In [1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import statsmodels.api as sm
from statsmodels.stats.outliers_influence import OLSInfluence
```

In [2]:
```
df = pd.read_csv('Carseats.csv')
df_copy = df.copy()
```

In [3]:
```
# Extract only the necessary variables
df_copy = df_copy[['Sales', 'Price', 'Urban', 'US']]

# Replace all 'Yes' and 'No' with 1 and 0 respectively
df_copy['Urban'] = df_copy['Urban'].replace({'Yes': 1.0, 'No': -1.0})
df_copy['US'] = df_copy['US'].replace({'Yes': 1.0, 'No': -1.0})
```

Multiple Linear Regression Model 1

Problem a

In [4]:

```
# Extracts the predictor and response columns, and creates design matrix
columns1 = ['Price', 'Urban', 'US']
X_var1 = np.asarray(df_copy[columns1]) # Extracts the variables as an array to use as predictor
y_true = np.asarray(df_copy[['Sales']]) # Extracts the mpg variable as an array to use as response
X_design1 = sm.add_constant(X_var1)

# Create the Multiple Linear Regression Model and fit it
MLRmodel1 = sm.OLS(y_true, X_design1)
result1 = MLRmodel1.fit()

columns1.insert(0,'Intercept')
# Create table of model prediction results
data1 = {'Attributes': columns1,
        'Coefficient Beta_i': result1.params,
        't-Values': result1.tvalues,
        'p-Values': result1.pvalues
       }
ModelResults1 = pd.DataFrame(data1)
ModelResults1.round(4) # Round values in table to 4-decimal places
```

Out[4]:

	Attributes	Coefficient Beta_i	t-Values	p-Values
0	Intercept	13.6328	22.0433	0.0000
1	Price	-0.0545	-10.3892	0.0000
2	Urban	-0.0110	-0.0807	0.9357
3	US	0.6003	4.6347	0.0000

Multiple Linear Regression Model 2

Problem e

In [5]:
```
# Extracts the predictor and response columns, and creates design matrix
new_columns = ['Price', 'US']
X_var2 = np.asarray(df_copy[new_columns]) # Extracts the variables as an array to use as predictor
y_true = np.asarray(df_copy[['Sales']]) # Extracts the mpg variable as an array to use as response
X_design2 = sm.add_constant(X_var2)

# Create the Multiple Linear Regression Model and fit it
MLRmodel2 = sm.OLS(y_true, X_design2)
result2 = MLRmodel2.fit()

new_columns.insert(0,'Intercept')
# Create table of model prediction results
data2 = {'Attributes': new_columns,
        'Coefficient Beta_i': result2.params,
        't-Values': result2.tvalues,
        'p-Values': result2.pvalues
       }

ModelResults2 = pd.DataFrame(data2)
ModelResults2.round(4) # Round values in table to 4-decimal places
```

Out[5]:

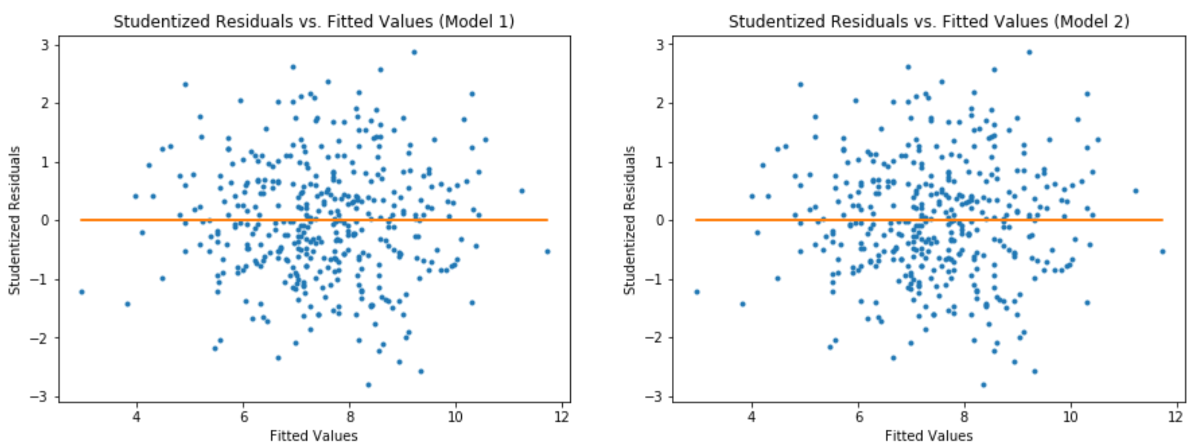
	Attributes	Coefficient Beta_i	t-Values	p-Values
0	Intercept	13.6306	22.0885	0.0
1	Price	-0.0545	-10.4161	0.0
2	US	0.5998	4.6415	0.0

Problem f

Comparison Between Models 1 and 2

In [6]:
```
y_pred1 = result1.predict(X_design1)
y_pred1 = np.reshape(y_pred1, (-1, 1))

residuals1 = y_true - y_pred1

influence1 = OLSInfluence(result1)
studentized_residuals1 = influence1.resid_studentized_internal


y_pred2 = result2.predict(X_design2)
y_pred2 = np.reshape(y_pred2, (-1, 1))

residuals2 = y_true - y_pred2

influence2 = OLSInfluence(result2)
studentized_residuals2 = influence2.resid_studentized_internal

f = plt.figure(figsize=(15,5))

# Create Studentized Resisuals vs. Fitted Values Plot For Model 1
ax = f.add_subplot(121)
ax.plot(y_pred1, studentized_residuals1, '.') # Plots the residual points
ax.plot(y_pred1, 0*y_pred1) # Plots the zero error line
ax.set_title('Studentized Residuals vs. Fitted Values (Model 1)')
ax.set_xlabel('Fitted Values')
ax.set_ylabel('Studentized Residuals')

# Create Studentized Resisuals vs. Fitted Values Plot For Model 2
ax2 = f.add_subplot(122)
ax2.plot(y_pred2, studentized_residuals2, '.') # Plots the residual points
ax2.plot(y_pred2, 0*y_pred2) # Plots the zero error line
ax2.set_title('Studentized Residuals vs. Fitted Values (Model 2)')
ax2.set_xlabel('Fitted Values')
ax2.set_ylabel('Studentized Residuals')
```

Out[6]:
```
Text(0,0.5,'Studentized Residuals')
```



Confidence Intervals

Problem g

In [7]:
```
conf_intervals2 = result2.conf_int(alpha=0.05)
conf_intervals2 = conf_intervals2.round(4)
# conf_intervals2.shape
print("Confidence Interval for Intercept: [" + repr(conf_intervals2[0][0]) +
", " + repr(conf_intervals2[0][1]) + "]")
print("Confidence Interval for Price: [" + repr(conf_intervals2[1][0]) + ", "
+ repr(conf_intervals2[1][1]) + "]")
print("Confidence Interval for US: [" + repr(conf_intervals2[2][0]) + ", " + r
epr(conf_intervals2[2][1]) + "]")
```

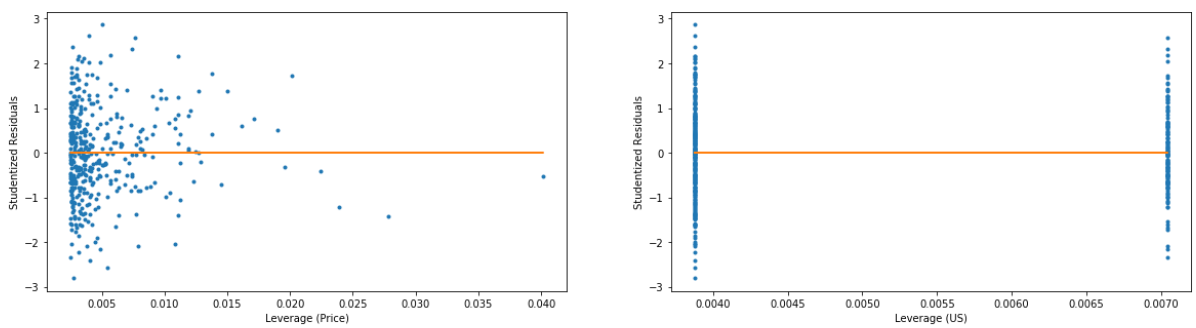
```
Confidence Interval for Intercept: [12.4174, 14.8438]
Confidence Interval for Price: [-0.0648, -0.0442]
Confidence Interval for US: [0.3458, 0.8539]
```

Problem h

In [8]:
```
def leveragePoints(predictorValues):
    # predictorValues is a list object
    numValues = len(predictorValues)
    output = np.zeros_like(predictorValues, dtype=float)
    mean = predictorValues.mean()

    TSS = 0.0
    for num in predictorValues:
        TSS = TSS + (num - mean)**2

    for i in range(numValues):
        output[i] = 1/numValues + ((predictorValues[i] - mean)**2)/(TSS)
    return output
```

In [9]:
```
f = plt.figure(figsize=(20,5))

# Create Plot for Price
ax = f.add_subplot(121)
Levs1 = leveragePoints(np.asarray(df_copy['Price']))
ax.plot(Levs1, studentized_residuals2, '.') # Plots the data points
ax.plot(Levs1, 0*Levs1) # Plots the linear regression line
# ax.set_title('Studentized Residuals vs. Leverage (cylinders)')
ax.set_xlabel('Leverage (Price)')
ax.set_ylabel('Studentized Residuals')

# Create Plot for US
ax2 = f.add_subplot(122)
Levs2 = leveragePoints(np.asarray(df_copy['US']))
ax2.plot(Levs2, studentized_residuals2, '.') # Plots the data points
ax2.plot(Levs2, 0*Levs2) # Plots the linear regression line
# ax2.set_title('Studentized Residuals vs. Leverage (displacement)')
ax2.set_xlabel('Leverage (US)')
ax2.set_ylabel('Studentized Residuals')
```

Out[9]:
```
Text(0,0.5,'Studentized Residuals')
```



In [ ]: