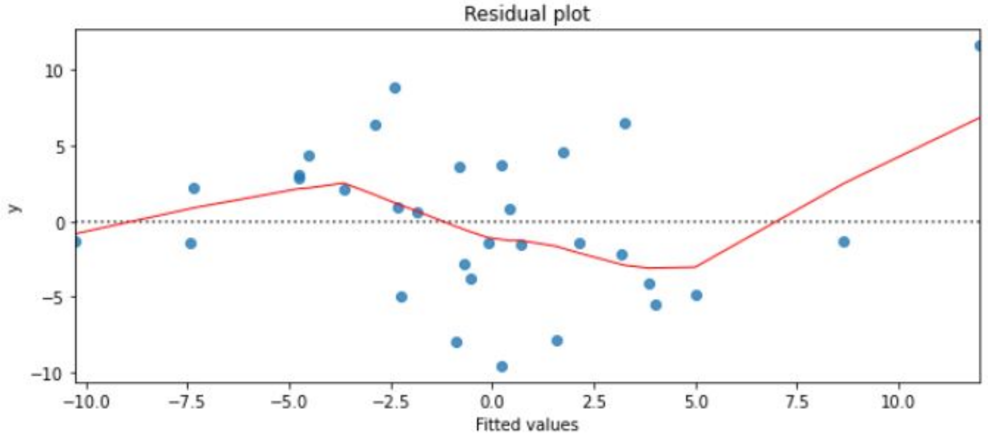
```
# fit regression to synthetic data
model_synth = smf.ols('y~x', data = synth_data)
reg_synth = model_synth.fit()

plt.figure(figsize = (10, 4))
sns.regplot(x = synth_data.x, y = reg_synth.resid, lowess = True)
plt.axhline(0, linestyle = '--', color = "red")
plt.ylabel("Residuals")
plt.title("Residuals Plot")
plt.show()
```



Residuals Plot (Seaborn Method)

- Seaborn's residplot() function will automatically perform each step for you if you supply the x and y variables

```
plt.figure(figsize = (10, 4))
sns.residplot(x = 'x', y = 'y', data=synth_data,lowess=True,
                                  line_kws={'color': 'red', 'lw': 1, 'alpha': 1})
plt.xlabel("Fitted values")
plt.title('Residual plot')
plt.show()
```



Studentized Residuals

- Sometimes outliers are so influential that they will pull the regression line up towards them
- This can make an outlier stand out much less than they otherwise should on a normal or standardized residual plot
- Studentized residuals compensate for this by showing how residuals would look if a regression was fit without that observation

```
# we can pill the studentized residuals from teh regression results
# The get_influence() method gives us access to many different attributes
# to analyze unusual observations
studentized_resid = reg_synth.get_influence().resid_studentized
studentized_resid
```

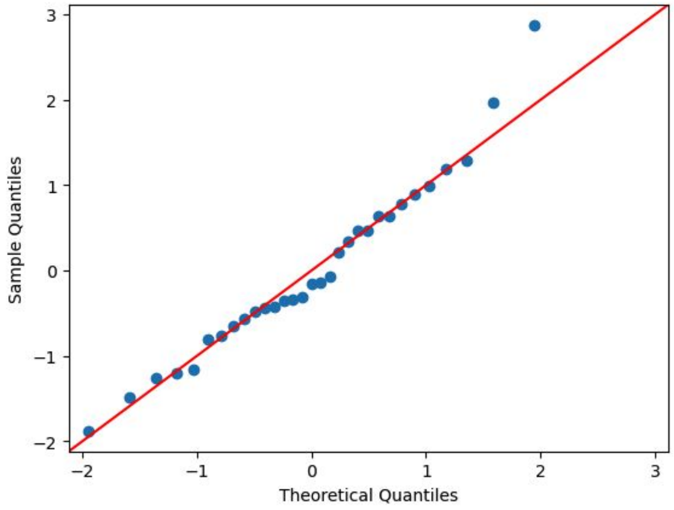
```
array([ 0.71678679,  0.17366805, -1.00700286,  1.28449929, -0.29130677,
        0.42149774,  0.12624827, -0.83218575,  1.31063067,  0.62513629,
        0.91530729, -1.11578704, -1.60691464,  1.79787371, -1.00292179,
        0.46394048,  0.57866064, -0.29573264, -1.57566548, -0.29794954,
       -0.30487763,  0.74255762, -0.56694208,  0.18624098, -1.92342092,
       -0.2881979 , -0.76742258,  0.88521679, -0.29335177, -0.4334391 ,
        2.69942689])
```

Studentized Residuals Q-Q Plot

- We can plot the studentized residuals against the corresponding quantiles of a t-distribution
- The t-distribution is parameterized with n-k-2 degrees of freedom
- Std. residuals > 2 are considered large

```python
import scipy.stats as stats

# get the degrees of freedom where k = 2 for each regression parameter
df = len(studentized_resid)-4

# use scipy stats to create distribution
t_dist = stats.t(df)
```

```python
import statsmodels.api as sm

sm.qqplot(studentized_resid, line='45', dist = t_dist)
plt.show()
```

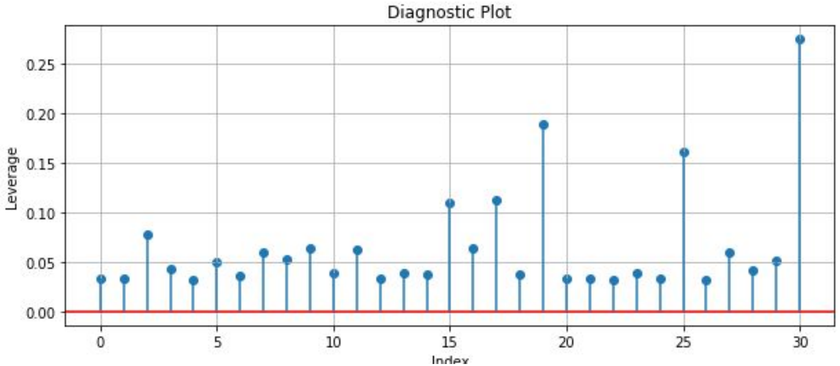


*Note this method of building qqplot is different than before, may be easier

Leverage Plot (Hat-Values Plot)

- Calculating an observation's leverage can help us identify observations with extreme x-values
- We typically will flag values that are two or three times larger than the mean hat value

```python
leverage = reg_synth.get_influence().hat_matrix_diag

plt.figure(figsize = (10, 4))
plt.scatter(synth_data.index, leverage)
plt.axhline(0, color = 'red')
plt.vlines(x = synth_data.index, ymin = 0, ymax = leverage)
plt.xlabel('Index')
plt.ylabel('Leverage')
plt.title("Diagnostic Plot")
plt.grid()
```



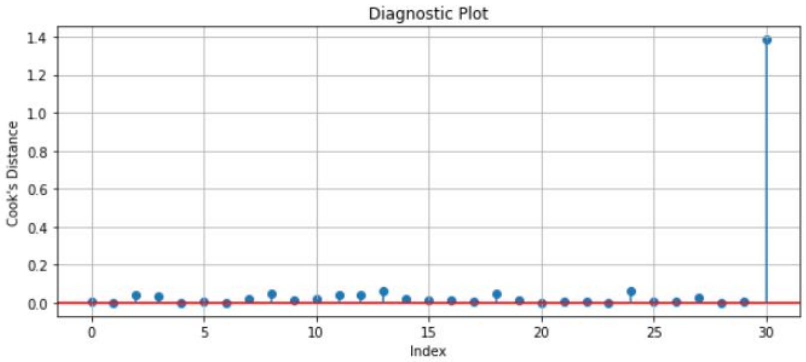
Cook’s Distance

- The standardized residuals and leverage can be combined into a more complete measure of “influence”
- Large values of D_i (more than 1 and/or larger relatively) are influential, some studies also prefer 4/n

$$D_i = \frac{e_{Si}^2}{k+1} \times \frac{h_i}{1-h_i}$$

```
# Get influence calculation of leverage
cooks_distance = reg_synth.get_influence().cooks_distance
```

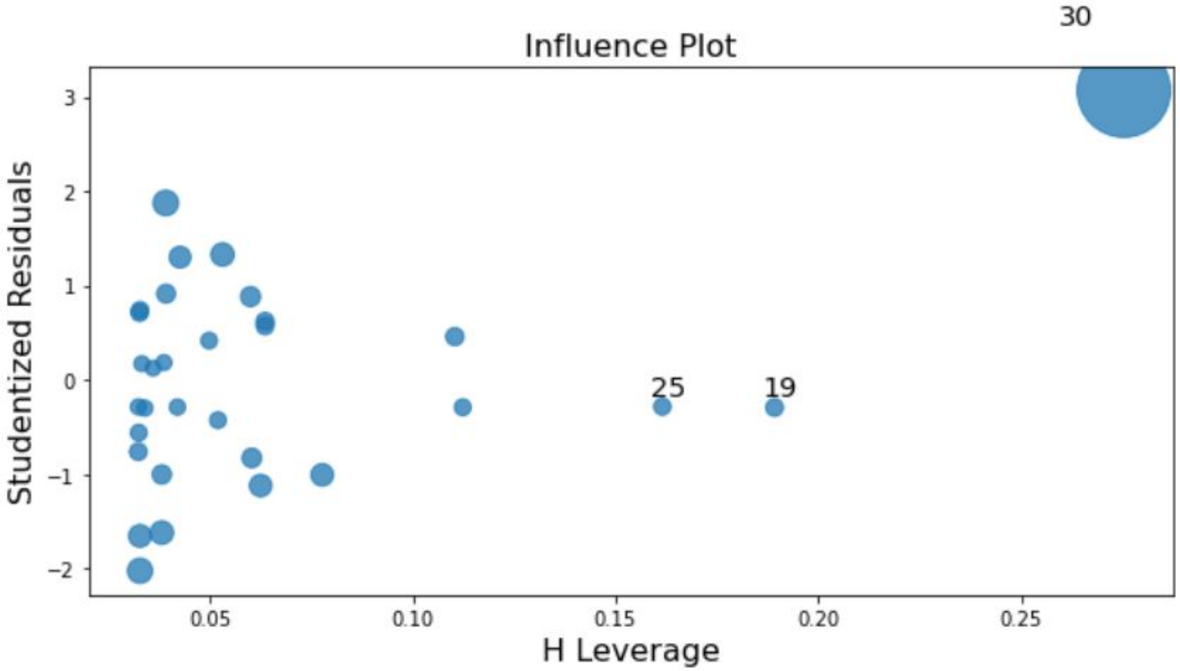
```
plt.figure(figsize = (10, 4))
plt.scatter(synth_data.index, cooks_distance[0])
plt.axhline(0, color = 'red')
plt.vlines(x = synth_data.index, ymin = 0, ymax = cooks_distance[0])
plt.xlabel('Index')
plt.ylabel('Cook\'s Distance')
plt.title("Diagnostic Plot")
plt.grid()
```



Combining Metrics (Cook's)

- statsmodels.graphics submodule allows us to create complex influence plots
- This combines three different measures
  - Studentized residuals
  - Leverage (hat values)
  - Cook's Distance
- The index of each observation is included on the plot

```
import statsmodels.api as sm
fig, ax = plt.subplots(figsize=(10,5))
fig = sm.graphics.influence_plot(reg_synth, ax = ax, criterion="cooks")
```

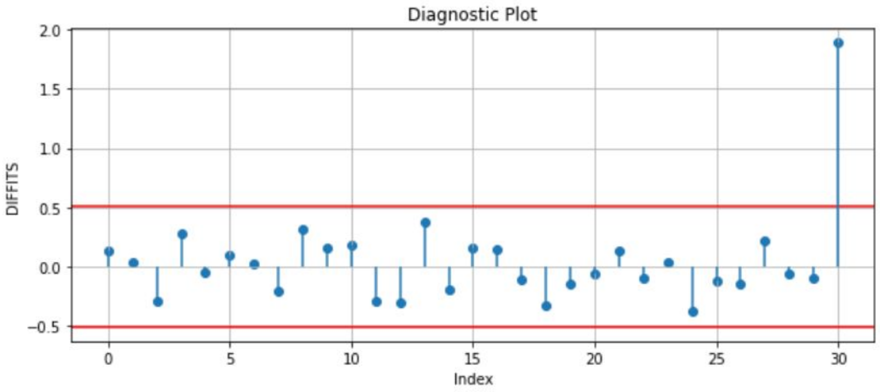


DFFITS

- DFFITS is another method used for identifying influential observations by computing the difference between the fitted values with and without a given observation
- The get_influence() method automatically returns the DFFITS values for a regression as well as a threshold over which we may want to investigate an observation

```python
# the dffits attribute returns a cutoff and the diagnostic values
dffits, threshold = reg_synth.get_influence().dffits
```

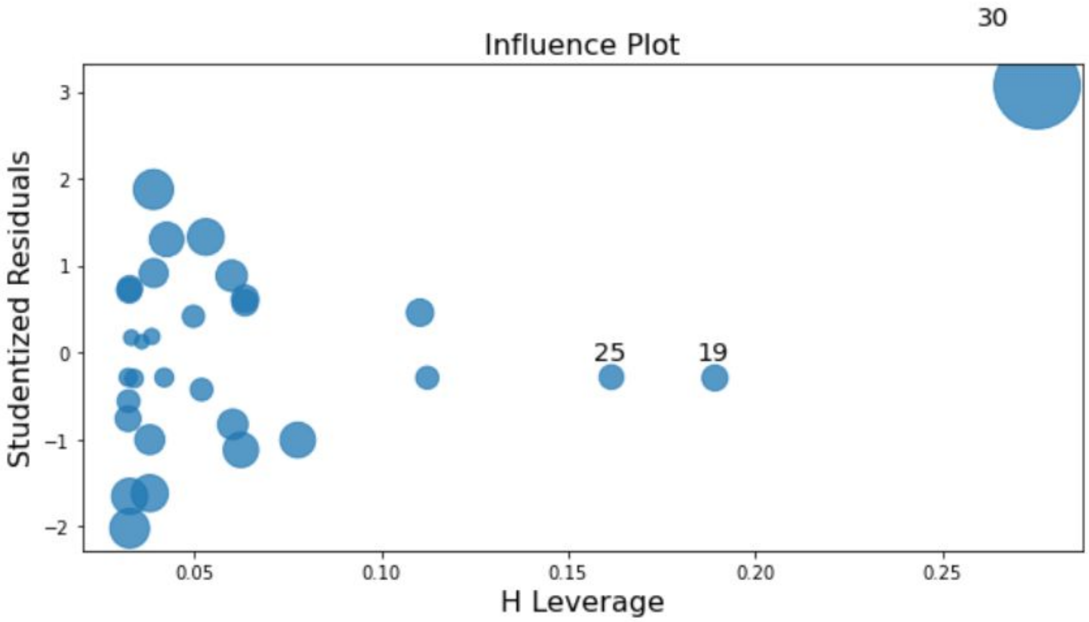
```python
plt.figure(figsize = (10, 4))
plt.scatter(synth_data.index, dffits)
plt.axhline(threshold, color = 'red')
plt.axhline(-threshold, color = 'red')
plt.vlines(x = synth_data.index, ymin = 0, ymax = dffits)
plt.xlabel('Index')
plt.ylabel('DIFFITS')
plt.title("Diagnostic Plot")
plt.grid()
```



Combining Metrics (DFFITS)

- The DFFITS criterion can be used in the statsmodels influence plot instead of Cook's Distance

```
fig, ax = plt.subplots(figsize=(10,5))
fig = sm.graphics.influence_plot(reg_synth, ax = ax, criterion="DFFITS")
```



DFBETAS

- DFFITS is another method used for identifying how influential an observation is on a given parameter
- The get_influence() method automatically returns the DFBETA values for a regression
- One possible cutoff found in the literature over which we may want to investigate is:

$$\frac{2}{\sqrt{n}}$$

```
reg_synth.get_influence().dfbeta[:5,:]
```

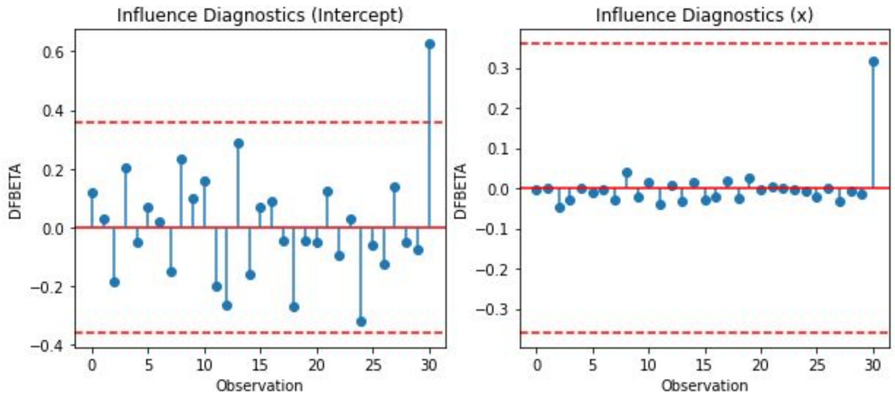
```
array([[ 0.11796804, -0.00282942],
       [ 0.02920481,  0.00110016],
       [-0.18665747, -0.04515436],
       [ 0.20496706, -0.02701122],
       [-0.04852522, -0.00056373]])
```

```
# pull out the dfbeta values for each coefficient
dfb_intercepts = reg_synth.get_influence().dfbeta[:,0]
dfb_x = reg_synth.get_influence().dfbeta[:,1]

# calculate the threshold using 2/sqrt(n)
thresh = 2/np.sqrt(len(reg_synth.fittedvalues))
```

Plotting DFBETAS

- We can plot the DFBETAS along with the thresholds in matplotlib
- Any observations that cross over the threshold for our variables merit investigation

```
fig, ax = plt.subplots(1,2, figsize = (10, 4))

ax[0].scatter(reg_synth.fittedvalues.index, dfb_intercepts)
ax[0].axhline(-thresh, color = "red", linestyle = '--')
ax[0].axhline(thresh, color = "red", linestyle = '--')
ax[0].axhline(0, color = "red")
ax[0].vlines(x = synth_data.index, ymin = 0, ymax = dfb_intercepts)
ax[0].set_title("Influence Diagnostics (Intercept)")
ax[0].set_xlabel("Observation")
ax[0].set_ylabel("DFBETA")

ax[1].scatter(reg_synth.fittedvalues.index, dfb_x)
ax[1].axhline(-thresh, color = "red", linestyle = '--')
ax[1].axhline(thresh, color = "red", linestyle = '--')
ax[1].vlines(x = synth_data.index, ymin = 0, ymax = dfb_x)
ax[1].axhline(0, color = "red")
ax[1].set_title("Influence Diagnostics (x)")
ax[1].set_xlabel("Observation")
ax[1].set_ylabel("DFBETA")

plt.show()
```



Removing Unusual Observations

- Once we have identified unusual observations, we may decide that some should be removed (first check these aren't recording errors)
- This can be done in loops or by simple conditional slicing
- For each method below the diagnostic values have an index corresponding to observations in the original data

```
# boolean slicing method
# only observatuons with a diagnostic value less than the cutoff
# note that we sometimes may need to use the absolute value of the diagnostic values

data[diagnostic_values<cutoff]

# drop observations greater than the cooks cutoff
drop_indices = [i for i,v in enumerate(diagnostic_values) if v > cutoff]
data.drop(drop_indices)
```

- Once influential observations are removed we can fit a regression, check measures of fit (R2, AIC, BIC) or check predictive accuracy out of sample
- We should always **report when outliers are removed**, and may also want to report model results with and without them

Unusual Observations Exercise

- Using the wage data fit the following regression:

$$wage = \beta_0 + \beta_1 educ + u$$

- Create the following diagnostic plots:
  - Residuals plot
  - QQ plot with Studentized residuals
  - Cook's Distance Plot (index vs Cook's distance)
  - DFFITS plot (index vs DFFITS)
- Pick two measures of influence and use them to drop influential observations using the corresponding recommended cutoff (Cook's may use 4/n)
- Fit a new regression for each new dataframe, how does the regression fit change?