<table>
<thead>
<tr>
<th>Obs</th>
<th>X1</th>
<th>X2</th>
<th>X3</th>
<th>Y</th>
<th>Dist from (0,0,0)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>3</td>
<td>0</td>
<td>R</td>
<td>( \sqrt{9} )</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>R</td>
<td>( \sqrt{4} )</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>1</td>
<td>3</td>
<td>R</td>
<td>( \sqrt{1+3^2} = \sqrt{10} )</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>G</td>
<td>( \sqrt{1+2^2} = \sqrt{5} )</td>
</tr>
<tr>
<td>5</td>
<td>-1</td>
<td>0</td>
<td>1</td>
<td>G</td>
<td>( \sqrt{1+1} = \sqrt{2} )</td>
</tr>
<tr>
<td>6</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>R</td>
<td>( \sqrt{1+1+1} = \sqrt{3} )</td>
</tr>
</tbody>
</table>

(b) \( K=1 \Rightarrow (0,0,0) \) is Green.
This is because the single closest neighbor is Obs 5 at \( \sqrt{2} \) distance away.

(c) \( K=3 \Rightarrow (0,0,0) \) is Red.
This is because the 3 closest neighbors are 5, 6, and 2, which are \( \{A, R, R\} \),
so final prediction is Red.

(d) We expect the best value of \( K \) to be smaller.
This is because, while both small and large values of \( K \) will work well at data points far from the Bayes Decision Boundary, data points close to the boundary will be overgeneralized by large values of \( K \).
3.7

(a) Ans. is III. (Males make more than Females if GPA high enough)

Reason:
The equation we get from the model is:

\[ Y = 50 + 20(GPA) + 0.07(IQ) + 35(Cender) + 0.01(GPA \cdot IQ) - 10(GPA \cdot Cender) \]

For males, Cender = 0, this eqn. becomes:

\[ Y_{\text{male}} = 50 + 20(GPA) + 0.07(IQ) + 0.01(GPA) \]

For females, Cender = 1, so the equation becomes:

\[ Y_{\text{female}} = 50 + 20(GPA) + 0.07(IQ) + 35 + 0.01(GPA \cdot IQ) - 10(GPA) \]

\[ = 85 + 10(GPA) + 0.07(IQ) + 0.01(GPA \cdot IQ) \]

As we can see, eliminating all other
We can see from the equations that
\[ Y_{\text{male}} > Y_{\text{female}} \text{ if GPA is high enough.} \]
(b)  IQ = 110, GPA = 4, Gender = 1
\[ y = 50 + 20 \cdot 4 + 0.07 \cdot 110 + 35 + 0.01(4 \times 110) - 10(4) \]
\[ = 85 + 40 + 7.7 + 4.4 = 137.1 \]

(c) **False**.
Evidence of an interaction effect does not come from the coefficient, but rather from testing the model with data.

4.
(a) The cubic regression will have lower RSS.
This is because higher order polynomials will in general fit data better.

(b) The linear regression will have lower RSS.
This is because the cubic regression is overfitted on the test data.

(c) Cubic regression has lower RSS, same reason as (a).

(d) Not enough information to tell. Explain.
\begin{align*}
\delta_{ij} &= x_i \hat{\beta} \\
&= \sum_{i=1}^{n} x_i y_i \\
&= \sum_{i=1}^{n} \left( x_{i1} y_i + x_{i2} y_i + \ldots + x_{in} y_i \right) \\
&= \sum_{i=1}^{n} \left( \frac{x_i}{\sum_{i=1}^{n} x_i^2} \right) x_i y_i \\
\therefore \quad a_i &= \frac{x_i}{\sum_{i=1}^{n} x_i^2} \\
&\quad \checkmark + 10
\end{align*}
Note that training a simple linear regression model yields the equation

\[ \hat{y} = \hat{\beta}_0 + \hat{\beta}_1 x, \quad x \text{ is meant to be test predictor} \]

(3.4) Tells us that \( \hat{\beta}_0 = \bar{y} - \hat{\beta}_1 \bar{x} \).

i.e. \( \hat{y} = \bar{y} - \hat{\beta}_1 \bar{x} + \hat{\beta}_1 x \).

Therefore, if we let \( x = \bar{x} \),

\[ \hat{y} = \bar{y} - \hat{\beta}_1 \bar{x} + \hat{\beta}_1 \bar{x} = \bar{y}, \]

\( \Rightarrow (\bar{x}, \bar{y}) \) lies on the line \( \hat{y} = \hat{\beta}_0 + \hat{\beta}_1 x \).
LINEAR ALGEBRA EXERCISE:

Let \( A = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} \) \( n \) rows.

\[
\therefore A^T A = \begin{bmatrix} 1 & 1 & \ldots & 1 \\ x_1 & x_2 & \ldots & x_n \end{bmatrix} \begin{bmatrix} 1 \\ x_1 \\ \vdots \\ x_n \end{bmatrix} = \begin{bmatrix} n \sum x_i \\ \sum x_i \sum x_i^2 \end{bmatrix}
\]

\[
\therefore (A^T A)^{-1} = \frac{1}{n \sum x_i^2 - (\sum x_i)^2} \begin{bmatrix} \sum x_i^2 & -\sum x_i \\ -\sum x_i & n \end{bmatrix}
\]

\[
A^T y = \begin{bmatrix} 1 & 1 & \ldots & 1 \\ x_1 & x_2 & \ldots & x_n \end{bmatrix} \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix} = \begin{bmatrix} \sum y_i \\ \sum x_i y_i \end{bmatrix}
\]
Plugging into $\beta = (A^T A)^{-1} A^T y$

$$(A^T A)^{-1} A^T y$$

$$= \frac{1}{n \cdot S_{Sx}} \begin{bmatrix} \sum x_i^2 - \overline{x}^2 \\ -\overline{x} \\ n \end{bmatrix} \begin{bmatrix} \sum y_i \\ \sum x_i y_i \end{bmatrix}$$

$$= \frac{1}{n \cdot S_{Sx}} \begin{bmatrix} (\sum x_i^2)(\sum y_i) - (\sum x_i)(\sum x_i y_i) \\ - (\sum x_i)(\sum y_i) + n \sum x_i y_i \end{bmatrix}$$

$$= \frac{1}{S_{Sx}} \begin{bmatrix} \overline{y}(\sum x_i^2) - \overline{x}(\sum x_i y_i) \\ \sum x_i y_i - n \overline{x} \overline{y} \end{bmatrix}$$

$$= \frac{1}{S_{Sx}} \begin{bmatrix} \overline{y} (\sum x_i^2) - \overline{y} (n \overline{x}^2) + \overline{x} (n \overline{y}) - \overline{x} (n \sum x_i y_i) \end{bmatrix}$$

$$= \frac{1}{S_{Sx}} \left[ \begin{array}{c} \sum x_i y_i - n \overline{x} \overline{y} \end{array} \right] = \left[ \begin{array}{c} \frac{SP_{xy}}{S_{Sx}} \end{array} \right]$$

$$= \frac{1}{S_{Sx}} \left[ \begin{array}{c} \sum x_i y_i - n \overline{x} \overline{y} \end{array} \right] = \left[ \begin{array}{c} \frac{SP_{xy}}{S_{Sx}} \end{array} \right] = \left[ \begin{array}{c} \beta_1 \end{array} \right]$$

Note:

$SS_x = \sum x_i^2 - n \overline{x}^2 = \sum (x_i - \overline{x})^2$

$SP_{xy} = \sum x_i y_i - n \overline{x} \overline{y} = \sum (x_i - \overline{x})(y_i - \overline{y}) + 10$
In [16]:  # (1.1) Read Auto data into Python.

import pandas as pd
import numpy as np
from sklearn.linear_model import LinearRegression
import matplotlib.pyplot as plt
%matplotlib inline

dataset = pd.read_csv("Auto.csv", index_col =[-1])

In [17]:  # (1.2) Simple Linear Regression.

regr = LinearRegression()
X = dataset["horsepower"].values.reshape(-1,1)
Y = dataset["mpg"].values.reshape(-1,1)

_a = regr.fit(X, Y)

In [18]:  # (1.3) Plot the response and the predictor, display line.

b0 = regr.intercept_[0]
b1 = regr.coef_[0][0]
print(‘y = ‘, b0, ‘ + ‘, b1, ‘ * x‘)

x = np.linspace(40, 200)
y = (b1*x) + b0
plt.scatter(X, Y, color=’gray’)
plt.plot(x, y, ‘-r’)
plt.show()

y = 39.93586102117047 + -0.15784473335365365 * x

In [19]:  # (1.4) RSS

print("Residual sum of squares: %.2f" % ((regr.predict(X) - Y)**2).sum())

Residual sum of squares: 9385.92
The line does not fit the data to well (perhaps a polynomial/quadratic model would be better).

```
In [32]: 
   # (2.1) Color Plot
   matrix = np.array(dataset.head(15).values)
   # showing only first 15 rows because otherwise it's too big.

   plt.imshow(matrix, cmap='viridis')
   plt.show()

*Note: due to campus-wide connectivity issues, I couldn't connect to CTY printer, so I had to print this document in B/W.*
```

```
In [21]: 
   # (2.2) Correlation matrix

   correlation_matrix = dataset.corr()
   print(correlation_matrix)

   mpg  cylinders  displacement  horsepower  weight
   acceleration  year  origin
   mpg   1.000000  -0.777618  -0.805127  -0.778427  -0.832244
   cylinders  -0.777618   1.000000   0.950823   0.842983   0.897527
   displacement  -0.805127   0.950823   1.000000   0.897257   0.932994
   horsepower  -0.778427   0.842983   0.897257   1.000000   0.864538
   weight     -0.832244   0.897527   0.932994   0.864538   1.000000
   acceleration  0.423329  -0.504683  -0.543800  -0.689196  -0.416839
   year       0.580541  -0.345647  -0.369855  -0.416361  -0.309120
   origin    0.565209  -0.568932  -0.614535  -0.455171  -0.505005
   acceleration  year  origin
   mpg   0.423329  0.580541  0.565209
   cylinders -0.504683 -0.345647 -0.568932
   displacement -0.543800 -0.369855 -0.614535
   horsepower   -0.689196 -0.416361 -0.455171
   weight       -0.416839 -0.309120  -0.585005
   acceleration 1.000000  0.290316  0.212746
   year        0.290316  1.000000  0.181528
   origin       0.212746  0.181528  1.000000
```
In [22]:

```python
# (2.3) Multiple Linear Regression

X = dataset[['cylinders', 'displacement', 'horsepower', 'weight', 'acceleration', 'year', 'origin']][
Y = dataset['mpg'].values.reshape(-1, 1)

regr = LinearRegression()
regr.fit(X, Y)

b0 = regr.intercept_[0]
bl = regr.coef_[0][0]
print('y = ', b0, ' + ', bl, ' * x')

print('Residual sum of squares: %.2f'
% (regr.predict(X) - Y) ** 2).sum())

y = -17.218434622017536 + -0.4933763188584681 * x
Residual sum of squares: 4252.21
```

(2.4) We can see by the RSS that the multiple linear regression is significantly lower. This implies that the multiple linear regression performed better.

In [23]:

```python
# (3) 3.7-13

#(a) Creating a feature vector.
X = np.random.normal(0, 1, 100)

# (b) Creating a random noise vector
EPS = np.random.normal(0, 0.25, 100)

# (c) Creating vector Y = -1 + 0.5X + eps
Y = [(-1 + (0.5*x)) for x in X]
Y = Y + EPS

print("size of Y is", Y.shape[0])

size of Y is 100
```

(c) The size of the Y vector is 100, since it was created from the X and eps vectors.

\[ \beta_0 = -1, \beta_1 = 0.5 \]
In [24]: # (d) Scatter plot of the generated data.
plt.scatter(x, y, color='gray')
plt.show()

(d) We can definitely see the linear relationship between $x$ and $y$, and that the slope of the data seems to be positive in general. We can also clearly see the effect of the added noise ($\text{np.random.normal}$) since the data definitely does not fit on an exact line.

In [25]: # (e) Least squares linear model.

    regr = LinearRegression()
    X = X.reshape(-1,1)
    Y = Y.reshape(-1,1)
    _a = regr.fit(X, Y)

    b0 = regr.intercept_[0]
    b1 = regr.coef_[0]

    print('y = ', b0, ' + ', b1, ' * x')
    print("Residual sum of squares: %.2f" % ((regr.predict(X) - Y) ** 2).sum())

    y = -1.0450360240231463 + [0.49493035] * x
    Residual sum of squares: 5.36

(e) We can clearly see that the coefficients in the learned model closely approximate the same coefficients in the model used to generate the training data.
In [26]:

```python
# (f) Plotting least squares model vs. Population regression.
plt.scatter(x, y, color='gray')

x = np.linspace(-2.5, 2)
y = (bl*x) + b0
y_true = (0.5*x) - 1

plt.plot(x, y, '-x', label = "least squares line")
plt.plot(x, y_true, '-b', label = "population regression line")
plt.legend(loc='upper left')
plt.show()
```

In [27]:

```python
# (g) Fitting a Polynomial Regression Model

from sklearn.preprocessing import PolynomialFeatures

poly = PolynomialFeatures(degree = 2)
X_ = poly.fit_transform(X)

regr = LinearRegression()
regr.fit(X_, Y)

print(regr.coef_)

print("Residual sum of squares: %2f" % ((regr.predict(X_) - Y) ** 2).sum())
```

```
[[ 0.  
  0.49564517 -0.00360006]]
Residual sum of squares: 5.36
```

We can see that using a polynomial model, while in theory should have fit the training data better, did not actually improve the RSS better.

I expect this might be either because, since the underlying model is actually linear, modelling it as a quadratic does not help. It might also be the case that I am using the libraries wrong though.
In [28]: # (h) Repeat with LESS noise

# Creating a feature vector.
X = np.random.normal(0, 1, 100)

# Creating a random noise vector
EPS = np.random.normal(0, 0.1, 100)

# (c) Creating vector \( Y = -1 + 0.5X + \epsilon \)
Y = [(-1 + (0.5*x)) for x in X]
Y = Y + EPS

# (e) least squares linear model.
regr = LinearRegression()
X = X.reshape(-1,1)
Y = Y.reshape(-1,1)
_a = regr.fit(X, Y)

b0 = regr.intercept_[0]
b1 = regr.coef_[0]

print('y = ', b0, ' + ', b1, ' * x')

print("Residual sum of squares: %.2f"
% ((regr.predict(X) - Y) ** 2).sum())

# (f) Plotting least squares model vs. Population regression.
plt.scatter(X, Y, color='gray')
x = np.linspace(-2.5, 2)
y = (b1*x) + b0
y_true = (0.5*x) - 1

plt.plot(x, y, 'r-', label = "least squares line")
plt.plot(x, y_true, 'b-', label = "population regression line")
plt.legend(loc='upper left')

plt.show()
\[ y = -1.01667298374288 + [0.49043578] \times x \]

Residual sum of squares: 0.85

(h) We can see that with less noise, our Linear Regression VERY closely approximates the population regression.
In [29]:

# (i) Repeat with MORE noise

# Creating a feature vector.
X = np.random.normal(0, 1, 100)

# Creating a random noise vector
EPS = np.random.normal(0, 0.5, 100)

# (c) Creating vector Y = -1 + 0.5X + eps
Y = [(-1 + (0.5*x)) for x in X]
Y = Y + EPS

# (e) least squares linear model.
regr = LinearRegression()
X = X.reshape(-1, 1)
Y = Y.reshape(-1, 1)
_a = regr.fit(X, Y)

b0 = regr.intercept_[0]
b1 = regr.coef_[0]

print('y = ', b0, ' + ', b1, ' * x')

print("Residual sum of squares: %.2f"
    % ((regr.predict(X) - Y) ** 2).sum())

# (f) Plotting least squares model vs. Population regression.
plt.scatter(X, Y, color='gray')

x = np.linspace(-2.5, 2)
y = (b1*x) + b0
y_true = (0.5*x) - 1

plt.plot(x, y, '-r', label = "least squares line")
plt.plot(x, y_true, '-b', label = "population regression line")
plt.legend(loc='upper left')

plt.show()}
\[ y = -0.8919423919140468 + 0.5228962 \times x \]

Residual sum of squares: 25.46

We can see that with more noise, our linear regression line does not approximate the line as well, which is to be expected. However, our model still does a good job of capturing the overall trend of the data.

\[ +0 \times 2 \]