In [1]:
```
x = [99,86,87,88,111,86,103,87,94,78,77,85,86]
```

In [2]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import statistics
from scipy import stats
```

In [3]:
```
x= [99,86,87,88,111,86,103,87,94,78,77,85,86]
# way to calculate mean with help of numpy

b=np.mean(x)
print(b)
```

```
89.76923076923077
```

In [4]:
```
a =x.copy()
```

In [5]:
```
print(a)
```

```
[99, 86, 87, 88, 111, 86, 103, 87, 94, 78, 77, 85, 86]
```

In [6]:
```
# way to find out median and mode
np.median(a)
```

Out[6]:
```
87.0
```

In [7]:
```
# how to find mode
stats.mode(a)
```

Out[7]:
```
ModeResult(mode=array([86]), count=array([3]))
```

In [8]:
```
 # how  to find standard deviation
np.std(a)
```

Out[8]:
```
9.258292301032677
```

In [9]:
```
# way to find variance
np.var(a)
```

Out[9]:
```
85.71597633136093
```

In [10]:
```
# Use the NumPy percentile() method to find the percentiles 75% in this case

ages = [5,31,43,48,50,41,7,11,15,39,80,82,32,2,8,6,25,36,27,61,31]
np.percentile(ages,75)
```

Out[10]:
```
43.0
```

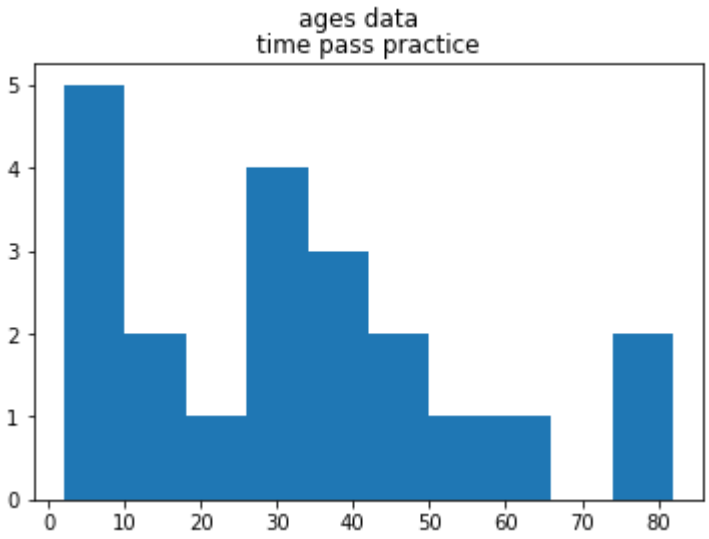
In [11]:
```
# What is the age that 90% of the people are younger than?
np.percentile(ages,90)
```

Out[11]:
```
61.0
```

In [12]:
```
plt.hist(ages)
plt.title("time pass practice")
```

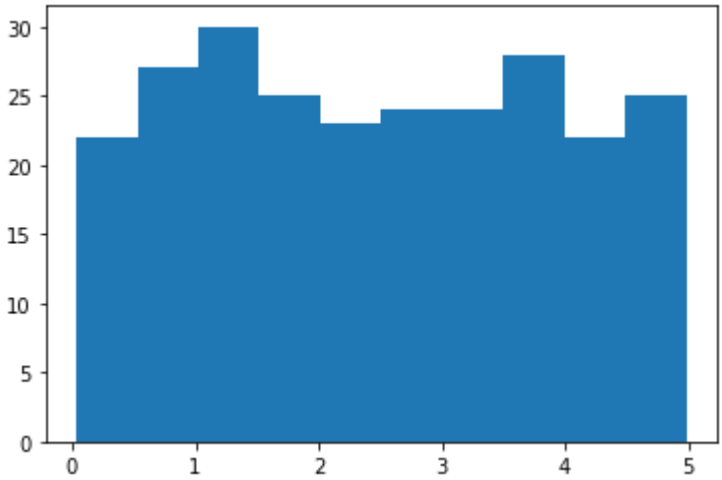
```
plt.suptitle("ages data")
```

Out[12]:
```
Text(0.5, 0.98, 'ages data')
```



In [13]:
```
# Create an array containing 250 random floats between 0 and 5:

x=np.random.uniform(0,5,250)
plt.hist(x)
```

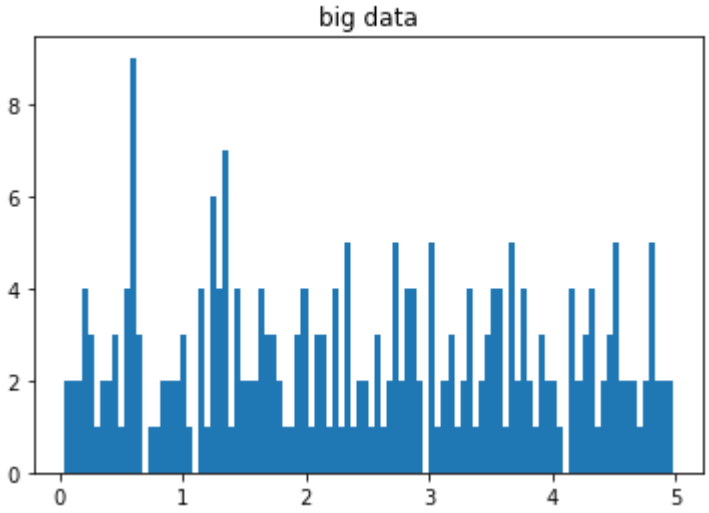
Out[13]:
```
(array([22., 27., 30., 25., 23., 24., 24., 28., 22., 25.]),
 array([0.03520559, 0.53009768, 1.02498977, 1.51988187, 2.01477396,
        2.50966605, 3.00455815, 3.49945024, 3.99434233, 4.48923443,
        4.98412652]),
 <BarContainer object of 10 artists>)
```



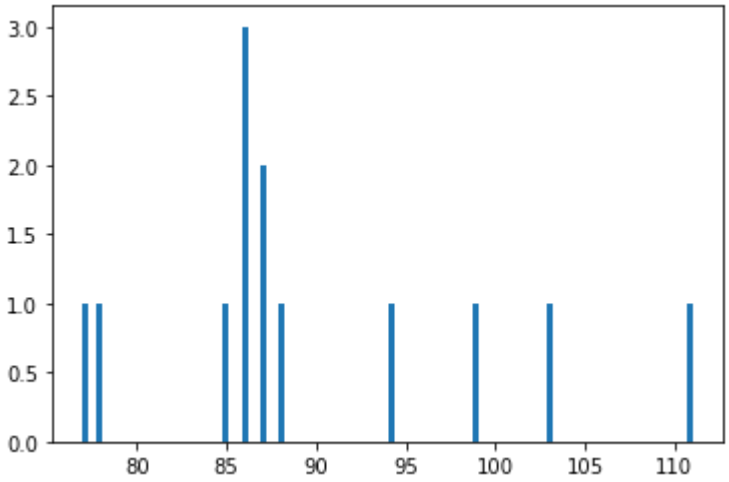
In [14]:
```
# Big Data Distributions
# An array containing 250 values is not considered very big, but now you know how to cr
# changing the parameters, you can create the data set as big as you want.


# Create an array with 100000 random numbers, and display them using a histogram with 1

p=np.random.uniform(0,5,100000)
print(p)

plt.hist(x,100)
plt.title("big data")
```

```
[0.84475072 4.43643861 3.33480459 ... 2.49021199 0.2182957  3.15314032]
```

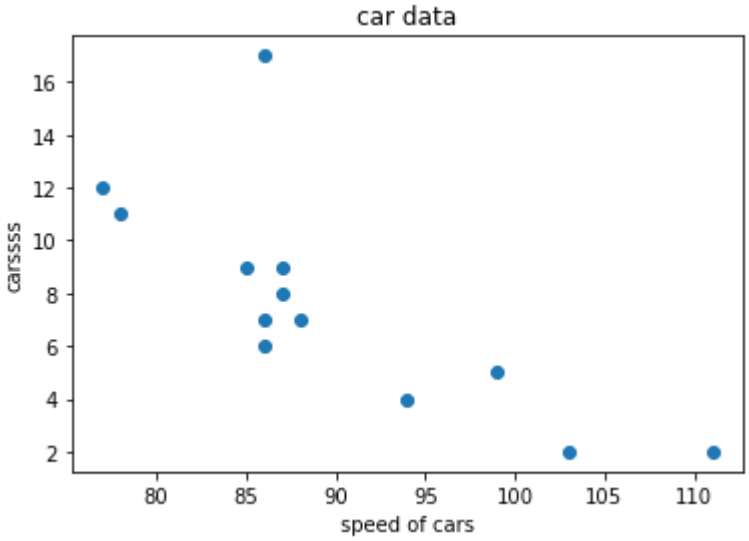
Out[14]: Text(0.5, 1.0, 'big data')



In [15]:
```
# Normal Data Distribution

# a=np.random.normal(5,1,1000)
plt.hist(a,100)
plt.show()
```



In [16]:
```
car = [5,7,8,7,2,17,2,9,4,11,12,9,6]
sp = [99,86,87,88,111,86,103,87,94,78,77,85,86]
plt.scatter(sp,car)
plt.title("car data")
plt.ylabel("carssss")
plt.xlabel("speed of cars")
plt.show()
```



In [17]:
```
# In Python a function is defined using the def keyword:
# learning function is important for linregression
# example
def my_function():
    print("hello print from function")


# To call a function, use the function name followed by parenthesis:
# like
my_function()

def vikram():
    print("print vikram from a function")

vikram()

# Information can be passed into functions as arguments.
def my_god(fname):
    print(fname + " love you so much")

my_god("babaji")

# return function/ return statement
def our_function(x):
    return 5 * x
print(our_function(3))
print(our_function(5))
print(our_function(7))
```

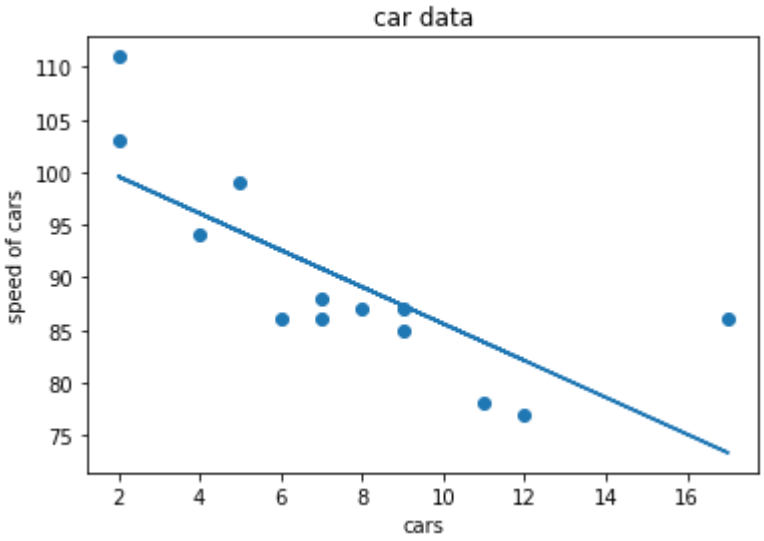
```
hello print from function
print vikram from a function
babaji love you so much
15
25
35
```

In [ ]:

In [ ]:

In [18]:
```
# Linear Regression
#Linear regression uses the relationship between the data-points to draw a straight lin
```

```
# This line can be used to predict future values.

x = [5,7,8,7,2,17,2,9,4,11,12,9,6]
y = [99,86,87,88,111,86,103,87,94,78,77,85,86]


slope, intercept,r,p,std_err = stats.linregress(x,y)

# Create a function that uses the slope and intercept values to return a new value.
# This new value represents where on the y-axis the corresponding x value will be place
def myfunc(x):
    return slope * x + intercept
mymodel = list(map(myfunc,x))


plt.scatter(x,y)
plt.plot(x,mymodel)
plt.xlabel("cars")
plt.ylabel("speed of cars")
plt.title("car data")
plt.show()
```



In [ ]:

In [19]:
```
slope,intercept,r,p,std_err=stats.linregress(x,y)
print(r)
print(p)
print(intercept)
print(std_err)
print(slope)
```

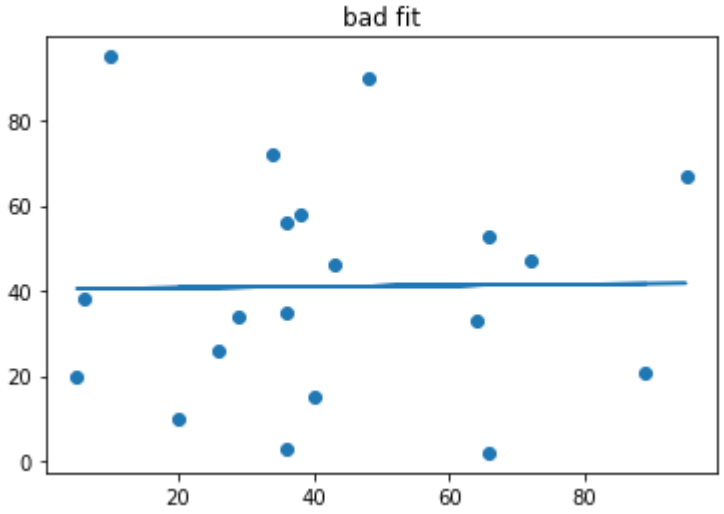
```
-0.7585915243761551
0.002646873922456101
103.10596026490066
0.45353615760774196
-1.751287711552612
```

In [20]:
```
# Predict Future Values
# we need the same myfunc() function from the example given above:
# Predict the speed of a 10 years old car:

x = [5,7,8,7,2,17,2,9,4,11,12,9,6]
y = [99,86,87,88,111,86,103,87,94,78,77,85,86]
```

```
slope,intercept,r,p,std_err = stats.linregress(x,y)

def myfunc(x):
    return slope * x + intercept

speed =myfunc(10)
print(speed)
```

```
85.59308314937454
```

In [21]:
```
# Bad Fit?
# Let us create an example where linear regression would not be the best method to pred

x = [89,43,36,36,95,10,66,34,38,20,26,29,48,64,6,5,36,66,72,40]
y = [21,46,3,35,67,95,53,72,58,10,26,34,90,33,38,20,56,2,47,15]

slope,intercept,r,p,std_err = stats.linregress(x,y)

def yfunc(x):
    return slope * x + intercept

mymodel=list(map(yfunc,x))

plt.scatter(x,y)
plt.plot(x, mymodel)
plt.title("bad fit")
plt.show()
print(r)

# The result: 0.013 indicates a very bad relationship and tells us that this data set i
```

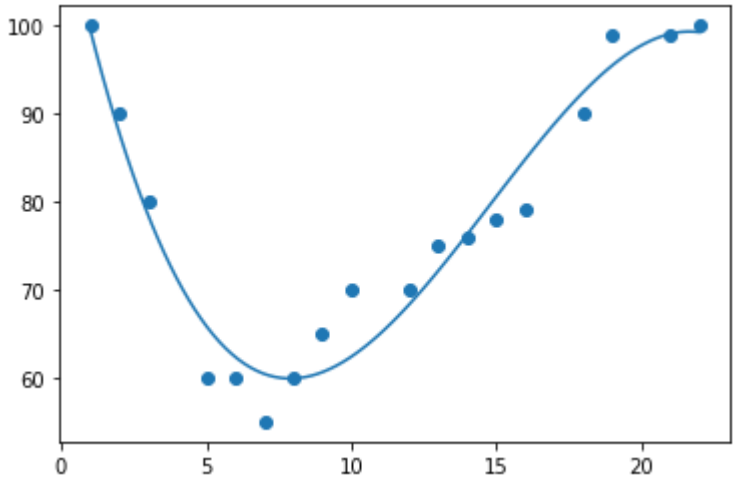



```
0.013318141542974908
```

In [ ]:
```

```

In [22]:
```
# Polynomial Regression
m = [1,2,3,5,6,7,8,9,10,12,13,14,15,16,18,19,21,22]
n= [100,90,80,60,60,55,60,65,70,70,75,76,78,79,90,99,99,100]

mymodel = np.poly1d(np.polyfit(m, n, 3))

myline = np.linspace(1, 22, 100)
```

```
plt.scatter(m, n)
plt.plot(myline, mymodel(myline))
plt.show()
```



In [23]:
```
from sklearn.metrics import r2_score
```

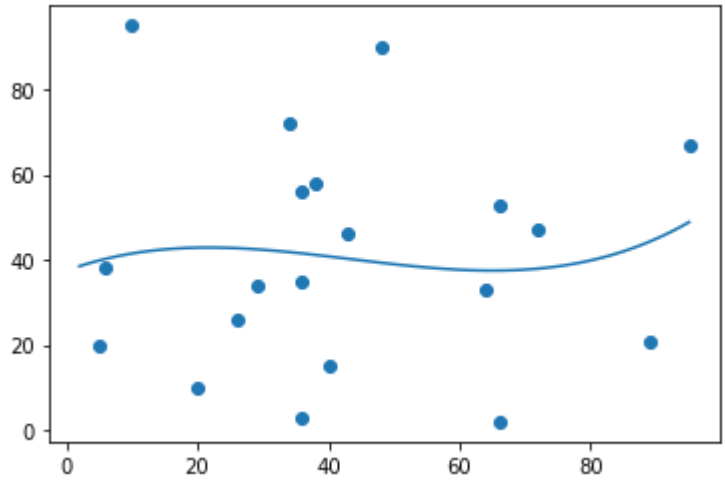
In [24]:
```
# R-Squared
# it tells how well does my data fit in polynomial regression

x = [1,2,3,5,6,7,8,9,10,12,13,14,15,16,18,19,21,22]
y = [100,90,80,60,60,55,60,65,70,70,75,76,78,79,90,99,99,100]

mymodel= np.poly1d(np.polyfit(x,y,3))
print(r2_score(y,mymodel(x)))
```

```
0.9432150416451026
```

In [25]:
```
# Predict Future Values
# Let us try to predict the speed of a car that passes the tollbooth at around 17 P.M:

x = [1,2,3,5,6,7,8,9,10,12,13,14,15,16,18,19,21,22]
y = [100,90,80,60,60,55,60,65,70,70,75,76,78,79,90,99,99,100]

mymodel=np.poly1d(np.polyfit(x,y,3))
speed = mymodel(17)
print(speed)
```

```
88.87331269697998
```

In [26]:
```
# where polynomial regression would not be the best method to predict future values.
x = [89,43,36,36,95,10,66,34,38,20,26,29,48,64,6,5,36,66,72,40]
y = [21,46,3,35,67,95,53,72,58,10,26,34,90,33,38,20,56,2,47,15]

mymodel=np.poly1d(np.polyfit(x,y,3))
myline = np.linspace(2,95,100)

plt.scatter(x,y)
plt.plot(myline,mymodel(myline))
plt.show()
print(r2_score(y,mymodel(x)))
```




```
0.009952707566680652
```

In [27]:
```
import pandas as pd
from sklearn import linear_model
# reading data file

df=pd.read_csv("cars.csv")

# putting the vaule of indpendent variable into X and depedent variable into y and maki

X =df[['Weight','Volume']]
y =df[['CO2']]

# LinearRegression() method to create a linear regression object.
# This object has a method called fit() that takes the independent and dependent values
# and fills the regression object with data that describes the relationship

regr= linear_model.LinearRegression()
regr.fit(X,y)

# regression object that are ready to predict CO2 values based on a car's weight and vo
# predict the CO2 emission of a car where the weight is 2300kg and the volume is 1300cm

predictedco2=regr.predict([[2300,1300]])
print(predictedco2)
# coefficient values of weight and volume.
print(regr.coef_)

# We have predicted that a car with 1.3 liter engine, and a weight of 2300 kg,
# will release approximately 107 grams of CO2 for every kilometer it drives.
```

```
[[107.2087328]]
[[0.00755095 0.00780526]]
```

In [28]:
```
stats.mode(df['Weight'])
```

Out[28]:
```
ModeResult(mode=array([1365], dtype=int64), count=array([3]))
```

In [29]:
```
# Machine Learning - Scale
# scale used for data transformation so that we can compare data in efficiently way
# Scale all values in the Weight and Volume columns

import pandas as pd
from sklearn import linear_model
from sklearn.preprocessing import StandardScaler
```

```
scale=StandardScaler()
df=pd.read_csv('cars.csv')
X=df[['Weight','Volume']]
scaledx = scale.fit_transform(X)
print(scaledx)
```

```
[[-2.10389253 -1.59336644]
 [-0.55407235 -1.07190106]
 [-1.52166278 -1.59336644]
 [-1.78973979 -1.85409913]
 [-0.63784641 -0.28970299]
 [-1.52166278 -1.59336644]
 [-0.76769621 -0.55043568]
 [ 0.3046118  -0.28970299]
 [-0.7551301  -0.28970299]
 [-0.59595938 -0.0289703 ]
 [-1.30803892 -1.33263375]
 [-1.26615189 -0.81116837]
 [-0.7551301  -1.59336644]
 [-0.16871166 -0.0289703 ]
 [ 0.14125238 -0.0289703 ]
 [ 0.15800719 -0.0289703 ]
 [ 0.3046118  -0.0289703 ]
 [-0.05142797  1.53542584]
 [-0.72580918 -0.0289703 ]
 [ 0.14962979  1.01396046]
 [ 1.2219378  -0.0289703 ]
 [ 0.5685001   1.01396046]
 [ 0.3046118   1.27469315]
 [ 0.51404696 -0.0289703 ]
 [ 0.51404696  1.01396046]
 [ 0.72348212 -0.28970299]
 [ 0.8281997   1.01396046]
 [ 1.81254495  1.01396046]
 [ 0.96642691 -0.0289703 ]
 [ 1.72877089  1.01396046]
 [ 1.30990057  1.27469315]
 [ 1.90050772  1.01396046]
 [-0.23991961 -0.0289703 ]
 [ 0.40932938 -0.0289703 ]
 [ 0.47215993 -0.0289703 ]
 [ 0.4302729   2.31762392]]
```

In [30]:

```
# When the data set is scaled, you will have to use the scale when you predict values
# for example (above given eample copy paste here)

import pandas
from sklearn import linear_model
from sklearn.preprocessing import StandardScaler


df=pd.read_csv("cars.csv")

X =df[['Weight','Volume']]
y =df[['CO2']]


scale = StandardScaler()
scaledX = scale.fit_transform(X)

regr= linear_model.LinearRegression()
regr.fit(scaledX,y)
```

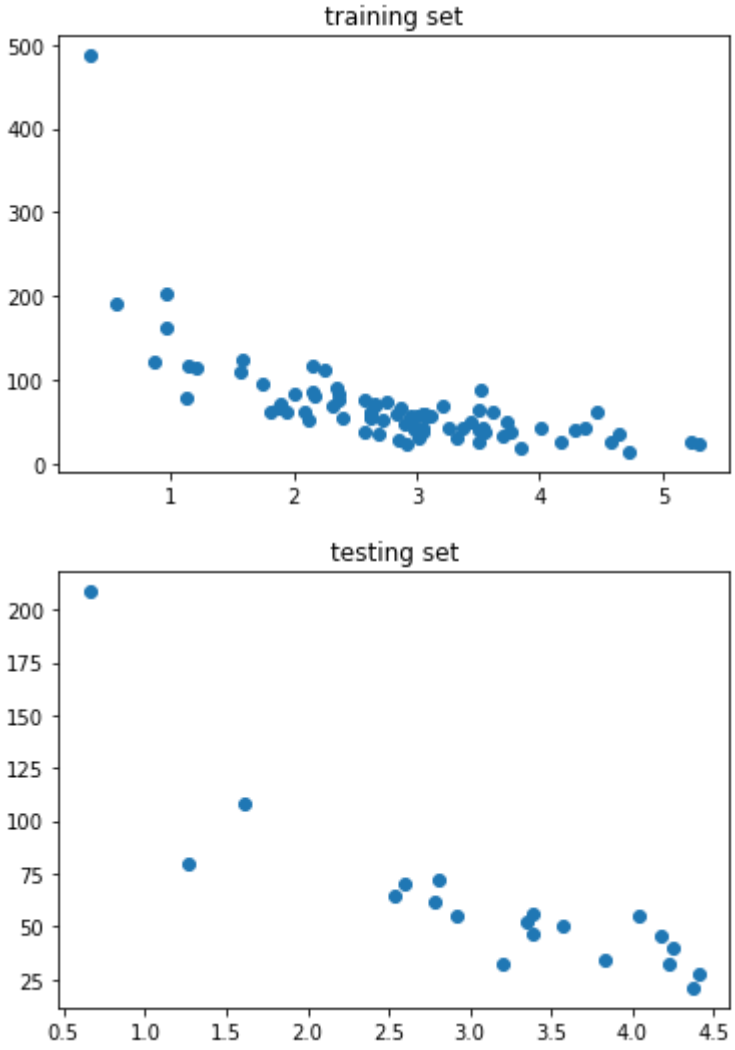
```
scaled = scale.transform([[2300, 1.3]])

predictedCO2 = regr.predict([scaled[0]])
print(predictedCO2)
```

```
[[97.07204485]]
```

In [31]:
```
# Machine Learning - Train/Test
# means evaluate your model
# to measure that model is good enough we can use method called train/test
# train/test is method to measure accuracy of your model its called train/test bec we d
# set and testing set 80% for traning and 20% for testing
# train the model means creat the model and testing means checking accuracy of the mode

import numpy as np
import matplotlib.pyplot as plt
np.random.seed(2)
x=np.random.normal(3,1,100)
y=np.random.normal(150,40,100)/x
plt.scatter(x,y)
plt.title("customer & their shopping habits")
plt.xlabel("time spent bef making purchase")
plt.ylabel("money spent on purchase")
plt.grid()
plt.show()
```

In [32]:
```
# Split Into Train/Test
# The training set should be a random selection of 80% of the original data.
# The testing set should be the remaining 20%.

train_x=x[:80]
train_y=y[:80]

test_x=x[80:]
test_y=y[80:]

# Display the same scatter plot with the training set
plt.scatter(train_x,train_y)
plt.title("training set")
plt.show()
# It looks like the original data set, so it seems to be a fair selection
```

```
# testing set
plt.scatter(test_x,test_y)
plt.title("testing set")
plt.show()
# The testing set also looks like the original data set
```

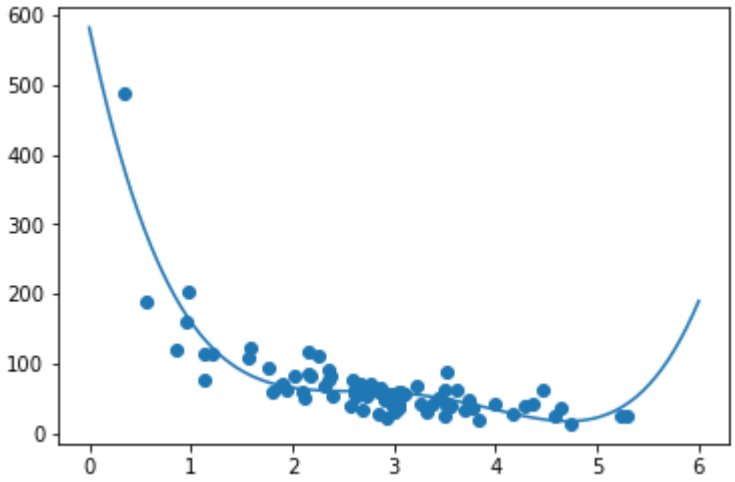


In [33]:

```
# fit the data set
# by looking scatter plot we understand that polynomial regression would be best choice
# data sets
import numpy
import matplotlib.pyplot as plt
numpy.random.seed(2)

x = numpy.random.normal(3, 1, 100)
y = numpy.random.normal(150, 40, 100) / x

train_x = x[:80]
train_y = y[:80]

test_x = x[80:]
test_y = y[80:]

mymodel=np.poly1d(np.polyfit(train_x,train_y,4))
myline=np.linspace(0,6,100)
# prediction part how much a person will spent if he will stay in shop for 5 minutes
print(mymodel(5))
```

```
plt.plot(myline,mymodel(myline))
plt.scatter(train_x,train_y)
plt.show()
```

```
22.879625918115835
```



In [34]:
```
# R-squared score, The R-squared score is a good indicator of how well my data set is f
from sklearn.metrics import r2_score
r2 = r2_score(train_y, mymodel(train_x))

print(r2)
# the 0.79 r-squared score is good so relationship is ok

# same in case of testing data
r2 = r2_score(test_y, mymodel(test_x))

print(r2)
# now we are confident that we can use the model to predict future values.
```

```
0.7988645544629798
0.8086921460343579
```

In [35]:
```
# Predict Values
# for Example How much money will a buying customer spend, if she or he stays in the sh

# by use above data example

print(mymodel(5))
```

```
22.879625918115835
```

In [36]:
```
pip install pydotplus
```

```
Requirement already satisfied: pydotplus in c:\programdata\anaconda3\lib\site-packages
(2.0.2)Note: you may need to restart the kernel to use updated packages.
Requirement already satisfied: pyparsing>=2.0.1 in c:\programdata\anaconda3\lib\site-pac
kages (from pydotplus) (2.4.7)
```

In [37]:
```
# Machine Learning - Decision Tree
# A Decision Tree is a Flow Chart, and can help you make decisions based on previous ex
# importing modules
import sklearn
from sklearn import tree
import matplotlib.pyplot as plt
import pydotplus
```

```
from sklearn.tree import DecisionTreeClassifier
import matplotlib.image as pltimg
```

In [ ]:

In [38]:
```
pip install graphviz
```

```
Requirement already satisfied: graphviz in c:\programdata\anaconda3\lib\site-packages
(0.16)
Note: you may need to restart the kernel to use updated packages.
```

In [ ]:

In [39]:
```
x ={"age":[36,42,23,52,43,44,66,35,52,35,24,18,45],

       "exp":[10,12,4,4,21,14,3,14,13,5,3,3,9],

    "rank":[9,4,6,4,8,5,7,9,7,9,5,7,9],
      "nation":['UK','USA','N','USA','USA','UK','N','UK','N','N','USA','UK','UK'],
      "Go":['NO','NO','NO','NO','YES','NO','YES','YES','YES','YES','NO','YES','YES']
}
```

In [ ]:

In [40]:
```
newdf=pd.DataFrame(x)

# Change string values into numerical values nation and go
# pandas has map() method function for the same

d={'UK':0,'USA':1,'N':2}
newdf['nation']= newdf['nation'].map(d)
d={'YES':1,'NO':0}
newdf['Go']= newdf['Go'].map(d)

print(newdf)

# then we have to separate the feature columns and target columns
# feature coluns means from (columns) where we try to predict the vlue
# target column is the column with the values we try to predict.
# x is the feature column and y is the target column

features=['age','exp','rank','nation']
X=newdf[features]
y=newdf['Go']

print(X)
print(y)
```

```
    age  exp  rank  nation  Go
0    36   10     9       0   0
1    42   12     4       1   0
2    23    4     6       2   0
3    52    4     4       1   0
4    43   21     8       1   1
5    44   14     5       0   0
6    66    3     7       2   1
7    35   14     9       0   1
8    52   13     7       2   1
9    35    5     9       2   1
10   24    3     5       1   0
```

```
11   18    3     7       0   1
12   45    9     9       0   1
    age  exp  rank  nation
0    36   10     9       0
1    42   12     4       1
2    23    4     6       2
3    52    4     4       1
4    43   21     8       1
5    44   14     5       0
6    66    3     7       2
7    35   14     9       0
8    52   13     7       2
9    35    5     9       2
10   24    3     5       1
11   18    3     7       0
12   45    9     9       0
0     0
1     0
2     0
3     0
4     1
5     0
6     1
7     1
8     1
9     1
10    0
11    1
12    1
Name: Go, dtype: int64
```

In [41]:
```
from sklearn.tree import export_graphviz
```

In [42]:
```
pip install graphviz
```

```
Requirement already satisfied: graphviz in c:\programdata\anaconda3\lib\site-packages
(0.16)
Note: you may need to restart the kernel to use updated packages.
```

In [45]:
```
# now we can create the actual decision tree, fit it with our details, and save a .png
from sklearn.tree import export_graphviz



# dtree = DecisionTreeClassifier()
# dtree = dtree.fit(X,y)
# data = tree.export_graphviz(dtree, out_file=None, feature_names=features)
# graph = pydotplus.graph_from_dot_data(data)
# graph.write_png('mydecisiontree.png')

# img=pltimg.imread('mydecisiontree.png')
# imgplot=plt.imsow(img)
# plt.show()
```

In [47]:
```
# What would the answer be if the comedy rank was 6?

# print(dtree.predict([[40, 10, 6, 1]]))
```

In [ ]:
```

```

In [ ]:
```

```

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]: