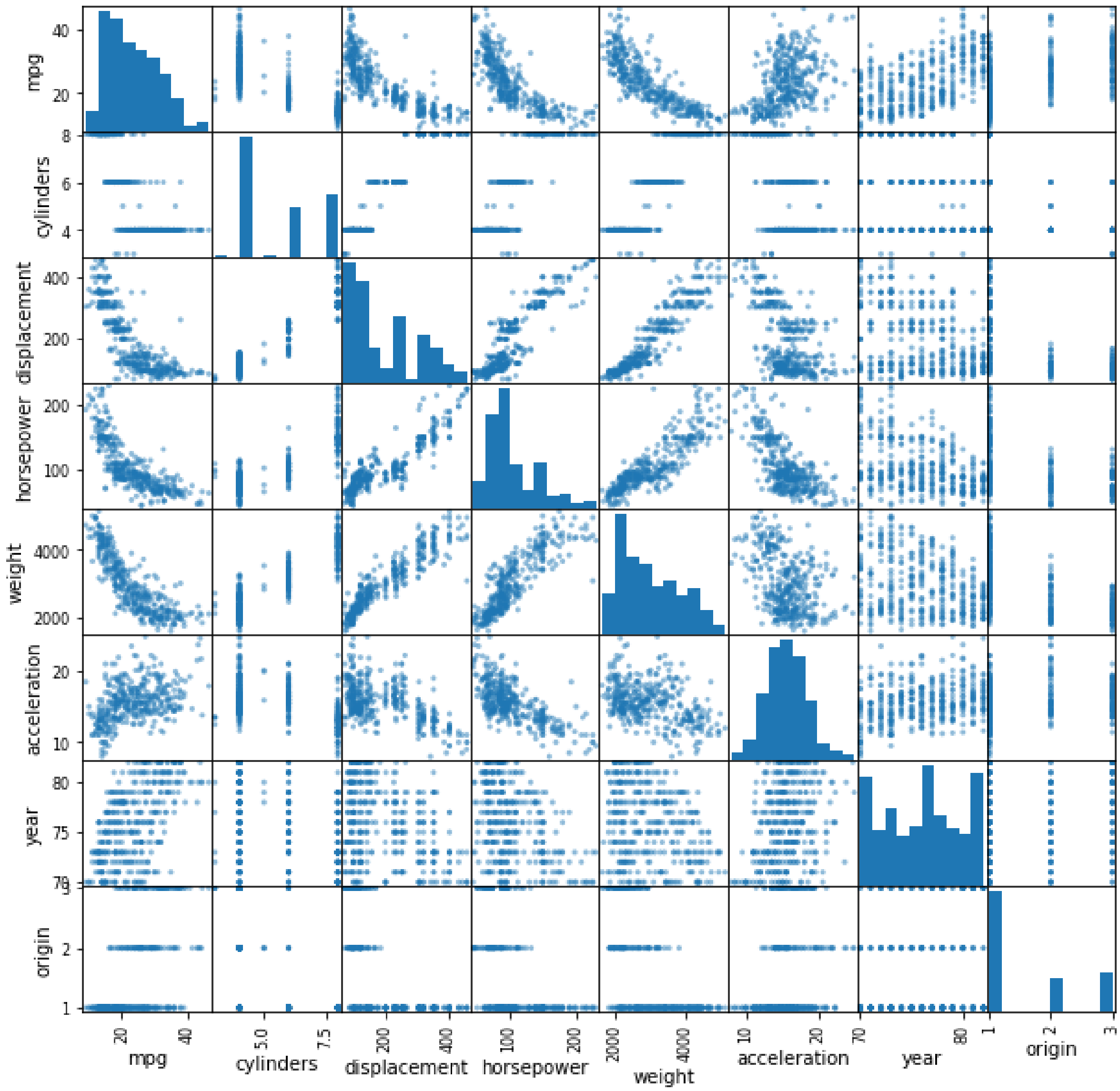
In [1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import statsmodels.api as sm
from sklearn.linear_model import LinearRegression
from statsmodels.stats.outliers_influence import OLSInfluence
```

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

In [3]:
```
# Eliminates the rows (instances) with '?' as a predictor value
df_copy['horsepower'] = pd.to_numeric(df_copy['horsepower'], errors='coerce')
df_copy = df_copy.dropna()
# df_copy['name'] = df_copy['name'].str.split(' ').str[0]

df_copy = df_copy.drop('name', 1)
```

Problem a

In [4]:
```
# Produces the scatterplot matrix
pd.plotting.scatter_matrix(df_copy, figsize=(10,10));
```



Problem b

In [5]:
```
# Produces the correlation matrix
df_copy.corr()
```

Out[5]:

	mpg	cylinders	displacement	horsepower	weight	acceleration	year
mpg	1.000000	-0.777618	-0.805127	-0.778427	-0.832244	0.423329	0.580541
cylinders	-0.777618	1.000000	0.950823	0.842983	0.897527	-0.504683	-0.345647
displacement	-0.805127	0.950823	1.000000	0.897257	0.932994	-0.543800	-0.369855
horsepower	-0.778427	0.842983	0.897257	1.000000	0.864538	-0.689196	-0.416361
weight	-0.832244	0.897527	0.932994	0.864538	1.000000	-0.416839	-0.309120
acceleration	0.423329	-0.504683	-0.543800	-0.689196	-0.416839	1.000000	0.290316
year	0.580541	-0.345647	-0.369855	-0.416361	-0.309120	0.290316	1.000000
origin	0.565209	-0.568932	-0.614535	-0.455171	-0.585005	0.212746	0.181528

Problem c

In [6]:
```
# MULTIPLE LINEAR REGRESSION MODEL



# Extracts the predictor and response columns, and creates design matrix
columns = ['cylinders', 'displacement', 'horsepower', 'weight', 'acceleration'
, 'year', 'origin']
X_var = np.asarray(df_copy[columns]) # Extracts the variables as an array to u
se as predictor
y_true = np.asarray(df_copy[['mpg']]) # Extracts the mpg variable as an array
 to use as response
X_design = sm.add_constant(X_var)

# Create the Multiple Linear Regression Model and fit it
MLRmodel = sm.OLS(y_true, X_design)
result = MLRmodel.fit()

# Create table of model prediction results
data = {'Coefficient Beta_i': result.params,
        't-Values': result.tvalues,
        'p-Values': result.pvalues
       }
df2 = pd.DataFrame(data)
df2.round(4) # Round values in table to 4-decimal places
```

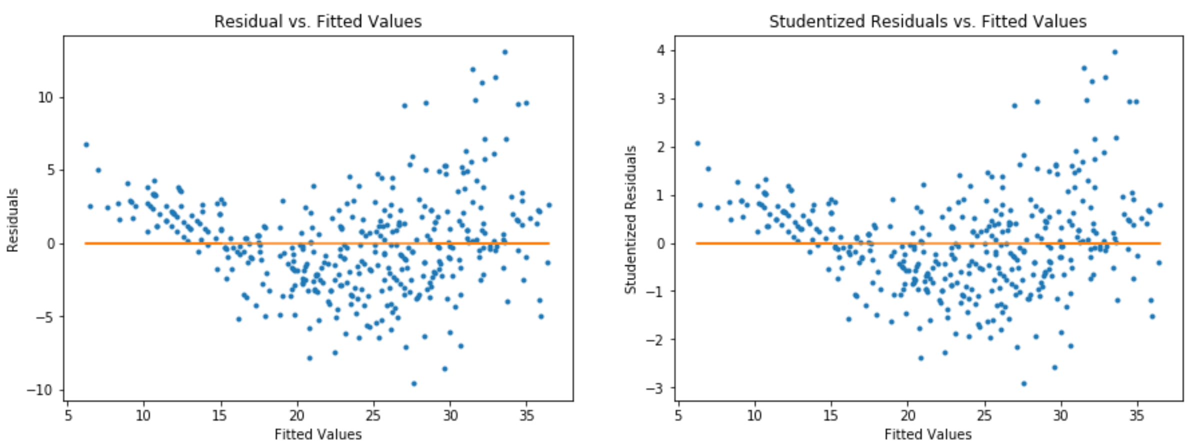
Out[6]:

	Coefficient Beta_i	t-Values	p-Values
0	-17.2184	-3.7074	0.0002
1	-0.4934	-1.5261	0.1278
2	0.0199	2.6474	0.0084
3	-0.0170	-1.2295	0.2196
4	-0.0065	-9.9288	0.0000
5	0.0806	0.8152	0.4155
6	0.7508	14.7288	0.0000
7	1.4261	5.1275	0.0000

Problem d

In [7]:
```
y_pred = result.predict(X_design)
y_pred = np.reshape(y_pred, (-1, 1))
residuals = y_true - y_pred

influence = OLSInfluence(result)
studentized_residuals = influence.resid_studentized_internal

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

# Create Residuals vs. Fitted Values Plot
ax = f.add_subplot(121)
ax.plot(y_pred, residuals, '.') # Plots the residual points
ax.plot(y_pred, 0*y_pred) # Plots the zero error line
ax.set_title('Residual vs. Fitted Values')
ax.set_xlabel('Fitted Values')
ax.set_ylabel('Residuals')

# Create Studentized Resisuals vs. Fitted Values Plot

ax2 = f.add_subplot(122)
ax2.plot(y_pred, studentized_residuals, '.') # Plots the residual points
ax2.plot(y_pred, 0*y_pred) # Plots the zero error line
ax2.set_title('Studentized Residuals vs. Fitted Values')
ax2.set_xlabel('Fitted Values')
ax2.set_ylabel('Studentized Residuals')
```

Out[7]: Text(0,0.5,'Studentized Residuals')

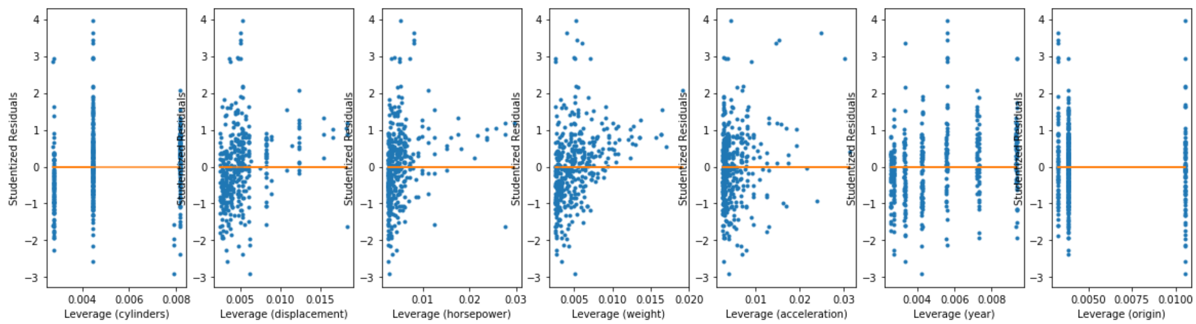


In [8]:
```
def leveragePoints(predictorValues):
    # predictorValues is a list object
    numValues = len(predictorValues)
    output = np.zeros_like(predictorValues, dtype=float)
    mean = predictorValues.mean()

    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]:
```
# Create Studentized Results vs. Leverage Points Plots

f = plt.figure(figsize=(20,5))

# Create Plot for cylinders
ax = f.add_subplot(171)
Levs1 = leveragePoints(np.asarray(df_copy['cylinders']))
ax.plot(Levs1, studentized_residuals, '.') # 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 (cylinders)')
ax.set_ylabel('Studentized Residuals')

# Create Plot for displacement
ax2 = f.add_subplot(172)
Levs2 = leveragePoints(np.asarray(df_copy['displacement']))
ax2.plot(Levs2, studentized_residuals, '.') # 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 (displacement)')
ax2.set_ylabel('Studentized Residuals')

# Create Plot for horsepower
ax3 = f.add_subplot(173)
Levs3 = leveragePoints(np.asarray(df_copy['horsepower']))
ax3.plot(Levs3, studentized_residuals, '.') # Plots the data points
ax3.plot(Levs3, 0*Levs3) # Plots the linear regression line
# ax3.set_title('Studentized Residuals vs. Leverage (horsepower)')
ax3.set_xlabel('Leverage (horsepower)')
ax3.set_ylabel('Studentized Residuals')

# Create Plot for weight
ax4 = f.add_subplot(174)
Levs4 = leveragePoints(np.asarray(df_copy['weight']))
ax4.plot(Levs4, studentized_residuals, '.') # Plots the data points
ax4.plot(Levs4, 0*Levs4) # Plots the linear regression line
# ax4.set_title('Studentized Residuals vs. Leverage (weight)')
ax4.set_xlabel('Leverage (weight)')
ax4.set_ylabel('Studentized Residuals')

# Create Plot for acceleration
ax5 = f.add_subplot(175)
Levs5 = leveragePoints(np.asarray(df_copy['acceleration']))
ax5.plot(Levs5, studentized_residuals, '.') # Plots the data points
ax5.plot(Levs5, 0*Levs5) # Plots the linear regression line
# ax5.set_title('Studentized Residuals vs. Leverage (acceleration)')
ax5.set_xlabel('Leverage (acceleration)')
ax5.set_ylabel('Studentized Residuals')

# Create Plot for year
ax6 = f.add_subplot(176)
Levs6 = leveragePoints(np.asarray(df_copy['year']))
ax6.plot(Levs6, studentized_residuals, '.') # Plots the data points
ax6.plot(Levs6, 0*Levs6) # Plots the linear regression line
# ax6.set_title('Studentized Residuals vs. Leverage (year)')
ax6.set_xlabel('Leverage (year)')
```

```
ax6.set_ylabel('Studentized Residuals')

# Create Plot for cylinders
ax7 = f.add_subplot(177)
Lev7 = leveragePoints(np.asarray(df_copy['origin']))
ax7.plot(Lev7, studentized_residuals, '.') # Plots the data points
ax7.plot(Lev7, 0*Lev7) # Plots the linear regression line
# ax7.set_title('Studentized Residuals vs. Leverage (origin)')
ax7.set_xlabel('Leverage (origin)')
ax7.set_ylabel('Studentized Residuals')
```

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



Problem e

In [10]:
```
df_copy2 = df_copy.copy()
# df_copy2['year*origin'] = df_copy2['year']*df_copy2['origin']
df_copy2['cylinders*horsepower'] = df_copy2['cylinders']*df_copy2['horsepower'
]
# df_copy2['displacement*acceleration'] = df_copy2['displacement']*df_copy2['a
cceleration']


# Extract Column Names as List in Pandas Dataframe
col_names = df_copy2.columns.tolist()
col_names[0] = 'Intercept'

X_var = np.asarray(df_copy2.loc[:, 'cylinders':]) # Extracts the horsepower va
riable as an array to use as predictor
y_true = np.asarray(df_copy2[['mpg']]) # Extracts the mpg variable as an array
to use as response
X_design = sm.add_constant(X_var)

# Create the Multiple Linear Regression Model and fit it
MLRmodel = sm.OLS(y_true, X_design)
result = MLRmodel.fit()

# Create table of model prediction results
data = {'Beta_i': result.params,
#          't-Values': result.tvalues,
        'p-Values': result.pvalues
       }
df2 = pd.DataFrame(data)
df2 = df2.round(4) # Round values in table to 4-decimal places
df2.insert(0, 'Attributes', col_names)
df2
```

Out[10]:

	Attributes	Beta_i	p-Values
0	Intercept	11.7025	0.0177
1	cylinders	-4.3061	0.0000
2	displacement	-0.0014	0.8404
3	horsepower	-0.3157	0.0000
4	weight	-0.0039	0.0000
5	acceleration	-0.1703	0.0596
6	year	0.7393	0.0000
7	origin	0.9032	0.0003
8	cylinders*horsepower	0.0402	0.0000

In [11]:
```
df_copy2 = df_copy.copy()
# df_copy2['year*origin'] = df_copy2['year']*df_copy2['origin']
# df_copy2['cylinders*horsepower'] = df_copy2['cylinders']*df_copy2['horsepowe
r']
df_copy2['displacement*acceleration'] = df_copy2['displacement']*df_copy2['acc
eleration']


# Extract Column Names as List in Pandas Dataframe
col_names = df_copy2.columns.tolist()
col_names[0] = 'Intercept'

X_var = np.asarray(df_copy2.loc[:, 'cylinders':]) # Extracts the horsepower va
riable as an array to use as predictor
y_true = np.asarray(df_copy2[['mpg']]) # Extracts the mpg variable as an array
to use as response
X_design = sm.add_constant(X_var)

# Create the Multiple Linear Regression Model and fit it
MLRmodel = sm.OLS(y_true, X_design)
result = MLRmodel.fit()

# Create table of model prediction results
data = {'Beta_i': result.params,
#          't-Values': result.tvalues,
        'p-Values': result.pvalues
       }
df2 = pd.DataFrame(data)
df2 = df2.round(4) # Round values in table to 4-decimal places
df2.insert(0, 'Attributes', col_names)
df2
```

Out[11]:

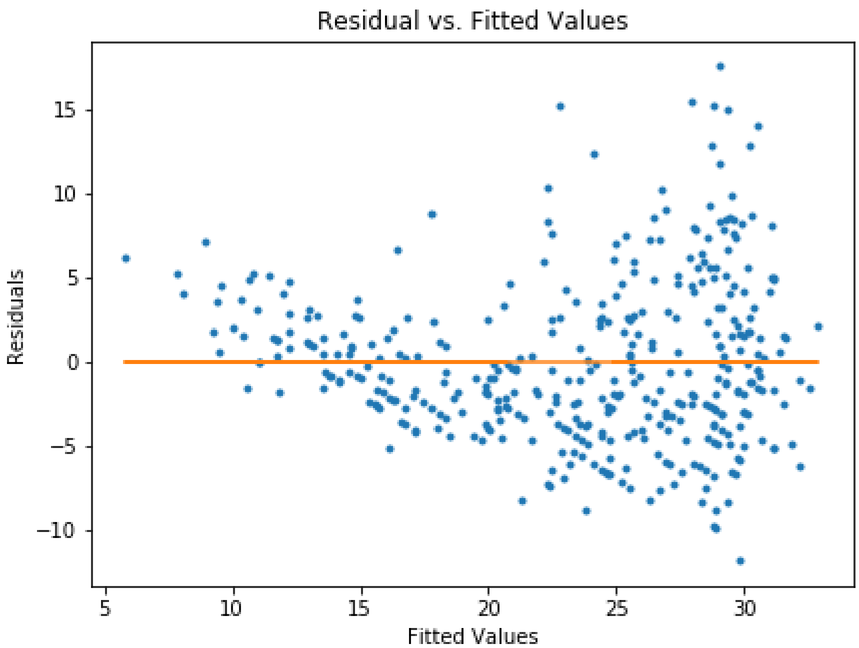
	Attributes	Beta_i	p-Values
0	Intercept	-30.0482	0.0000
1	cylinders	0.0021	0.9946
2	displacement	0.0702	0.0000
3	horsepower	-0.0551	0.0001
4	weight	-0.0042	0.0000
5	acceleration	0.7530	0.0000
6	year	0.7722	0.0000
7	origin	1.0573	0.0001
8	displacement*acceleration	-0.0049	0.0000

Problem f

In [12]:
```
# Creates dataframe with interactions
data = {
        'cylinders*horsepower*acceleration': df_copy['cylinders']*df_copy['hor
sepower']*df_copy['horsepower'],
#         'cylinders*acceleration': df_copy['cylinders']*df_copy['acceleratio
n'],
#         'weight*acceleration': df_copy['weight']*df_copy['acceleration'],
#         'year*origin': df_copy['year']*df_copy['origin'],
        'year*weight': df_copy['year']*df_copy['weight'],
       }
df_temp = pd.DataFrame(data)

# Create design matrix
X_var = np.asarray(df_temp)
X_design = sm.add_constant(X_var)

column_names = list(df_temp.columns)
column_names.insert(0, 'Intercept')

# Create the Simple Linear Regression Model and fit it
MLRmodel = sm.OLS(y_true, X_design)
result = MLRmodel.fit()

# Use fitted model to make predictions and residuals
y_pred = result.predict(X_design)
y_pred = np.reshape(y_pred, (-1, 1))
residuals = y_true - y_pred


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

# Create Residuals vs. Fitted Values Plot
ax = f.add_subplot(121)
ax.plot(y_pred, residuals, '.') # Plots the residual points
ax.plot(y_pred, 0*y_pred) # Plots the zero error line
ax.set_title('Residual vs. Fitted Values')
ax.set_xlabel('Fitted Values')
ax.set_ylabel('Residuals')



# Create table of model prediction results
data = {'Attributes': column_names,
        'Coefficient Beta_i': result.params,
        't-Values': result.tvalues,
        'p-Values': result.pvalues
       }
ModelResults = pd.DataFrame(data)
ModelResults.round(6) # Round values in table to 4-decimal places
```

Out[12]:

	Attributes	Coefficient Beta_i	t-Values	p-Values
0	Intercept	41.628587	34.740992	0.0
1	cylinders*horsepower*acceleration	-0.000024	-5.360653	0.0
2	year*weight	-0.000072	-11.293020	0.0

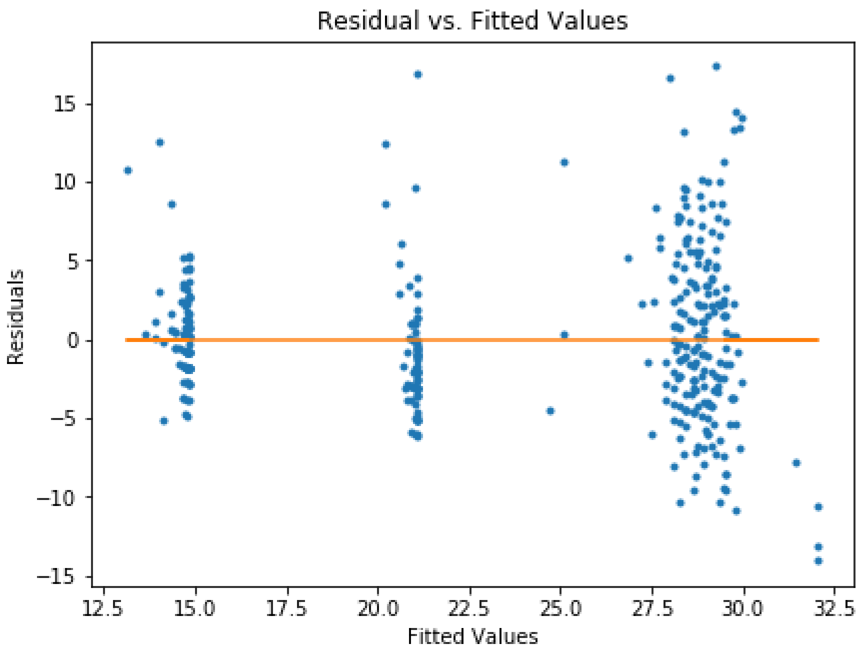


Problem g

In [13]:
```
# Creates dataframe with interactions
data = {
#         'square(cylinders*acceleration)': np.square(df_copy['cylinders']*df_
copy['horsepower']),
#         'log(horsepower*acceleration)': np.log(df_copy['horsepower']*df_copy
['acceleration']),
        'sqrt(cylinders*acceleration)': np.sqrt(df_copy['cylinders']*df_copy[
'acceleration']),
        'log(acceleration)': np.log(df_copy['acceleration']),
#         'horsepower': df_copy['horsepower']*df_copy['horsepower'],
#         'log(year*origin)': np.log(df_copy['year']*df_copy['origin']),
#         'year*weight': df_copy['year']*df_copy['weight'],
       }
df_temp = pd.DataFrame(data)

# Create design matrix
X_var = np.asarray(df_temp)
X_design = sm.add_constant(X_var)

column_names = list(df_temp.columns)
column_names.insert(0, 'Intercept')

# Create the Simple Linear Regression Model and fit it
MLRmodel = sm.OLS(y_true, X_design)
result = MLRmodel.fit()

# Use fitted model to make predictions and residuals
y_pred = result.predict(X_design)
y_pred = np.reshape(y_pred, (-1, 1))
residuals = y_true - y_pred


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

# Create Residuals vs. Fitted Values Plot
ax = f.add_subplot(121)
ax.plot(y_pred, residuals, '.') # Plots the residual points
ax.plot(y_pred, 0*y_pred) # Plots the zero error line
ax.set_title('Residual vs. Fitted Values')
ax.set_xlabel('Fitted Values')
ax.set_ylabel('Residuals')



# Create table of model prediction results
data = {'Attributes': column_names,
        'Coefficient Beta_i': result.params,
        't-Values': result.tvalues,
        'p-Values': result.pvalues
       }
ModelResults = pd.DataFrame(data)
ModelResults.round(6) # Round values in table to 4-decimal places
```

Out[13]:

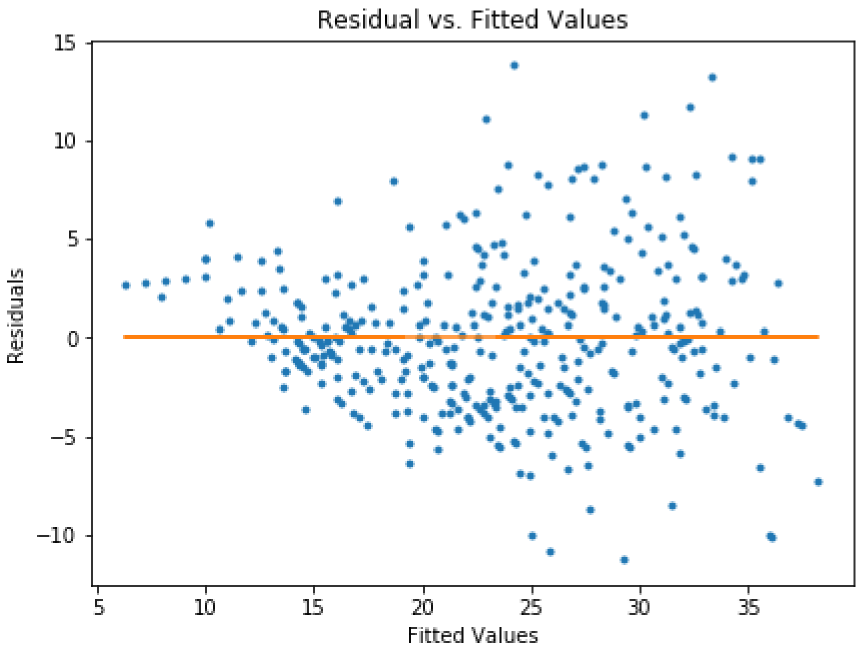
	Attributes	Coefficient Beta_i	t-Values	p-Values
0	Intercept	2.532738	0.629591	0.529332
1	sqrt(cylinders*acceleration)	-4.281003	-20.417817	0.000000
2	log(acceleration)	21.816990	15.862587	0.000000



In [14]:
```
# Creates dataframe with interactions
data = {
#         'square(cylinders*acceleration)': np.square(df_copy['cylinders']*df_
copy['horsepower']),
        'log(horsepower*acceleration)': np.log(df_copy['horsepower']*df_copy[
'acceleration']),
#         'sqrt(cylinders*acceleration)': np.sqrt(df_copy['cylinders']*df_copy
['acceleration']),
        'log(acceleration)': np.log(df_copy['acceleration']),
#         'horsepower': df_copy['horsepower']*df_copy['horsepower'],
        'year*origin': df_copy['year']*df_copy['origin'],
#         'year*weight': df_copy['year']*df_copy['weight'],
       }
df_temp = pd.DataFrame(data)

# Create design matrix
X_var = np.asarray(df_temp)
X_design = sm.add_constant(X_var)

column_names = list(df_temp.columns)
column_names.insert(0, 'Intercept')

# Create the Simple Linear Regression Model and fit it
MLRmodel = sm.OLS(y_true, X_design)
result = MLRmodel.fit()

# Use fitted model to make predictions and residuals
y_pred = result.predict(X_design)
y_pred = np.reshape(y_pred, (-1, 1))
residuals = y_true - y_pred


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

# Create Residuals vs. Fitted Values Plot
ax = f.add_subplot(121)
ax.plot(y_pred, residuals, '.') # Plots the residual points
ax.plot(y_pred, 0*y_pred) # Plots the zero error line
ax.set_title('Residual vs. Fitted Values')
ax.set_xlabel('Fitted Values')
ax.set_ylabel('Residuals')



# Create table of model prediction results
data = {'Attributes': column_names,
        'Coefficient Beta_i': result.params,
        't-Values': result.tvalues,
        'p-Values': result.pvalues
       }
ModelResults = pd.DataFrame(data)
ModelResults.round(6) # Round values in table to 4-decimal places
```

Out[14]:

	Attributes	Coefficient Beta_i	t-Values	p-Values
0	Intercept	143.052156	17.387799	0.0
1	log(horsepower*acceleration)	-20.240600	-21.388294	0.0
2	log(acceleration)	9.241881	8.061905	0.0
3	year*origin	0.027054	7.317776	0.0



In [ ]: