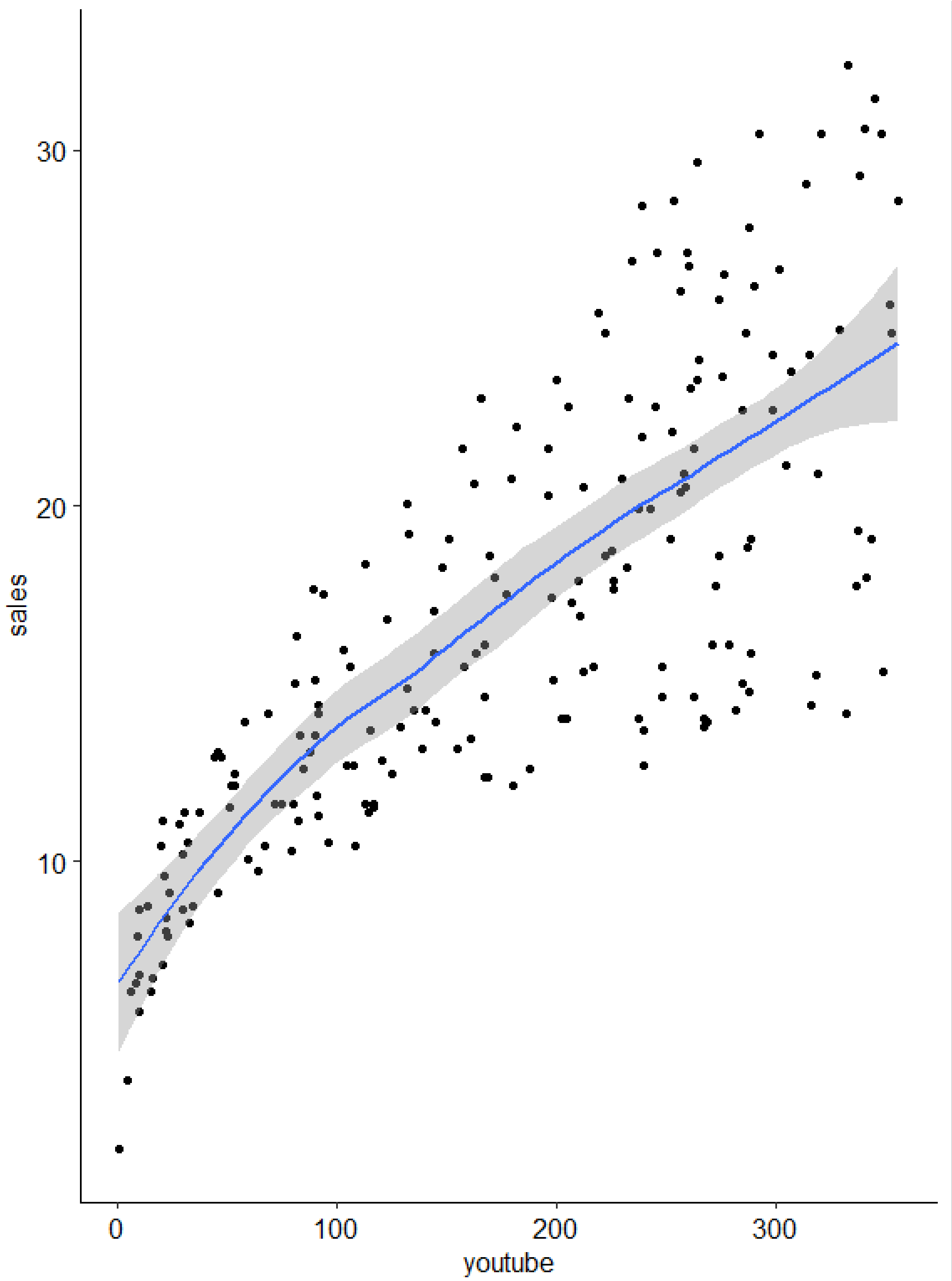
ASSIGNMENT 3 - LINEAR REGRESSION & CLASSIFICATION
INF2190 - Winter 2022
[Che Zhu] [Group Member Names / Names of anyone you discussed the assignment with]

Note this assignment is to be done individually. Make a copy of this doc and use it as a template for the assignment. Share your google doc with me: tegan.maharaj@utoronto.ca. Don't share it with anyone else. I may ask you for your source R code at any time during the semester. Submit a PDF of your doc via Quercus.

PART I: LINEAR REGRESSION
Extra background + simple example: https://onlinestatbook.com/2/regression/intro.html http://www.sthda.com/english/articles/40-regression-analysis/167-simple-linear-regression-in-r/

1. Paste a screenshot of your R code and output (plots etc.) for replicating the tutorial (for youtube data). Before each command include a comment describing what the line of code does.

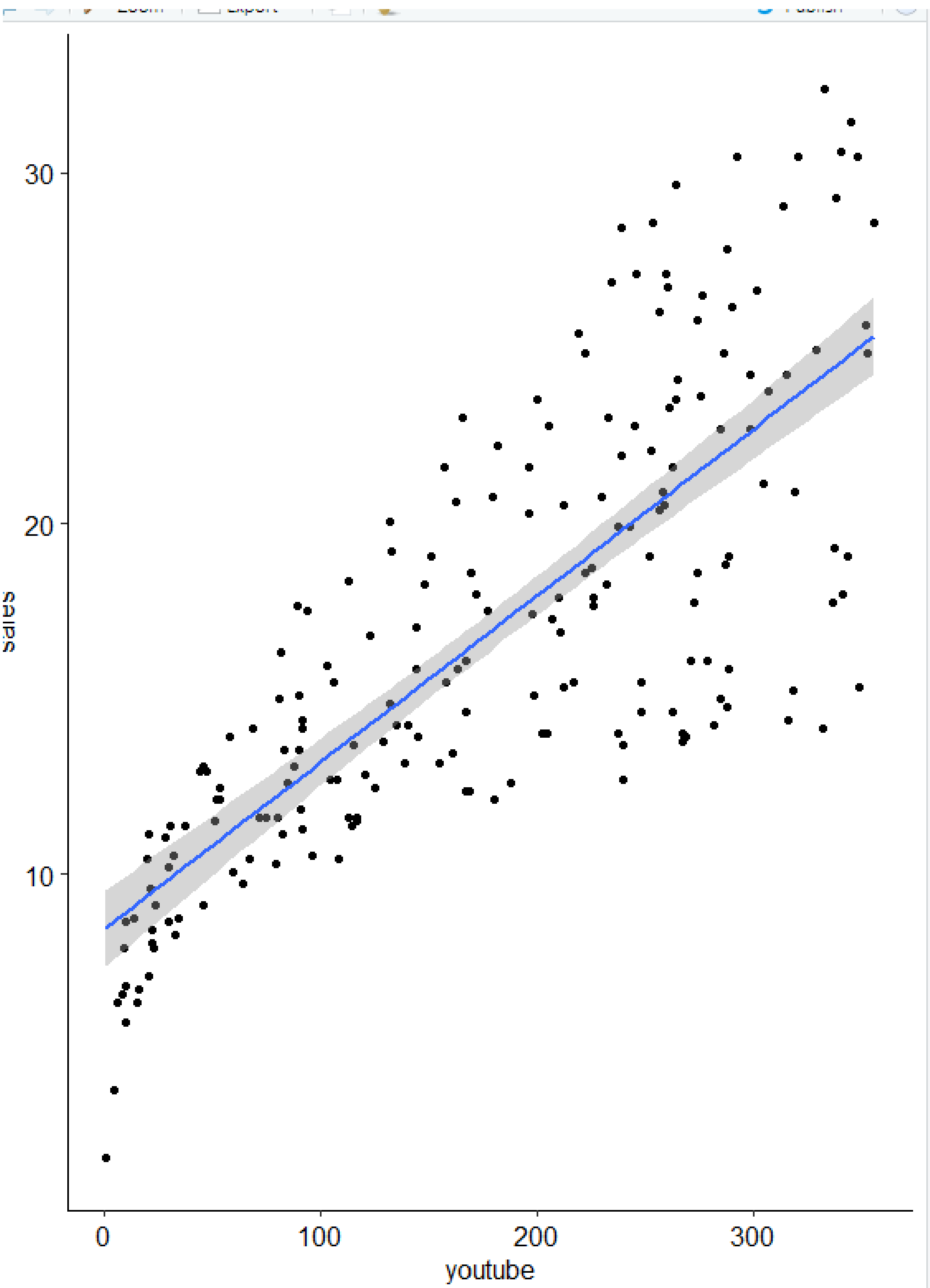
```
library(tidyverse)
library(ggpubr)
# set theme to a publication ready theme
theme_set(theme_pubr())


# Load the package
data("marketing", package = "datarium")
# show the first 4 lines of the data "marketing
head(marketing, 4)

  youtube facebook newspaper sales
1  276.12    45.36     83.04 26.52
2   53.40    47.16     54.12 12.48
3   20.64    55.08     83.16 11.16
4  181.80    49.56     70.20 22.20

#Using ggplot function to plot a youtube verse sales graph with
#a fit line and confident interval.
ggplot(marketing, aes(x = youtube, y = sales)) +
  geom_point() +
  stat_smooth()
```



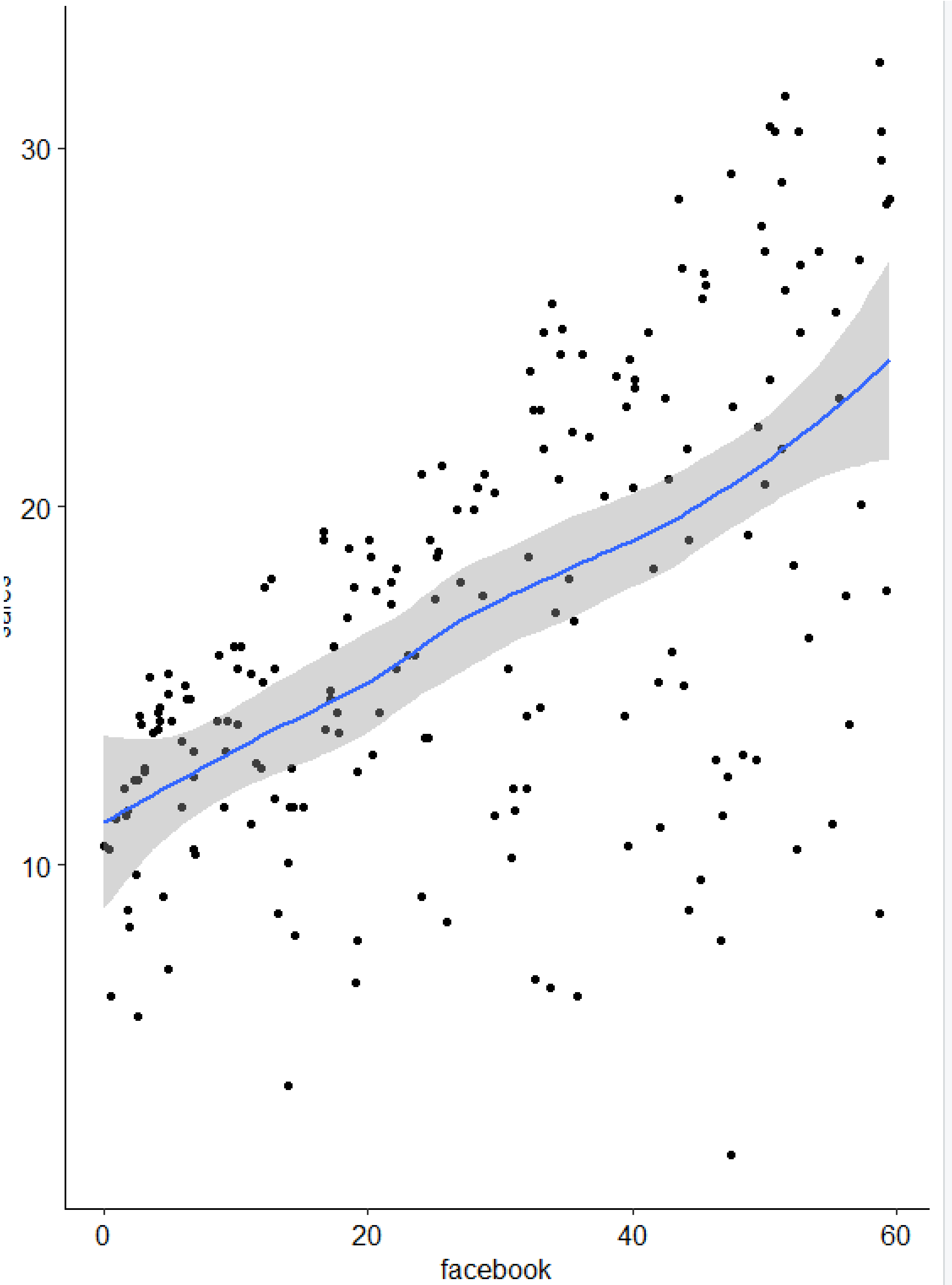

```
> #Calculate the correlation between sales and youtube
> cor(marketing$sales, marketing$youtube)
[1] 0.7822244

#Calculate the intercept and the betaa coeficient for the youtube variable
model <- lm(sales ~ youtube, data = marketing)
model
```

```
Call:
lm(formula = sales ~ youtube, data = marketing)

Coefficients:
(Intercept)      youtube
    8.43911      0.04754
```

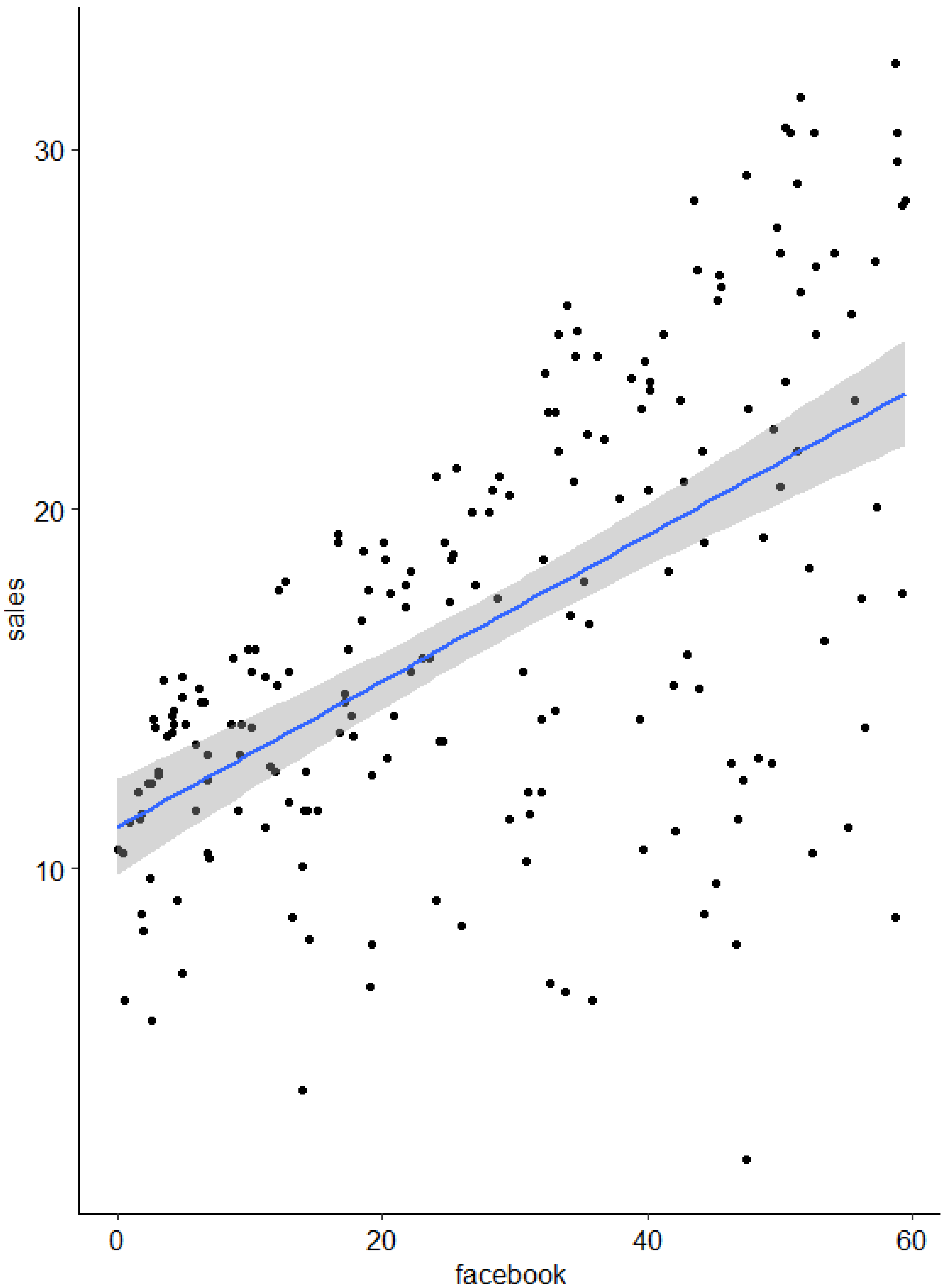
```
#sketch the regression line
ggplot(marketing, aes(youtube, sales)) +
  geom_point() +
  stat_smooth(method = lm)
```



2. Do the same analysis for facebook; this time you only need to paste your plots and summary output

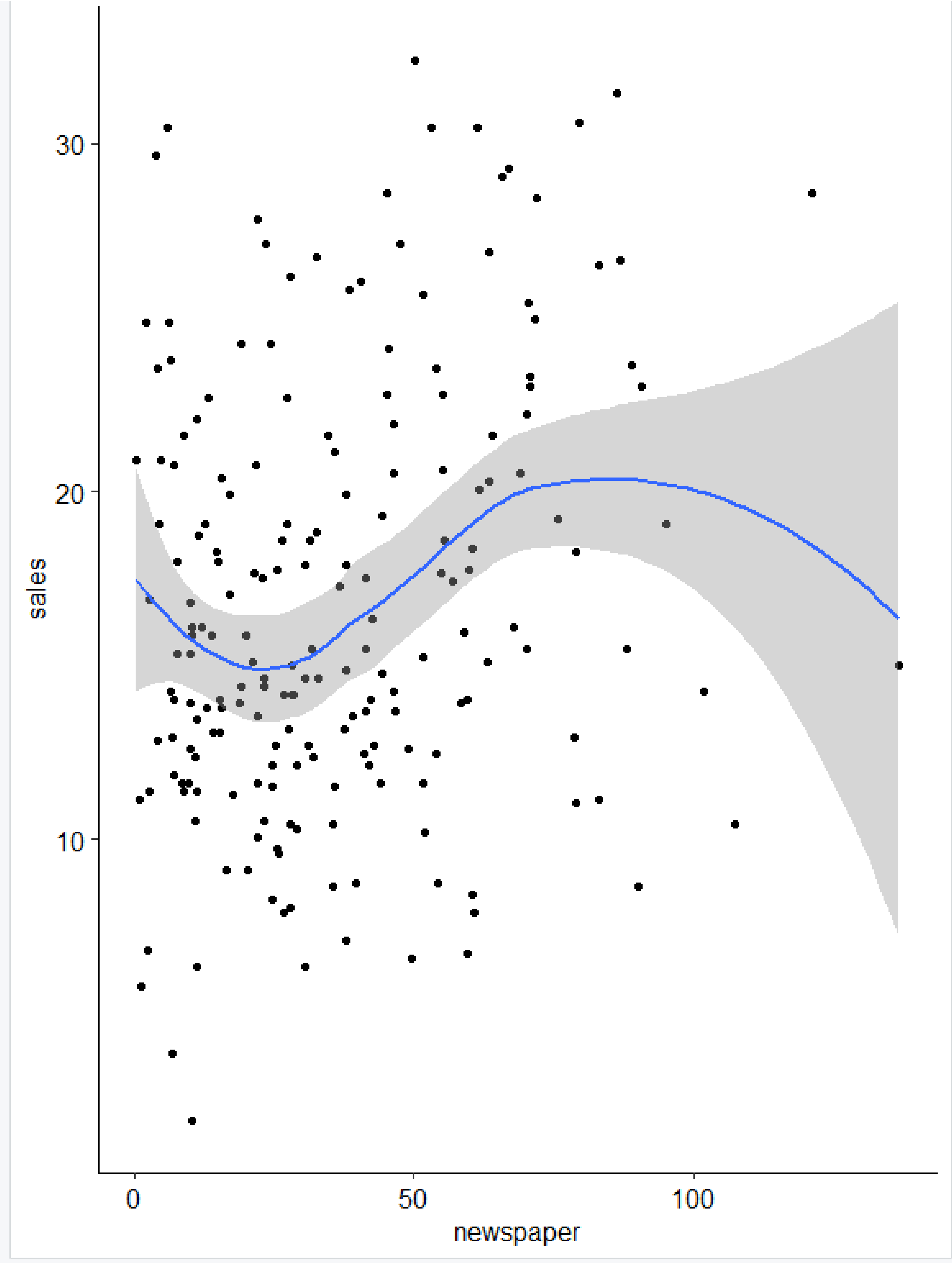


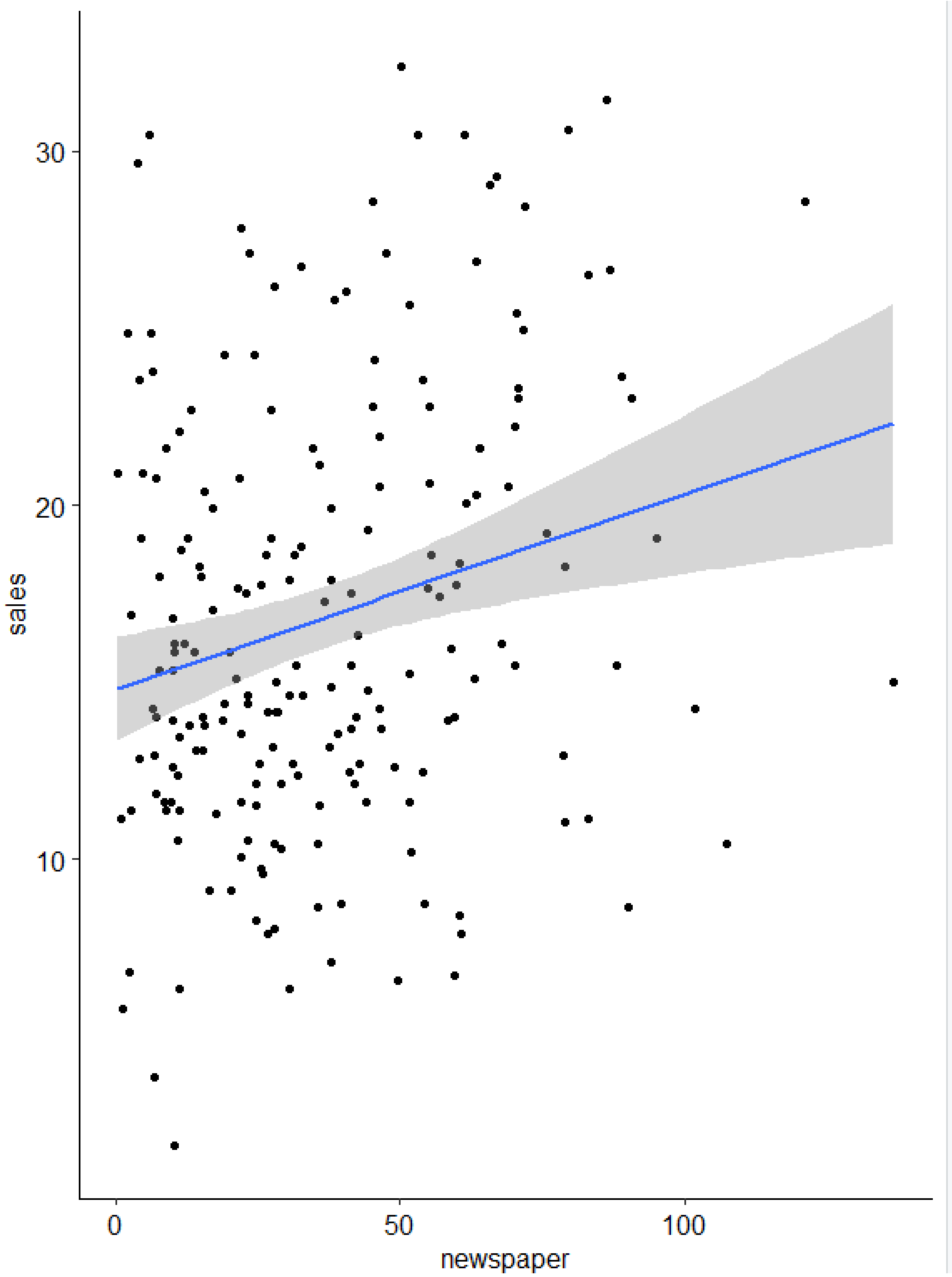
30
20
10
0
20
40
60
facebook



As a result, the fit line has an intercept of 11.1740 and a slope of 0.2025. The coefficient between data is 0.5762226

3. Do the same analysis for newspaper; this time you only need to paste your plots and summary output


30
20
10
sales
0
50
100
newspaper



As a result, the fit line for newspapers has an intercept of 14.82169 and a slope of 0.05469. The coefficient between data is 0.228299.

4. Discuss the results from the 3 analyses - what did you learn about the impact of the different factors on sales? What are some examples of decisions you might make if you were on the sales team and you had collected this data?

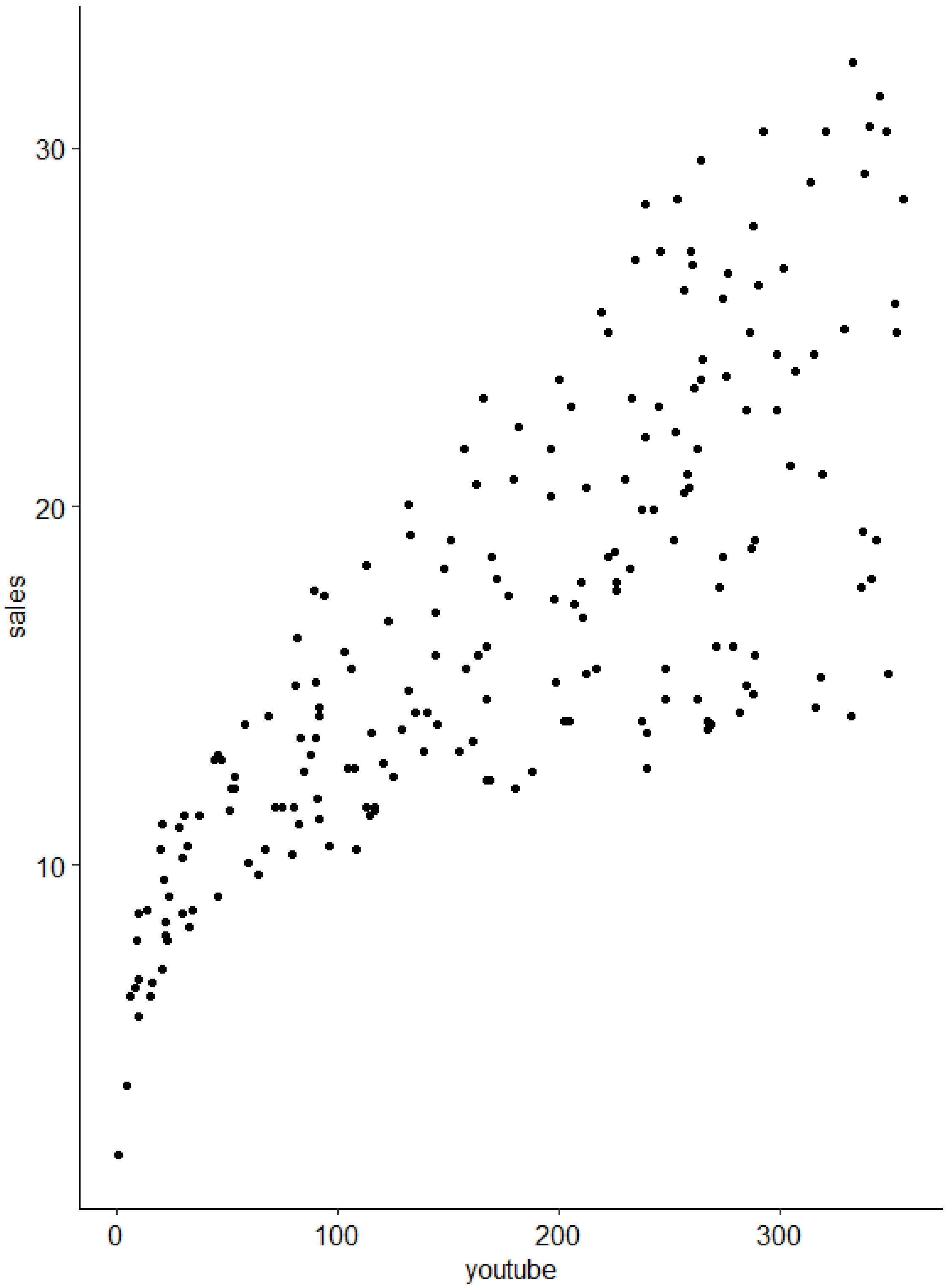
Comparing the three plots, it is clear that youtube has the highest correlation (~0.78) between the advertising budget but significantly smaller slop (~0.048) compared to Facebook(0.2025). The small correlation coefficient for newspapers implies a weak correlation between sales and budget in newspapers. I would have decided to go with Facebook for its high sales ratio, although the sales do not correlate that strongly (~0.58).

PART 2: POLYNOMIAL REGRESSION

https://www.statology.org/polynomial-regression-r/

5. Paste a screenshot of your R code and output (plots etc.) for replicating the tutorial (for youtube). Before each command include a comment describing what the line of code does.

```
#plot scatter point for the sales vs advertising budget for youtube
ggplot(marketing, aes(x=youtube, y= sales)) +
  geom_point()
```



30
20
10
sales
0
100
200
300
youtube

```r
marketing.shuffled <- marketing[sample(nrow(marketing)),]
K<- 10

#define degree of polynomials to fit
degree <- 5

#create k equal-sized folds
folds <- cut(seq(1,nrow(marketing.shuffled)),breaks = K, labels = FALSE)

#create object to hold MSE's of models
mse = matrix(data =NA, nrow=K,ncol = degree)
```

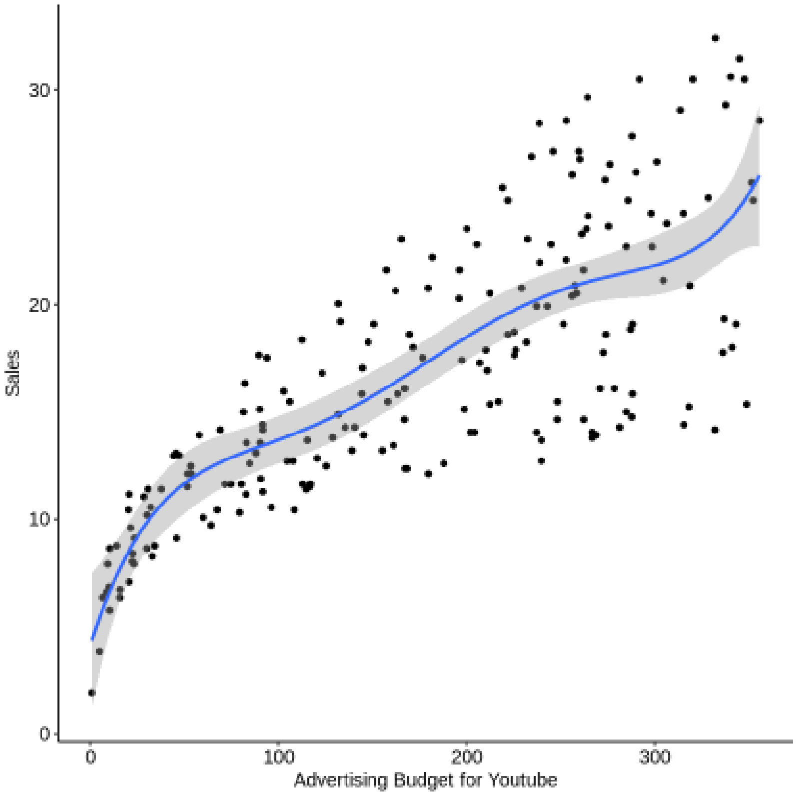
```r
for (i in 1:K){

  #define training and testing data
  testIndexes <- which(folds == i, arr.ind=TRUE)
  testData <- marketing.shuffled[testIndexes,]
  trainData <- marketing.shuffled[-testIndexes,]

  #use k-fold cv to evaluate models
  for (j in 1:degree){
    fit.train = lm(sales~ poly(youtube, j), data=trainData)
    fit.test = predict(fit.train, newdata =testData)
    mse[i,j] = mean((fit.test-testData$sales)^2)
  }
}
```

```r
colMeans(mse)
```

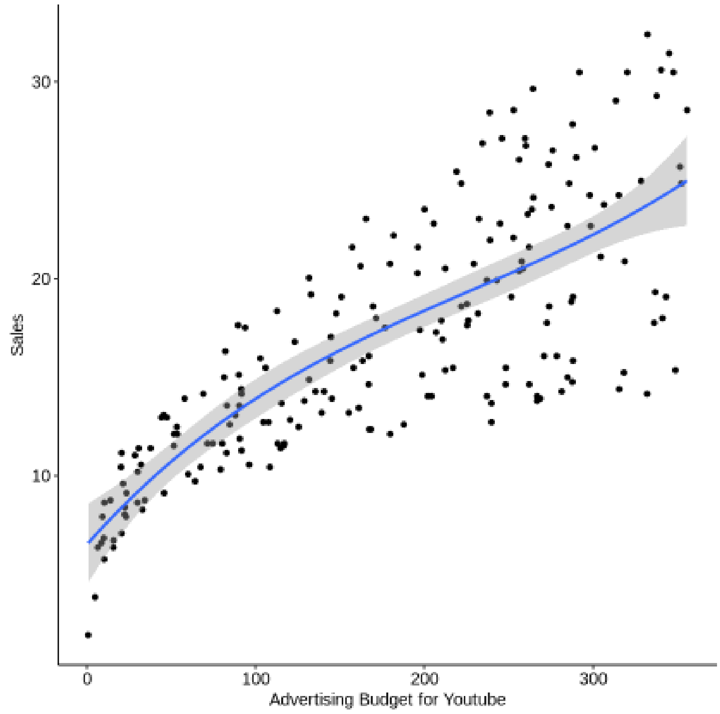
15.2796076578675 · 15.0911816807757 · 15.0675992224909 · 15.2250307588323 · 15.06024457341

6. Create plots for the polynomials of degree 3 and degree 5, paste in your screenshots

```
# plot the fit function for degree 5
ggplot(marketing, aes(x=youtube, y=sales)) +
          geom_point() +
          stat_smooth(method='lm', formula = y ~ poly(x,5), size = 1) +
          xlab('Advertising Budget for Youtube') +
          ylab('Sales')
```



```
In [18]: # plot the fit function for degree 3
         ggplot(marketing, aes(x=youtube, y=sales)) +
                  geom_point() +
                  stat_smooth(method='lm', formula = y ~ poly(x,3), size = 1) +
                  xlab('Advertising Budget for Youtube') +
                  ylab('Sales')
```



PART 3: LINEAR REGRESSION IN PYTHON (SKLEARN)
https://realpython.com/linear-regression-in-python/

1. Go to https://jupyter.utoronto.ca/ and start a new Python3 notebook
2. Starting at "Step 1", follow the tutorial, and paste a screenshot of your code here, including a comment before each line describing what the line does.

In [2]:
```
import numpy as np
from sklearn.linear_model import LinearRegression
```

In [3]:
```
#create original x and y array
x = np.array([5, 15, 25, 35, 45, 55]).reshape((-1, 1))
y = np.array([5, 20, 14, 32, 22, 38])
```

In [4]:
```
#set up the linear regression variable
model = LinearRegression()
```

In [5]:
```
# the linear regression to fit current data
model.fit(x, y)
```

Out[5]:
```
LinearRegression()
```

In [6]:
```
#examine the coefficient of determination of the fit
r_sq = model.score(x, y)
print(f"coefficient of determination: {r_sq}")
```

```
coefficient of determination: 0.7158756137479542
```

In [7]:
```
# get the intercept and slop of the fit line
print(f"intercept: {model.intercept_}")
print(f"slope: {model.coef_}")
```

```
intercept: 5.633333333333329
slope: [0.54]
```

In [8]:
```
# same result when providing a two-dimensional y array
new_model = LinearRegression().fit(x, y.reshape((-1, 1)))
print(f"intercept: {new_model.intercept_}")
print(f"slope: {new_model.coef_}")
```

```
intercept: [5.63333333]
slope: [[0.54]]
```

In [9]:
```
# check out the prediction of y according to fit line with given x
y_pred = model.predict(x)
print(f"predicted response:\n{y_pred}")
```

```
predicted response:
[ 8.33333333 13.73333333 19.13333333 24.53333333 29.93333333 35.33333333]
```

In [10]:
```
x
```

Out[10]:
```
array([[ 5],
       [15],
       [25],
       [35],
       [45],
       [55]])
```

In [11]:
```
# set up a new x arange to try prediction with our model
x_new = np.arange(5).reshape((-1,1))
x_new
```

Out[11]:
```
array([[0],
       [1],
       [2],
       [3],
       [4]])
```

In [12]:
```
# The y prediction with given new x array
y_new = model.predict(x_new)
y_new
```

Out[12]:
```
array([5.63333333, 6.17333333, 6.71333333, 7.25333333, 7.79333333])
```

In [13]:
```
# again, set up the original x and y array, while x has set of two values
x = [[0, 1], [5, 1], [15, 2], [25, 5], [35, 11], [45, 15], [55, 34], [60, 35]]
y = [4, 5, 20, 14, 32, 22, 38, 43]
x, y = np.array(x), np.array(y)
```

In [15]:
```
# set up linear regression model
model = LinearRegression().fit(x,y)
```

In [16]:
```
# get the r^2, intercept and slop value, where there a slop for each x set
r_sq = model.score(x,y)
print(f"coefficient of determination: {r_sq}")
print(f"intercept: {model.intercept_}")
print(f"coefficients: {model.coef_}")
```

```
coefficient of determination: 0.8615939258756776
intercept: 5.52257927519819
coefficients: [0.44706965 0.25502548]
```

In [17]:
```
# prediction same as the simple linear regression
y_pred = model.predict(x)
print(f"predicted response:\n{y_pred}")
```

```
predicted response:
[ 5.77760476  8.012953   12.73867497 17.9744479  23.97529728 29.4660957
 38.78227633 41.27265006]
```

In [18]:
```
# use a new set of x to predict y using the fit line
x_new = np.arange(10).reshape((-1, 2))
x_new
y_new = model.predict(x_new)
y_new
```

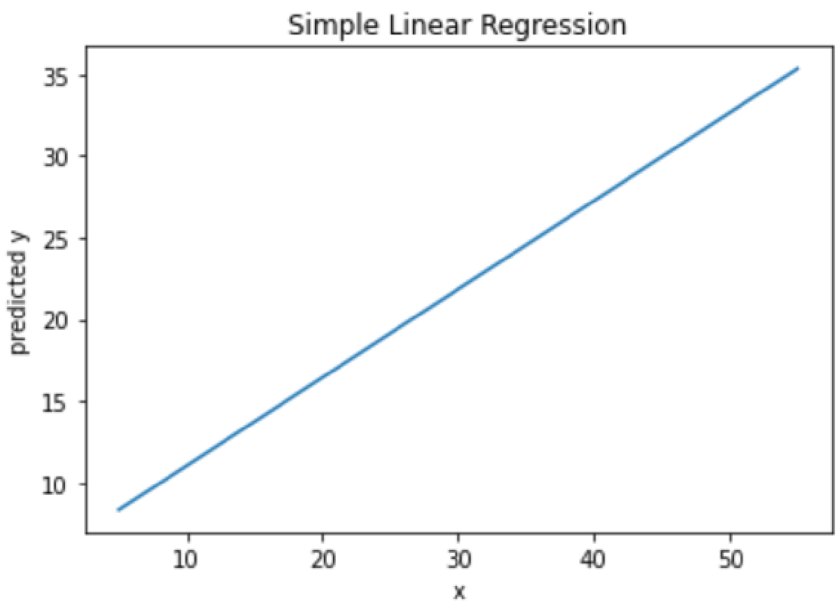
Out[18]:
```
array([ 5.77760476,  7.18179502,  8.58598528,  9.99017554, 11.3943658 ])
```

```
In [32]: # adding the needed package
         from sklearn.preprocessing import PolynomialFeatures

In [33]: # set up new x and y array
         x = np.array([5, 15, 25, 35, 45, 55]).reshape((-1, 1))
         y = np.array([15, 11, 2, 8, 25, 32])

In [34]: # set up a transformer to manipulate input array to include x^2
         transformer = PolynomialFeatures(degree=2, include_bias=False)

In [36]: # applying transformer to x
         transformer.fit(x)

Out[36]: array([[ 5],
                [15],
                [25],
                [35],
                [45],
                [55]])

In [38]: # assign a new variable x_ to the transformed x array
         x_ = transformer.transform(x)
         x_

Out[38]: array([[   5.,   25.],
                [  15.,  225.],
                [  25.,  625.],
                [  35., 1225.],
                [  45., 2025.],
                [  55., 3025.]])

In [39]: # create and fit the model
         model = LinearRegression().fit(x_, y)

In [40]: # get r^2 intercept and coefficients for fitted line
         r_sq = model.score(x_, y)
         print(f"coefficient of determination: {r_sq}")
         print(f"intercept: {model.intercept_}")
         print(f"coefficients: {model.coef_}")

         coefficient of determination: 0.8908516262498564
         intercept: 21.37232142857144
         coefficients: [-1.32357143  0.02839286]

In [42]: # get the predicted y
         y_pred = model.predict(x_)
         print(f"predicted response:\n{y_pred}")

         predicted response:
         [15.46428571  7.90714286  6.02857143  9.82857143 19.30714286 34.46428571]
```

3. Plot the line with the slope and intercept you got, using matplotlib (https://www.w3schools.com/python/matplotlib_line.asp)
   Fit line with simple linear regression

In [26]:
```
import matplotlib.pyplot as plt
plt.plot(x, y_pred)
plt.ylabel("predicted y")
plt.xlabel("x")
plt.title("Simple Linear Regression" )
```

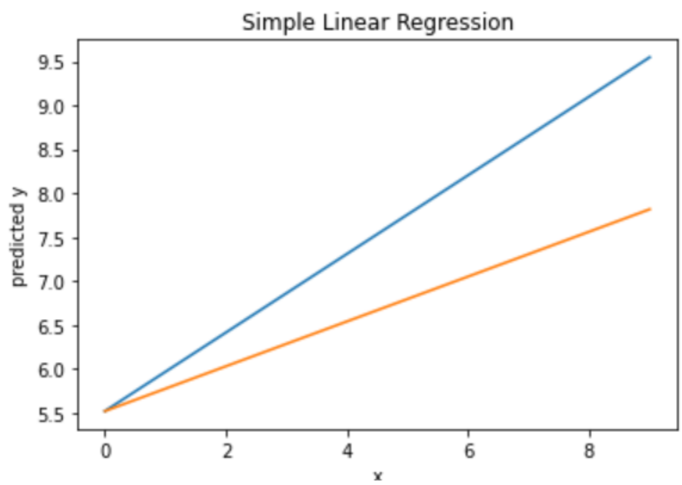
Out[26]: Text(0.5, 1.0, 'Simple Linear Regression')



Fit line for multiple input

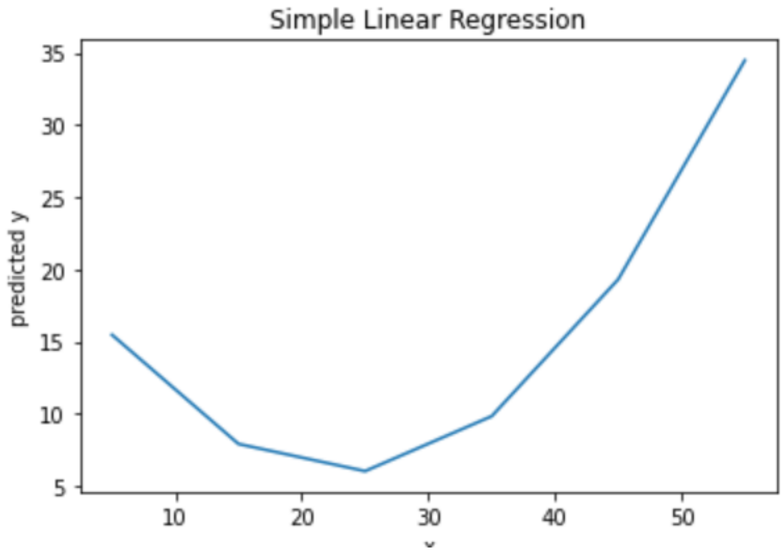
In [31]:
```
plt.plot(np.arange(10), (model.intercept_ + model.coef_[0] * np.arange(10)))
plt.plot(np.arange(10), (model.intercept_ + model.coef_[1] * np.arange(10)))
plt.ylabel("predicted y")
plt.xlabel("x")
plt.title("Simple Linear Regression" )
```

Out[31]: Text(0.5, 1.0, 'Simple Linear Regression')



Fit line with polynomial regression

```
In [43]: plt.plot(x, y_pred)
         plt.ylabel("predicted y")
         plt.xlabel("x")
         plt.title("Simple Linear Regression" )

Out[43]: Text(0.5, 1.0, 'Simple Linear Regression')
```



PART 3: CLASSIFICATION IN PYTHON (SKLEARN)
https://www.activestate.com/resources/quick-reads/how-to-classify-data-in-python/

1. Paste a screenshot of your R code and output (plots etc.) for replicating the tutorial (KNN and Naive Bayes). Before each command include a comment describing what the line of code does.

In [48]:
```python
# Import libraries and classes required for this example:
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import classification_report, confusion_matrix
import pandas as pd

# Import dataset:
url = "Iris.csv"

# Assign column names to dataset:
names = ['sepal-length', 'sepal-width', 'petal-length', 'petal-width', 'Class']

# Convert dataset to a pandas dataframe:
dataset = pd.read_csv(url, names=names)
```

In [49]:
```python
# Use head() function to return the first 5 rows:
dataset.head()
```

Out[49]:

	sepal-length	sepal-width	petal-length	petal-width	Class
0	sepal.length	sepal.width	petal.length	petal.width	variety
1	5.1	3.5	1.4	0.2	Setosa
2	4.9	3	1.4	0.2	Setosa
3	4.7	3.2	1.3	0.2	Setosa
4	4.6	3.1	1.5	0.2	Setosa

In [61]:
```python
# Assign values to the X and y variables:
X = dataset.iloc[1:, :-1].values
y = dataset.iloc[1:, 4].values
```

In [61]:
```python
# Assign values to the X and y variables:
X = dataset.iloc[1:, :-1].values
y = dataset.iloc[1:, 4].values
```

In [62]:
```python
X
```

```
['5.6', '2.9', '3.6', '1.3'],
['6.7', '3.1', '4.4', '1.4'],
['5.6', '3', '4.5', '1.5'],
['5.8', '2.7', '4.1', '1'],
['6.2', '2.2', '4.5', '1.5'],
['5.6', '2.5', '3.9', '1.1'],
['5.9', '3.2', '4.8', '1.8'],
['6.1', '2.8', '4', '1.3'],
['6.3', '2.5', '4.9', '1.5'],
['6.1', '2.8', '4.7', '1.2'],
['6.4', '2.9', '4.3', '1.3'],
['6.6', '3', '4.4', '1.4'],
['6.8', '2.8', '4.8', '1.4'],
['6.7', '3', '5', '1.7'],
['6', '2.9', '4.5', '1.5'],
['5.7', '2.6', '3.5', '1'],
['5.5', '2.4', '3.8', '1.1'],
['5.5', '2.4', '3.7', '1'],
['5.8', '2.7', '3.9', '1.2'],
['6', '2.7', '5.1', '1.6'],
```

In [63]:
```python
# Split dataset into random train and test subsets:
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.20)

# Standardize features by removing mean and scaling to unit variance:
scaler = StandardScaler()
scaler.fit(X_train)

X_train = scaler.transform(X_train)
X_test = scaler.transform(X_test)
```

In [64]:
```
X_train
```

Out[64]:
```
array([[ 0.51713109,  0.7575646 ,  1.01525085,  1.53645828],
       [ 0.39663453,  0.7575646 ,  0.90207844,  1.40578855],
       [ 0.1556414 , -0.35377711,  0.39280258,  0.36043068],
       [-0.08535173,  2.09117465, -1.47454223, -1.33827585],
       [-0.44684143, -1.24285048,  0.10987155,  0.09909121],
       [-1.29031739,  0.09075957, -1.2481974 , -1.33827585],
       [ 0.39663453, -1.90965551,  0.39280258,  0.36043068],
       [ 2.20408301, -0.57604545,  1.63769912,  1.01377935],
       [ 1.24011048,  0.31302792,  1.07183706,  1.40578855],
       [ 0.99911735,  0.53529626,  1.07183706,  1.14444908],
       [ 0.27613796, -0.35377711,  0.505975  ,  0.22976095],
       [ 0.99911735,  0.09075957,  0.33621638,  0.22976095],
       [-1.16982082, -1.24285048,  0.39280258,  0.62177015],
       [ 0.39663453, -0.35377711,  0.27963017,  0.09909121],
       [ 0.51713109, -1.24285048,  0.67573362,  0.88310961],
       [-1.04932426,  0.97983294, -1.2481974 , -0.81559692],
       [ 0.39663453, -0.57604545,  0.5625612 ,  0.75243988],
       [-0.44684143, -1.46511882, -0.00330086, -0.16224825],
       [-0.32634486, -0.57604545,  0.61914741,  1.01377935],
```

In [65]:
```
# Use the KNN classifier to fit data:
classifier = KNeighborsClassifier(n_neighbors=5)
classifier.fit(X_train, y_train)

# Predict y data with classifier:
y_predict = classifier.predict(X_test)

# Print results:
print(confusion_matrix(y_test, y_predict))
print(classification_report(y_test, y_predict))
```

```
[[11  0  0]
 [ 0 10  2]
 [ 0  0  7]]
              precision    recall  f1-score   support

      Setosa       1.00      1.00      1.00        11
  Versicolor       1.00      0.83      0.91        12
   Virginica       0.78      1.00      0.88         7

    accuracy                           0.93        30
   macro avg       0.93      0.94      0.93        30
weighted avg       0.95      0.93      0.93        30
```

```
In [1]: # Import dataset and classes needed in this example:
        from sklearn.datasets import load_iris
        from sklearn.model_selection import train_test_split

        # Import Gaussian Naive Bayes classifier:
        from sklearn.naive_bayes import GaussianNB
        from sklearn.metrics import accuracy_score

        # Load dataset:
        data = load_iris()

        # Organize data:
        label_names = data['target_names']
        labels = data['target']
        feature_names = data['feature_names']
        features = data['data']

        # Print data:
        print(label_names)
        print('Class label = ', labels[0])
        print(feature_names)
        print(features[0])

        # Split dataset into random train and test subsets:
        train, test, train_labels, test_labels = train_test_split(features, labels, test_size=0.33, random_state=42)

        # Initialize classifier:
        gnb = GaussianNB()

        # Train the classifier:
        model = gnb.fit(train, train_labels)
        # Make predictions with the classifier:
        predictive_labels = gnb.predict(test)
        print(predictive_labels)

        # Evaluate label (subsets) accuracy:
        print(accuracy_score(test_labels, predictive_labels))

        ['setosa' 'versicolor' 'virginica']
        Class label =  0
        ['sepal length (cm)', 'sepal width (cm)', 'petal length (cm)', 'petal width (cm)']
        [5.1 3.5 1.4 0.2]
        [1 0 2 1 1 0 1 2 1 1 2 0 0 0 0 2 2 1 1 2 0 2 0 2 2 2 2 2 0 0 0 0 1 0 0 2 1
         0 0 0 2 1 1 0 0 1 1 2 1 2]
        0.96
```

7. What is another classifier you would like to try out on this data, and why?
   I would try to use the decision tree. The feature of selecting categories according to each column fit well with the iris database.

PART 4: MORE CLASSIFIERS: CHOOSE 1

Choose one of the following (if you do more than one, you will get bonus points):

- Run KNN and Naive Bayes classifier on a different dataset, paste your R or Python code and results

In [12]:
```
# Import libraries and classes required for this example:
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import classification_report, confusion_matrix
import pandas as pd

# Import dataset:
url = "winequality-white.csv"

# Assign column names to dataset:
names = ["fixed acidity","volatile acidity","citric acid","residual sugar","chlorides","free sulfur dioxide","total sulfur dioxic

# Convert dataset to a pandas dataframe:
dataset = pd.read_csv(url, names=names)

# Use head() function to return the first 5 rows:
dataset.head()
```

Out[12]:

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcohol	quality
0	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcohol	quality
1	7	0.27	0.36	20.7	0.045	45	170	1.001	3	0.45	8.8	6
2	6.3	0.3	0.34	1.6	0.049	14	132	0.994	3.3	0.49	9.5	6
3	8.1	0.28	0.4	6.9	0.05	30	97	0.9951	3.26	0.44	10.1	6
4	7.2	0.23	0.32	8.5	0.058	47	186	0.9956	3.19	0.4	9.9	6

In [25]:
```
# Assign values to the X and y variables:
X = dataset.iloc[1:, :-1].values.astype("float64")
y = dataset.iloc[1:, -1].values.astype("float64")
```

In [26]:
```
y
```

Out[26]:
```
array([6., 6., 6., ..., 6., 7., 6.])
```

In [27]:
```
# Split dataset into random train and test subsets:
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.20)

# Standardize features by removing mean and scaling to unit variance:
scaler = StandardScaler()
scaler.fit(X_train)

X_train = scaler.transform(X_train)
X_test = scaler.transform(X_test)
```

In [28]:
```
# Use the KNN classifier to fit data:
classifier = KNeighborsClassifier(n_neighbors=5)
classifier.fit(X_train, y_train)

# Predict y data with classifier:
y_predict = classifier.predict(X_test)
```

In [27]:
```
# Split dataset into random train and test subsets:
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.20)

# Standardize features by removing mean and scaling to unit variance:
scaler = StandardScaler()
scaler.fit(X_train)

X_train = scaler.transform(X_train)
X_test = scaler.transform(X_test)
```

In [28]:
```
# Use the KNN classifier to fit data:
classifier = KNeighborsClassifier(n_neighbors=5)
classifier.fit(X_train, y_train)

# Predict y data with classifier:
y_predict = classifier.predict(X_test)

# Print results:
print(confusion_matrix(y_test, y_predict))
print(classification_report(y_test, y_predict))
```

```
[[  0   0   1   2   0   0   0]
 [  0   3  20  10   4   0   0]
 [  0   8 170  99   8   1   0]
 [  0   1 100 286  41   2   0]
 [  0   0  14  89  73   7   0]
 [  0   0   0  17  19   4   0]
 [  0   0   0   1   0   0   0]]
              precision    recall  f1-score   support

         3.0       0.00      0.00      0.00         3
         4.0       0.25      0.08      0.12        37
         5.0       0.56      0.59      0.58       286
         6.0       0.57      0.67      0.61       430
         7.0       0.50      0.40      0.45       183
         8.0       0.29      0.10      0.15        40
         9.0       0.00      0.00      0.00         1

    accuracy                           0.55       980
   macro avg       0.31      0.26      0.27       980
weighted avg       0.53      0.55      0.53       980
```

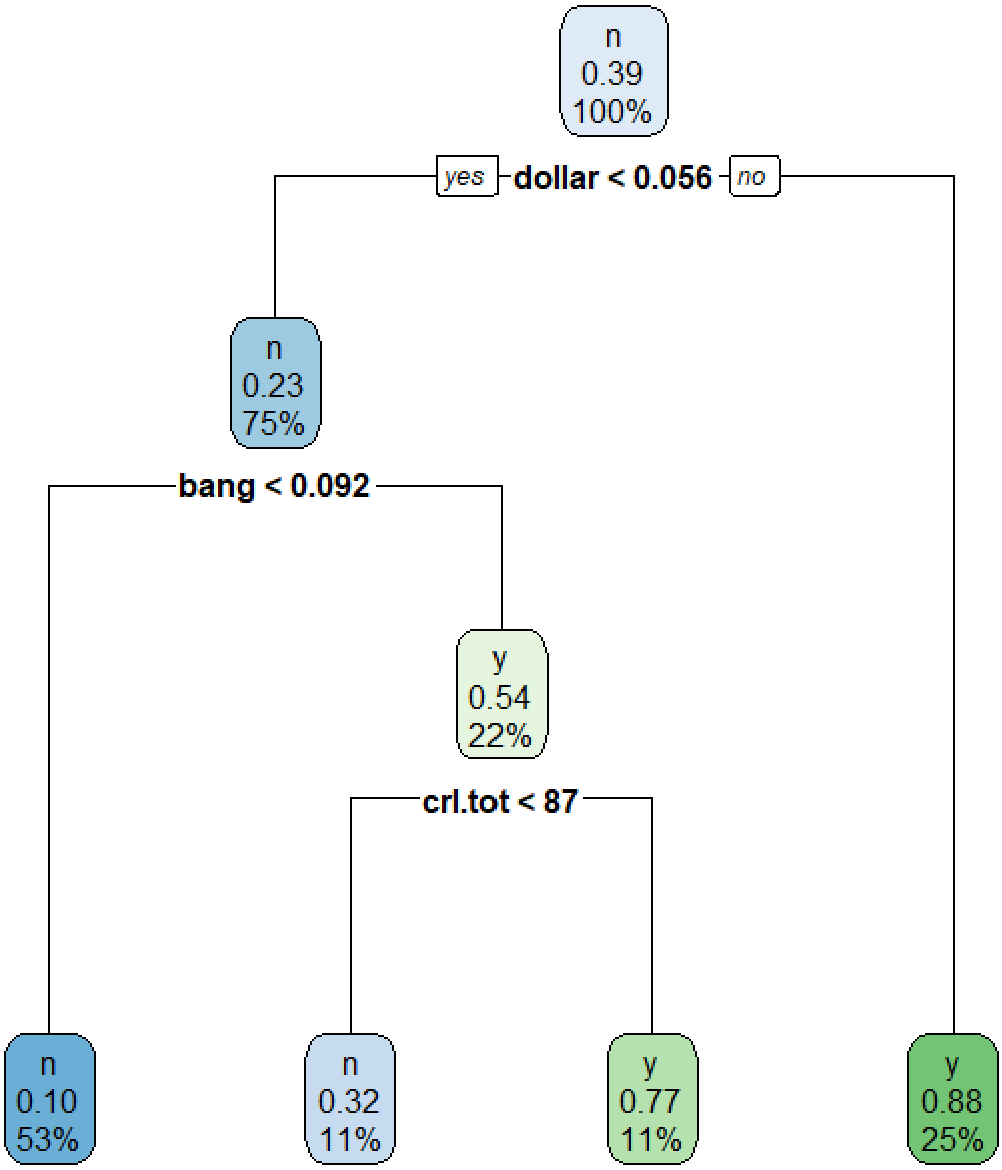
In [9]:
```
# Import dataset and classes needed in this example:
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split

# Import Gaussian Naive Bayes classifier:
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import accuracy_score


# Organize data:
labels = dataset.iloc[1:, -1].values.astype("float64")
feature_names = dataset.iloc[0, :-1].values
features = dataset.iloc[1:, :-1].values.astype("float64")

# Print data:
print('Class label = ', labels[0])
print(feature_names)
print(features[0])

# Split dataset into random train and test subsets:
train, test, train_labels, test_labels = train_test_split(features, labels, test_size=0.33, random_state=42)

# Initialize classifier:
gnb = GaussianNB()

# Train the classifier:
model = gnb.fit(train, train_labels)
# Make predictions with the classifier:
predictive_labels = gnb.predict(test)
print(predictive_labels)

# Evaluate label (subsets) accuracy:
print(accuracy_score(test_labels, predictive_labels))
```

```
Class label =  6.0
['fixed acidity' 'volatile acidity' 'citric acid' 'residual sugar'
 'chlorides' 'free sulfur dioxide' 'total sulfur dioxide' 'density' 'pH'
 'sulphates' 'alcohol']
[7.000e+00 2.700e-01 3.600e-01 2.070e+01 4.500e-02 4.500e+01 1.700e+02
 1.001e+00 3.000e+00 4.500e-01 8.800e+00]
[6. 7. 7. ... 7. 7. 6.]
0.4260977118119975
```

- Follow this tutorial to do a logistic regression for diabetes data http://www.sthda.com/english/articles/36-classification-methods-essentials/151-logistic-regression-essentials-in-r/
- Follow this tutorial to do a basic decision tree in R: https://www.r-bloggers.com/2021/04/decision-trees-in-r/

```
> library(DAAG)
> library(party)
> library(rpart)
> library(rpart.plot)
> library(mlbench)
> library(caret)
> library(pROC)
> library(tree)
>
> #checking the content of spam7 database
> str(spam7)
'data.frame':	4601 obs. of  7 variables:
 $ crl.tot: num  278 1028 2259 191 191 ...
 $ dollar : num  0 0.18 0.184 0 0 0 0.054 0 0.203 0.081 ...
 $ bang   : num  0.778 0.372 0.276 0.137 0.135 0 0.164 0 0.181 0.244 ...
 $ money  : num  0 0.43 0.06 0 0 0 0 0 0.15 0 ...
 $ n000   : num  0 0.43 1.16 0 0 0 0 0 0 0.19 ...
 $ make   : num  0 0.21 0.06 0 0 0 0 0 0.15 0.06 ...
 $ yesno  : Factor w/ 2 levels "n","y": 2 2 2 2 2 2 2 2 2 2 ...
>
> # assign database to a variable
> mydata <- spam7
```

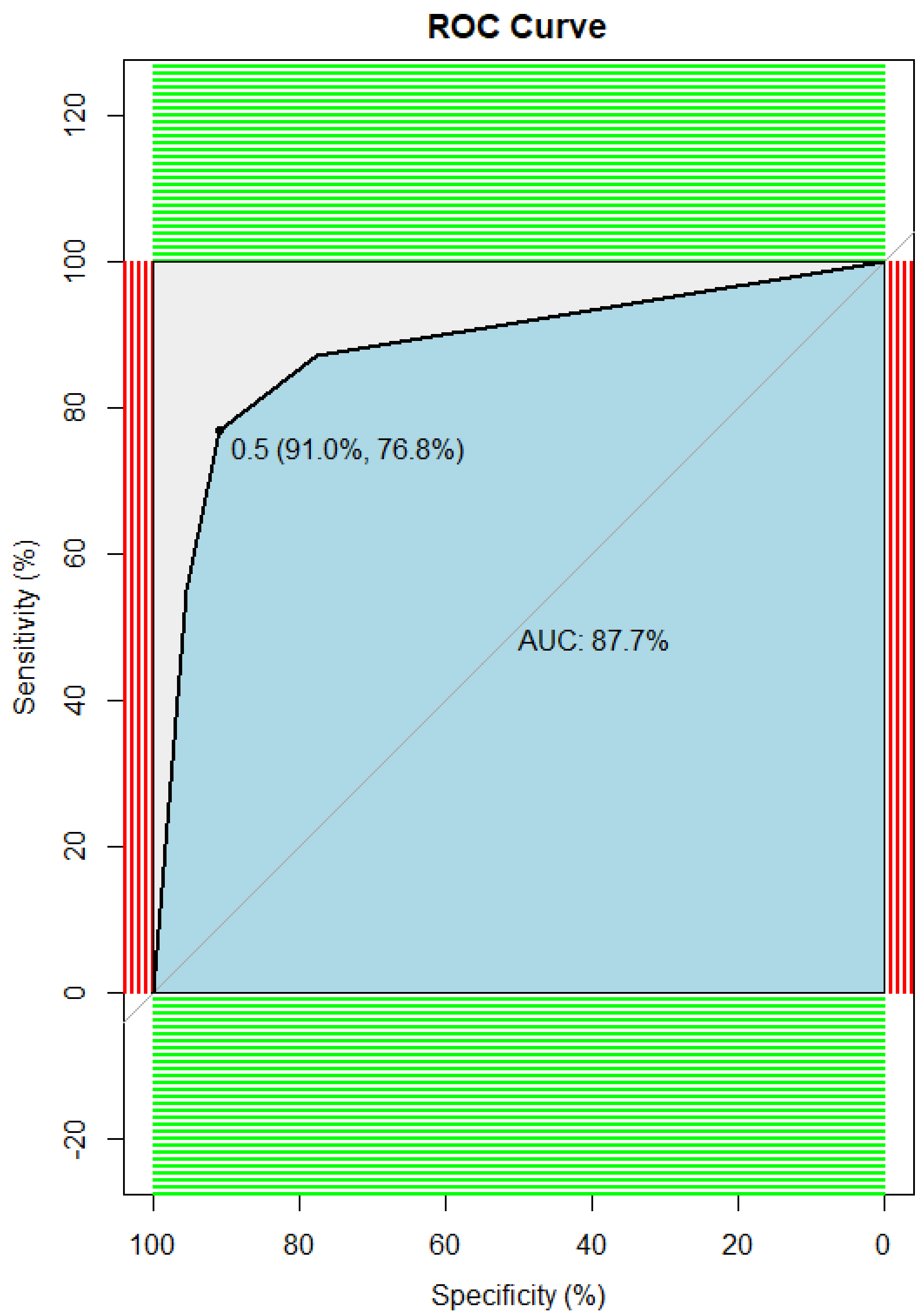
```
> # creating reproduciable sample
> set.seed(1234)
> # create a random sample with value being either 1 or 2, quantity same
> # as the number of row in data, each value take on a probability of 0.5
> ind <- sample(2,nrow(mydata), replace = T,prob = c(0.5,0.5))
>
> train <- mydata[ind == 1,]
> test <- mydata[ind == 2,]
>
> #Tree Classification
> tree <- rpart(yesno ~., data = train)
> rpart.plot(tree)
```


n
0.39
100%
yes
dollar < 0.056
no
n
0.23
75%
bang < 0.092
y
0.54
22%
crl.tot < 87
n
0.10
53%
n
0.32
11%
y
0.77
11%
y
0.88
25%

```
> printcp(tree)

Classification tree:
rpart(formula = yesno ~ ., data = train)

Variables actually used in tree construction:
[1] bang    crl.tot dollar

Root node error: 900/2305 = 0.39046

n= 2305

        CP nsplit rel error  xerror     xstd
1 0.474444      0   1.00000 1.00000 0.026024
2 0.074444      1   0.52556 0.56556 0.022128
3 0.010000      3   0.37667 0.42111 0.019773
> #Classification tree:
>   rpart(formula = yesno ~ ., data = train)
n= 2305

node), split, n, loss, yval, (yprob)
      * denotes terminal node

 1) root 2305 900 n (0.6095445 0.3904555)
   2) dollar< 0.0555 1740 404 n (0.7678161 0.2321839)
     4) bang< 0.092 1227 128 n (0.8956805 0.1043195) *
     5) bang>=0.092 513 237 y (0.4619883 0.5380117)
      10) crl.tot< 86.5 263  84 n (0.6806084 0.3193916) *
      11) crl.tot>=86.5 250  58 y (0.2320000 0.7680000) *
   3) dollar>=0.0555 565  69 y (0.1221239 0.8778761) *
>
```

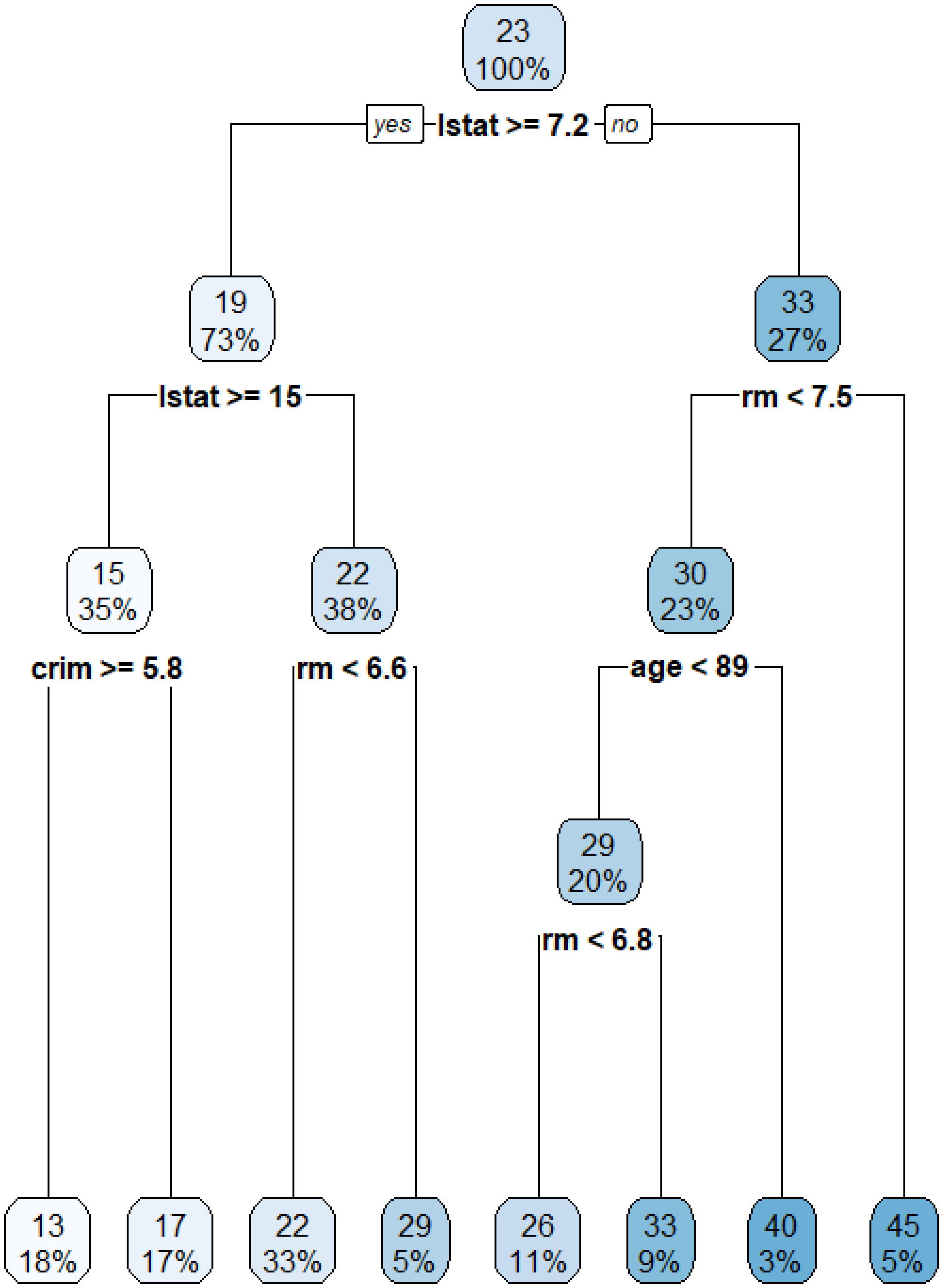
```
> tree <- rpart(yesno ~., data = train,cp=0.07444)
>
> p <- predict(tree, train, type = 'class')
> confusionMatrix(p, train$yesno, positive='y')
Confusion Matrix and Statistics

          Reference
Prediction    n    y
         n 1278  212
         y  127  688

               Accuracy : 0.8529
                 95% CI : (0.8378, 0.8671)
    No Information Rate : 0.6095
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 0.6857

 Mcnemar's Test P-Value : 5.061e-06

            Sensitivity : 0.7644
            Specificity : 0.9096
         Pos Pred Value : 0.8442
         Neg Pred Value : 0.8577
             Prevalence : 0.3905
         Detection Rate : 0.2985
   Detection Prevalence : 0.3536
      Balanced Accuracy : 0.8370

       'Positive' Class : y
```

```
#ROC
p1 <- predict(tree, test, type = 'prob')
p1 <- p1[,2]
r <- multiclass.roc(test$yesno, p1, percent = TRUE)
roc <- r[['rocs']]
r1 <- roc[[1]]
plot.roc(r1,
         print.auc=TRUE,
         auc.polygon=TRUE,
         grid=c(0.1, 0.2),
         grid.col=c("green", "red"),
         max.auc.polygon=TRUE,
         auc.polygon.col="lightblue",
         print.thres=TRUE,
         main= 'ROC Curve')
```



ROC Curve
0.5 (91.0%, 76.8%)
AUC: 87.7%
Sensitivity (%)
Specificity (%)
120
100
80
60
40
20
0
-20
100
80
60
40
20
0

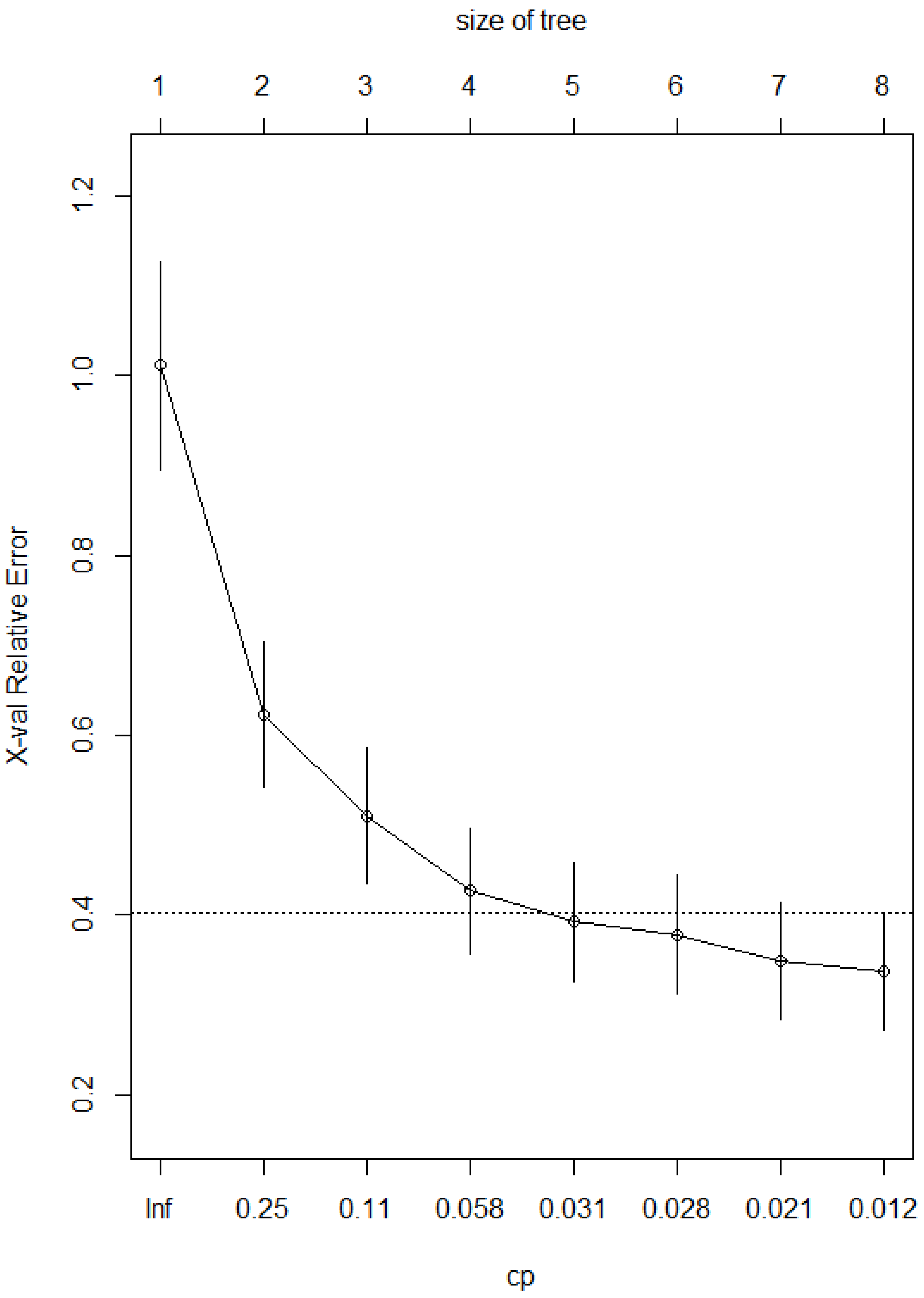
```r
# Method 2 Regression Tree
data('BostonHousing')
mydata <- BostonHousing

#Prepare data
set.seed(1234)
ind <- sample(2, nrow(mydata), replace = T, prob = c(0.5, 0.5))
train <- mydata[ind == 1,]
test <- mydata[ind == 2,]
#Regression tree
tree <- rpart(medv ~., data = train)
rpart.plot(tree)
```


23
100%
yes
lstat >= 7.2
no
19
73%
33
27%
lstat >= 15
rm < 7.5
15
35%
22
38%
30
23%
crim >= 5.8
rm < 6.6
age < 89
29
20%
rm < 6.8
13
18%
17
17%
22
33%
29
5%
26
11%
33
9%
40
3%
45
5%

```
> printcp(tree)

Classification tree:
rpart(formula = yesno ~ ., data = train, cp = 0.07444)

Variables actually used in tree construction:
[1] bang    crl.tot dollar

Root node error: 900/2305 = 0.39046

n= 2305

        CP nsplit rel error  xerror     xstd
1 0.474444      0   1.00000 1.00000 0.026024
2 0.074444      1   0.52556 0.53667 0.021710
3 0.074440      3   0.37667 0.49889 0.021127
```

```
> #Regression tree:
> rpart(formula = medv ~ ., data = train)
n= 262

node), split, n, deviance, yval
      * denotes terminal node

 1) root 262 22619.5900 22.64809
   2) lstat>=7.15 190  6407.5790 18.73000
     4) lstat>=14.8 91  1204.9870 14.64725
       8) crim>=5.7819 47   483.9983 12.77021 *
       9) crim< 5.7819 44   378.5098 16.65227 *
     5) lstat< 14.8 99  2291.4410 22.48283
      10) rm< 6.6365 87  1491.4180 21.52874 *
      11) rm>=6.6365 12   146.6600 29.40000 *
   3) lstat< 7.15 72  5598.1990 32.98750
     6) rm< 7.4525 59  2516.6520 30.37458
      12) age< 88.6 52  1024.2690 29.05385
        24) rm< 6.776 29   220.2917 25.94483 *
        25) rm>=6.776 23   170.2243 32.97391 *
      13) age>=88.6 7   727.8686 40.18571 *
     7) rm>=7.4525 13   850.5723 44.84615 *
```

```
> rpart.rules(tree)
 medv
   13 when lstat >=        14.8                               & crim >= 5.8
   17 when lstat >=        14.8                               & crim <  5.8
   22 when lstat is 7.2 to 14.8 & rm <  6.6
   26 when lstat <  7.2         & rm <  6.8         & age <  89
   29 when lstat is 7.2 to 14.8 & rm >=         6.6
   33 when lstat <  7.2         & rm is 6.8 to 7.5 & age <  89
   40 when lstat <  7.2         & rm <  7.5         & age >= 89
   45 when lstat <  7.2         & rm >=         7.5
```



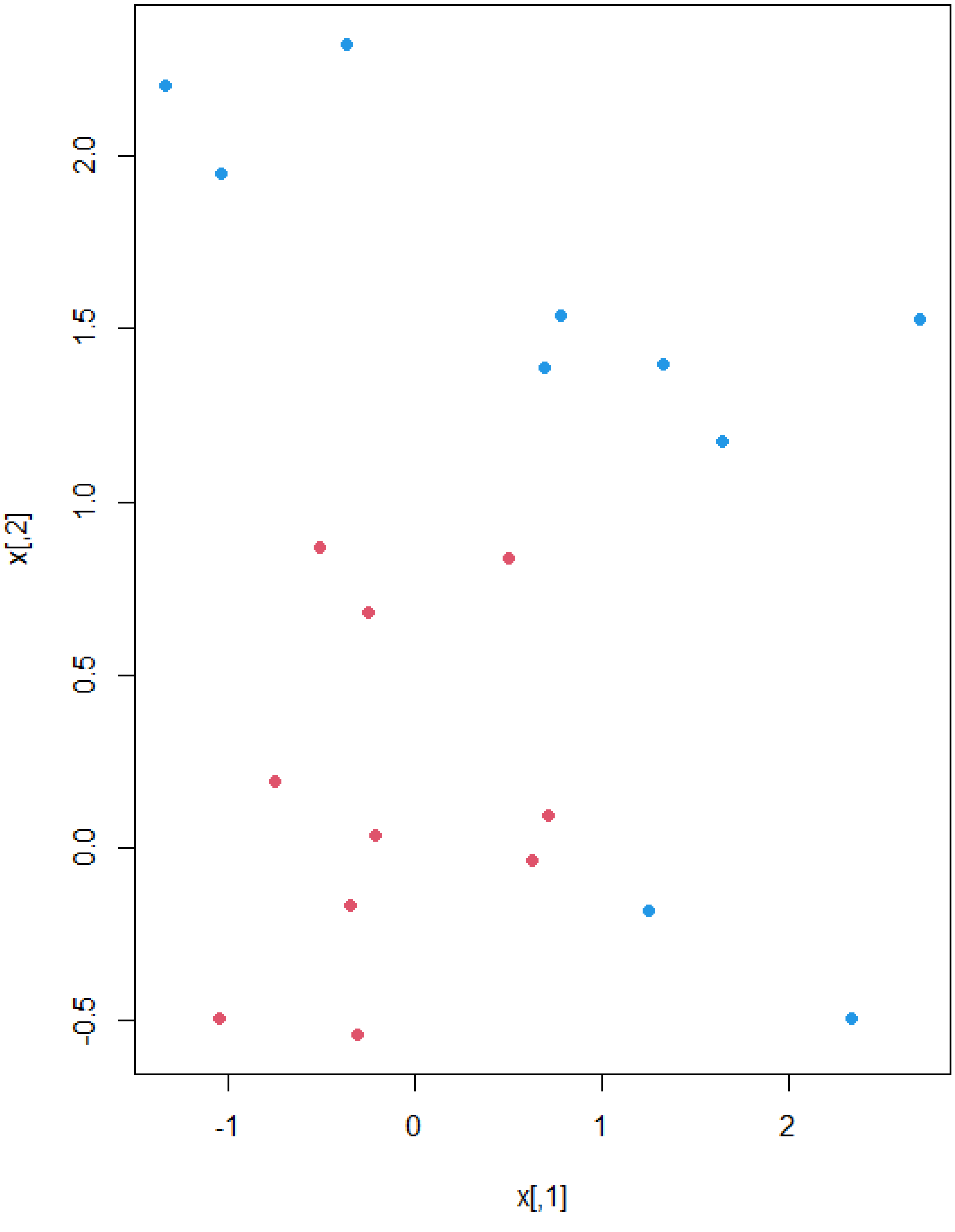

```
> #predict
> p <- predict(tree, train)
> sqrt(mean((train$medv-p)^2))
[1] 4.130294
> (cor(train$medv,p))^2
[1] 0.8024039
```

- Follow this tutorial on boosted trees (XGboost) in python (sklearn) https://towardsdatascience.com/a-beginners-guide-to-xgboost-87f5d4c30ed7

```
In [7]: from sklearn.model_selection import train_test_split

        X_train, X_test, Y_train, Y_test = train_test_split(X, y, test_size=0.2)

In [10]: D_train = xgb.DMatrix(X_train, label=Y_train)
         D_test = xgb.DMatrix(X_test, label=Y_test)

In [11]: param = {
             'eta': 0.3,
             'max_depth': 3,
             'objective': 'multi:softprob',
             'num_class': 3}

         steps = 20  # The number of training iterations

In [12]: odel = xgb.train(param, D_train, steps)

In [14]: model = xgb.train(param, D_train, steps)

In [15]: import numpy as np
         from sklearn.metrics import precision_score, recall_score, accuracy_score

         preds = model.predict(D_test)
         best_preds = np.asarray([np.argmax(line) for line in preds])

         print("Precision = {}".format(precision_score(Y_test, best_preds, average='macro')))
         print("Recall = {}".format(recall_score(Y_test, best_preds, average='macro')))
         print("Accuracy = {}".format(accuracy_score(Y_test, best_preds)))

         Precision = 1.0
         Recall = 1.0
         Accuracy = 1.0
```

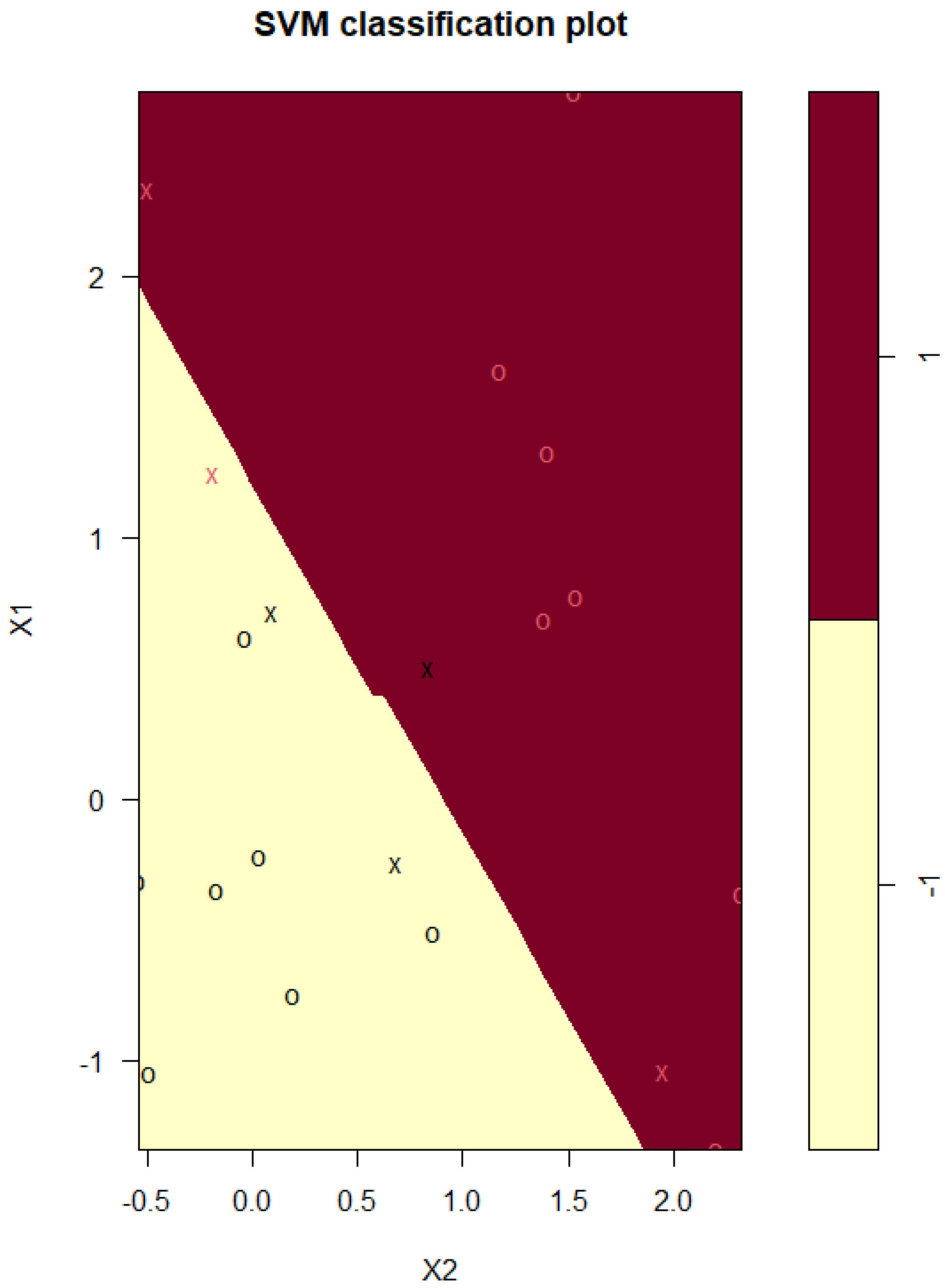
-
-
- Follow this tutorial to train a perceptron classifier and plot the decision boundary https://databricks-prod-cloudfront.cloud.databricks.com/public/4027ec902e239c93eaaa8714f173bcfc/2961012104553482/2761297084239405/1806228006848429/latest.html
- Follow this tutorial to train a neural network on loan data in R https://www.pluralsight.com/guides/machine-learning-with-neural-networks-r
  With this data: https://www.kaggle.com/code/panamby/bank-loan-status-dataset/data
- Follow this tutorial to use an SVM on mixture data, plot the decision boundary https://www.datacamp.com/community/tutorials/support-vector-machines-r

```
> set.seed(10111)
> x = matrix(rnorm(40), 20, 2)
> y = rep(c(-1, 1), c(10, 10))
> x[y == 1,] = x[y == 1,] + 1
> plot(x, col = y + 3, pch = 19)
```



2.0
1.5
1.0
0.5
0.0
-0.5
x[,2]
-1
0
1
2
x[,1]

```
> dat = data.frame(x, y = as.factor(y))
> svmfit = svm(y ~ ., data = dat, kernel = "linear",
+              cost = 10, scale = FALSE)
> print(svmfit)

Call:
svm(formula = y ~ ., data = dat, kernel = "linear", cost = 10,
    scale = FALSE)


Parameters:
   SVM-Type:  C-classification
 SVM-Kernel:  linear
       cost:  10

Number of Support Vectors:  6


> plot(svmfit, dat)
```


SVM classification plot
X1
X2
2
1
0
-1
-0.5
0.0
0.5
1.0
1.5
2.0
1
-1

In [6]:
```
make.grid = function(x, n = 75) {
  grange = apply(x, 2, range)
  x1 = seq(from = grange[1,1], to = grange[2,1], length = n)
  x2 = seq(from = grange[1,2], to = grange[2,2], length = n)
  expand.grid(X1 = x1, X2 = x2)
}
```

In [7]:
```
xgrid = make.grid(x)
xgrid[1:10,]
```

A data.frame: 10 × 2

	X1	X2
	<dbl>	<dbl>
1	-1.3406379	-0.5400074
2	-1.2859572	-0.5400074
3	-1.2312766	-0.5400074
4	-1.1765959	-0.5400074
5	-1.1219153	-0.5400074
6	-1.0672346	-0.5400074
7	-1.0125540	-0.5400074
8	-0.9578733	-0.5400074
9	-0.9031927	-0.5400074
10	-0.8485120	-0.5400074

- Follow this tutorial to make a neural network from scratch in Python: https://realpython.com/python-ai-neural-network/