In [18]:

```
# Import the libraries
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import seaborn as sns
```

In [19]:

```
# Import the CSV Data
dataset = pd.read_csv("//home//yeshua//Documents//study//excel//Auto.csv")
```

In [20]:

```
dataset.head()
```

Out[20]:

	symboling	make	fuel-type	aspiration	num-of-doors	body-style	drive-wheels	engine-location	wheel-base	length	...	engine-size	fuel-system	bore	stroke
0	3	alfa-romero	gas	std	two	convertible	rwd	front	88.6	168.8	...	130	mpfi	3.47	2.68
1	3	alfa-romero	gas	std	two	convertible	rwd	front	88.6	168.8	...	130	mpfi	3.47	2.68
2	1	alfa-romero	gas	std	two	hatchback	rwd	front	94.5	171.2	...	152	mpfi	2.68	3.47
3	2	audi	gas	std	four	sedan	fwd	front	99.8	176.6	...	109	mpfi	3.19	3.4
4	2	audi	gas	std	four	sedan	4wd	front	99.4	176.6	...	136	mpfi	3.19	3.4

5 rows × 25 columns

In [21]:

```
print(dataset.keys())
```

```
Index(['symboling', 'make', 'fuel-type', 'aspiration', 'num-of-doors',
       'body-style', 'drive-wheels', 'engine-location', 'wheel-base', 'length',
       'width', 'height', 'curb-weight', 'engine-type', 'num-of-cylinders',
       'engine-size', 'fuel-system', 'bore', 'stroke', 'compression-ratio',
       'horsepower', 'peak-rpm', 'city-mpg', 'highway-mpg', 'price'],
      dtype='object')
```

In [22]:

```
dataset.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 205 entries, 0 to 204
Data columns (total 25 columns):
symboling            205 non-null int64
make                 205 non-null object
fuel-type            205 non-null object
aspiration           205 non-null object
num-of-doors         205 non-null object
body-style           205 non-null object
drive-wheels         205 non-null object
engine-location      205 non-null object
wheel-base           205 non-null float64
length               205 non-null float64
width                205 non-null float64
height               205 non-null float64
```

```
curb-weight          205 non-null int64
engine-type          205 non-null object
num-of-cylinders     205 non-null object
engine-size          205 non-null int64
fuel-system          205 non-null object
bore                 205 non-null object
stroke               205 non-null object
compression-ratio    205 non-null float64
horsepower           205 non-null object
peak-rpm             205 non-null object
city-mpg             205 non-null int64
highway-mpg          205 non-null int64
price                205 non-null object
dtypes: float64(5), int64(5), object(15)
memory usage: 40.1+ KB
```

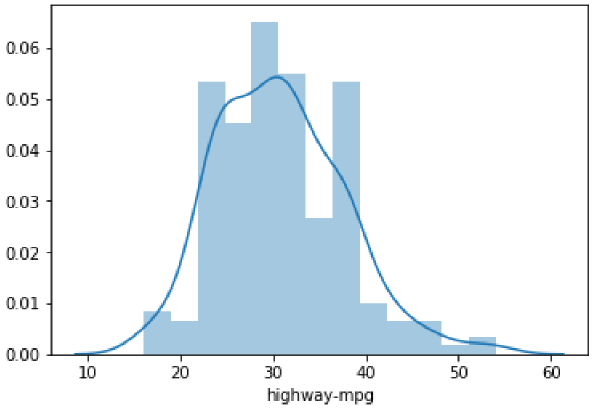
In [28]:

```
sns.distplot(dataset["highway-mpg"])
```

```
/home/yeshua/anaconda3/lib/python3.6/site-packages/matplotlib/axes/_axes.py:6462: UserWarning: The
'normed' kwarg is deprecated, and has been replaced by the 'density' kwarg.
  warnings.warn("The 'normed' kwarg is deprecated, and has been "
```

Out[28]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f460535b9b0>
```



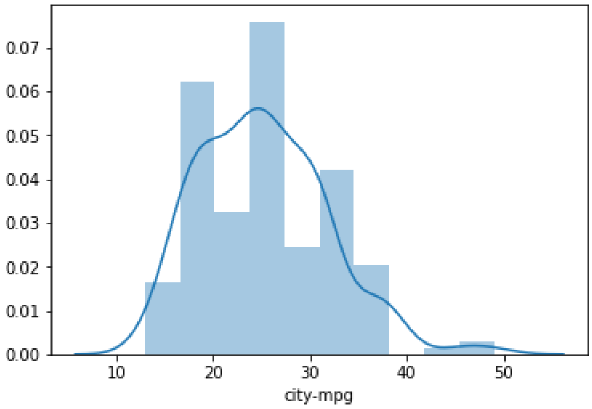
In [30]:

```
sns.distplot(dataset['city-mpg'])
```

```
/home/yeshua/anaconda3/lib/python3.6/site-packages/matplotlib/axes/_axes.py:6462: UserWarning: The
'normed' kwarg is deprecated, and has been replaced by the 'density' kwarg.
  warnings.warn("The 'normed' kwarg is deprecated, and has been "
```

Out[30]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f46052bb358>
```

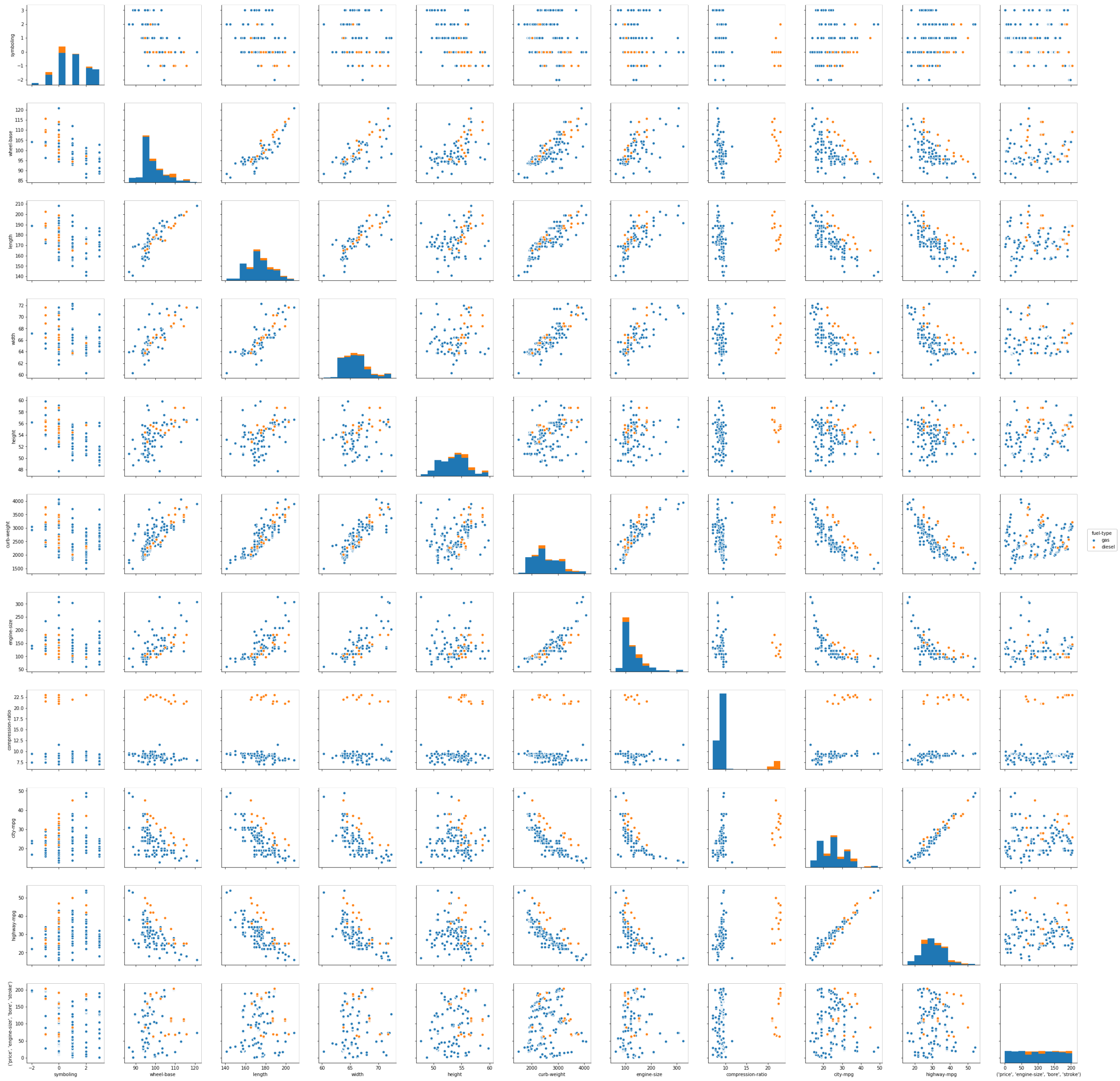


In [64]:

```
sns.pairplot(dataset.drop("price", axis=1), hue="fuel-type", size=3)
```

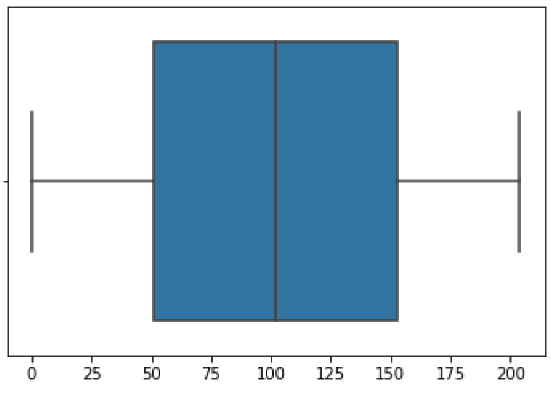
Out[64]:

```
<seaborn.axisgrid.PairGrid at 0x7fe568b6ef28>
```



In [65]:

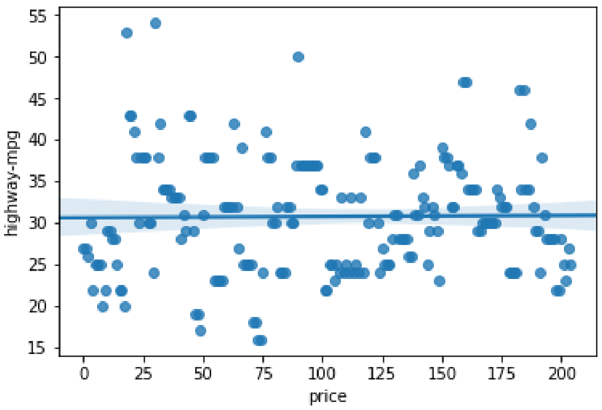
```
sns.boxplot(x=dataset["price"])
plt.show()
```



price

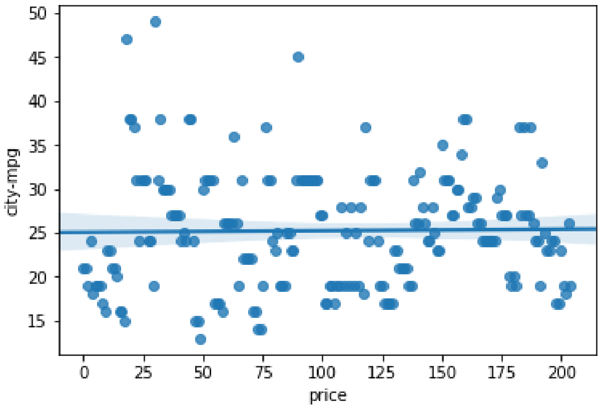
In [67]:

```
sns.regplot(x=dataset["price"],y=dataset["highway-mpg"]),
plt.show()
```



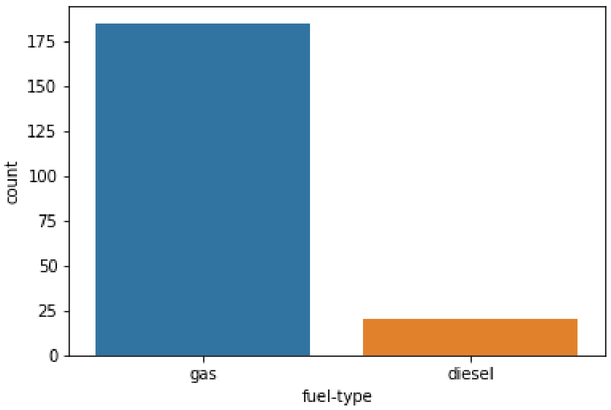
In [68]:

```
sns.regplot(x=dataset["price"],y=dataset["city-mpg"]),
plt.show()
```



In [69]:

```
sns.countplot(x=dataset["fuel-type"])
plt.show()
```



In [29]:

```
y= np.array(dataset.iloc[:,24].values).reshape(-1,1)
X= np.array(dataset.iloc[:,23].values).reshape(-1,1)
```

In [30]:

```
# Split the dataset into Training set and Test set
from sklearn.model_selection import train_test_split
xtrain, xtest, ytrain, ytest = train_test_split(X, y, test_size = 0.2)
```

In [31]:

```
from sklearn.preprocessing import PolynomialFeatures
poly_reg = PolynomialFeatures(degree = 12)
X_poly = poly_reg.fit_transform(X)
poly_reg.fit(X_poly, y)
lin_reg_2 = LinearRegression()
lin_reg_2.fit(X_poly, y)
```

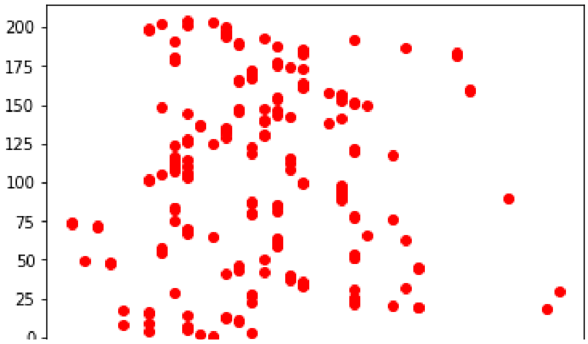
Out[31]:

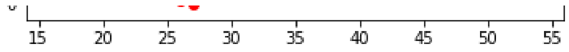
```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=1, normalize=False)
```

In [34]:

```
plt.scatter(X, y, color = 'red')
plt.plot(X, lin_reg.predict(X), color = 'blue')
plt.title('Truth or Bluff (Linear Regression)')
plt.xlabel('city')
plt.ylabel('highway')
plt.show()
```

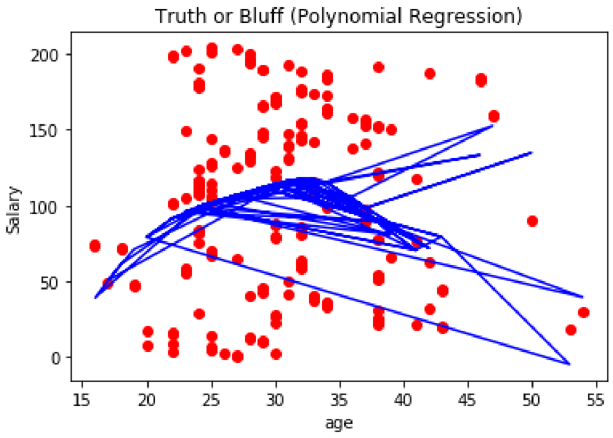
```
---------------------------------------------------------------------------
NotFittedError                            Traceback (most recent call last)
<ipython-input-34-def0195d8a08> in <module>()
      1 plt.scatter(X, y, color = 'red')
----> 2 plt.plot(X, lin_reg.predict(X), color = 'blue')
      3 plt.title('Truth or Bluff (Linear Regression)')
      4 plt.xlabel('city')
      5 plt.ylabel('highway')

~/anaconda3/lib/python3.6/site-packages/sklearn/linear_model/base.py in predict(self, X)
    254             Returns predicted values.
    255         """
--> 256         return self._decision_function(X)
    257 
    258     _preprocess_data = staticmethod(_preprocess_data)

~/anaconda3/lib/python3.6/site-packages/sklearn/linear_model/base.py in _decision_function(self, X)
    235 
    236     def _decision_function(self, X):
--> 237         check_is_fitted(self, "coef_")
    238 
    239         X = check_array(X, accept_sparse=['csr', 'csc', 'coo'])

~/anaconda3/lib/python3.6/site-packages/sklearn/utils/validation.py in check_is_fitted(estimator, attributes, msg, all_or_any)
    766 
    767     if not all_or_any([hasattr(estimator, attr) for attr in attributes]):
--> 768         raise NotFittedError(msg % {'name': type(estimator).__name__})
    769 
    770 

NotFittedError: This LinearRegression instance is not fitted yet. Call 'fit' with appropriate arguments before using this method.
```





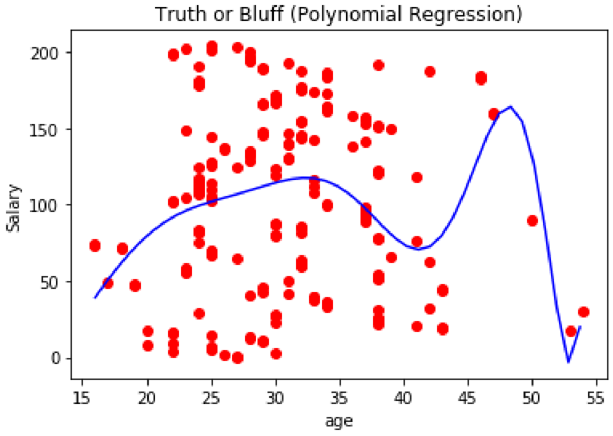
In [35]:

```
plt.scatter(X, y, color = 'red')
plt.plot(X, lin_reg_2.predict(poly_reg.fit_transform(X)), color = 'blue')
plt.title('Truth or Bluff (Polynomial Regression)')
plt.xlabel('age')
plt.ylabel('Salary')
plt.show()
```



In [36]:

```
X_grid = np.arange(min(X), max(X), 0.9)
X_grid = X_grid.reshape((len(X_grid), 1))
plt.scatter(X, y, color = 'red')
plt.plot(X_grid, lin_reg_2.predict(poly_reg.fit_transform(X_grid)), color = 'blue')
plt.title('Truth or Bluff (Polynomial Regression)')
plt.xlabel('age')
plt.ylabel('Salary')
plt.show()
```



In [39]:

```
cofficient=lin_reg_2.coef_
cofficient
```

Out[39]:

```
array([[ 0.00000000e+00,  5.26581943e-07,  5.19798175e-09,
         9.38417835e-08,  1.35632529e-06,  1.40434164e-05,
         8.07452076e-05, -1.20758367e-05,  7.50916313e-07,
        -2.48719898e-08,  4.61490077e-10, -4.53769137e-12,
         1.84447843e-14]])
```

In [40]:

```
intercept=lin_reg_2.intercept_
intercept
```

Out[40]:

```
array([-37.50003633])
```

In [41]:

```
import statsmodels.formula.api as smf
results = smf.OLS(y,X).fit()
results.summary()
```

Out[41]:

OLS Regression Results

Dep. Variable:	y	**R-squared:**	0.715
Model:	OLS	**Adj. R-squared:**	0.713
Method:	Least Squares	**F-statistic:**	511.0
Date:	Mon, 29 Oct 2018	**Prob (F-statistic):**	1.83e-57
Time:	20:56:31	**Log-Likelihood:**	-1140.2
No. Observations:	205	**AIC:**	2282.
Df Residuals:	204	**BIC:**	2286.
Df Model:	1		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
x1	3.1639	0.140	22.605	0.000	2.888	3.440

Omnibus:	16.988	**Durbin-Watson:**	0.114
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	6.175
Skew:	-0.075	**Prob(JB):**	0.0456
Kurtosis:	2.163	**Cond. No.**	1.00

Warnings:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.