Project 3

In addition to answering the bolded questions on Coursera, also attach your notebook, both as `.ipynb` and `.html`.

In the following exercise, we will perform linear regression to fit various data sets and to predict outputs. Perform the following analyses by starting a new notebook.

In this assignment, we will be using PennGrader, a Python package built by a former TA for autograding Python notebooks. PennGrader was developed to provide students with instant feedback on their answer. You can submit your answer and know whether it's right or wrong instantly. We then record your most recent answer in our backend database. You will have 100 attempts per test case, which should be more than sufficient.

NOTE：Please remember to remove the

```
    raise notImplementedError
```

after your implementation, otherwise the cell will not compile.

Getting Setup

Please run the below cells to get setup with the autograder. If you need to install packages, please uncomment and try the following lines; if they do not work, please try running them in the terminal without the `!` sign! (e.g. `pip install sklearn --user`

In [1]:
```
# %%capture
# !pip install penngrader --user
```

In [2]:
```
# !pip install seaborn --user
# !pip install sklearn --user
# !pip install statsmodels --user
```

Let's try PennGrader out! Fill in the cell below with your PennID and then run the following cell to initialize the grader.

Warning: Please make sure you only have one copy of the student notebook in your directory in Codio upon submission. The autograder looks for the variable `STUDENT_ID` across all notebooks, so if there is a duplicate notebook, it will fail.

In [3]:
```
#PLEASE ENSURE YOUR STUDENT_ID IS ENTERED AS AN INT (NOT A STRING). IF
 NOT, THE AUTOGRADER WON'T KNOW WHO
#TO ASSIGN POINTS TO YOU IN OUR BACKEND

STUDENT_ID = 49731093                   # YOUR 8-DIGIT PENNID GOES HERE
STUDENT_NAME = "Newman Alexander Ilgenfritz"     # YOUR FULL NAME GOES
 HERE
```

In [4]:
```
import penngrader.grader

grader = penngrader.grader.PennGrader(homework_id = 'ESE542_Online_Spri
ng_2021_HW3', student_id = STUDENT_ID)
```

In [5]:
```
# Let's import the relevant Python packages here
# Feel free to import any other packages for this project

# Data Wrangling
import pandas as pd
import numpy as np

# ML
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import PolynomialFeatures

# Statistics
import statsmodels.formula.api as smf
import statsmodels.api as sm
```

```
# Evaluation Metrics
from sklearn.metrics import mean_squared_error

# Plotting
import matplotlib.pyplot as plt

%matplotlib inline
```

Data Leakage

A very important (read: **the most important** topic) in practical data science scenarios is that of data leakage. Data leakage is a situation that occurs when the creator of a machine learning model allows the model to read both training data and test data to train the model. In Programming Project 3, the training data and the test data are separated for you already. Thus, the linear regression model should only be trained with the training data. Predictions can be made on either the training data or the test data. In the upcoming weeks, we will explore why you shouldn't train the model with the test data as well and what methods we can employ to choose the training set and the test set.

Part A

First, we will use the `Ch3PartA` dataset to generate polynomial regressions using `scikit-learn`. This dataset contains 100 observations of points x and their corresponding response, y. The data is divided into a training set (x_{tr}, y_{tr}) and a test set (x_{te}, y_{te}), and all the values are doubles.

A1.

To start, load `Ch3PartA.csv` into your notebook.

```
In [6]: Ch3PartAFile = pd.read_csv('Ch3PartA.csv').copy()
        Ch3PartA = pd.DataFrame(Ch3PartAFile)
```

A2.

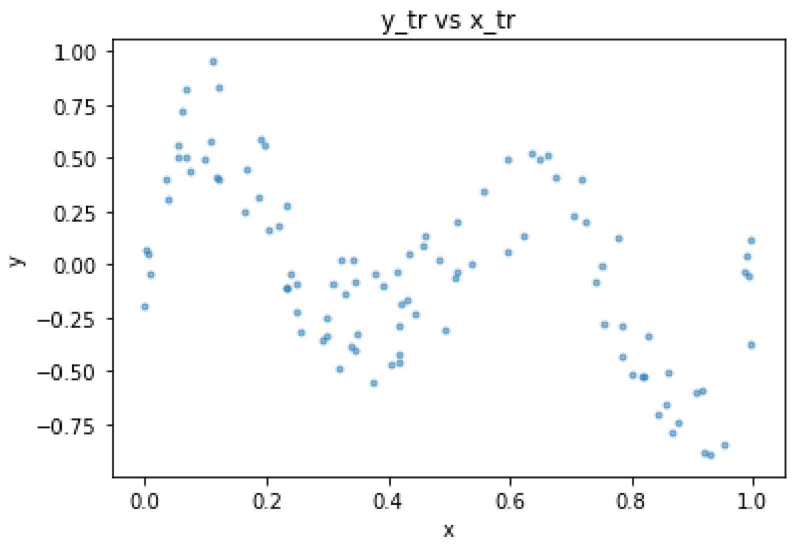
Create a scatter plot of:

(a) `y_tr` against `x_tr` and another of

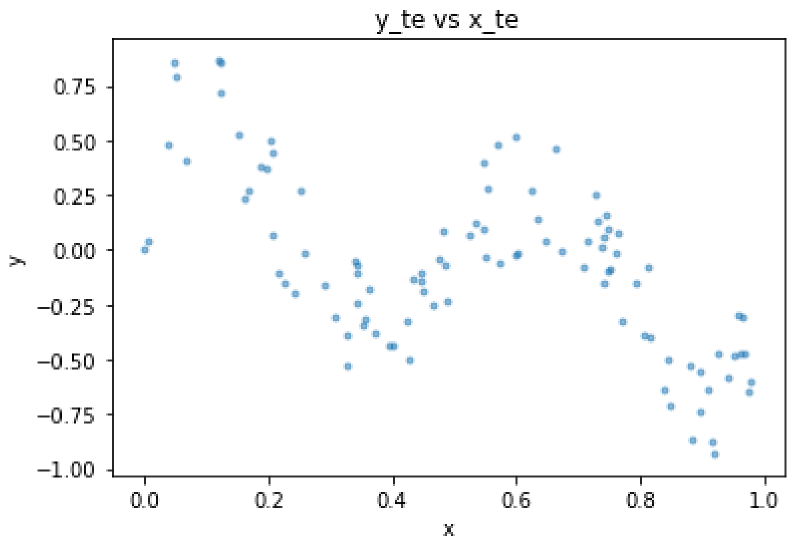
(b) `y_te` against `x_te`.

Then, observe and comment on the similarities and differences between the plots.

In [7]:
```
area = np.pi*3
plt.title('y_tr vs x_tr')
plt.xlabel('x')
plt.ylabel('y')
x =Ch3PartA["x_tr"]
y= Ch3PartA["y_tr"]
#plt.scatter(x, y, s=area, c=colors, alpha=0.5)
plt.scatter(x, y, s=area,  alpha=0.5)
plt.show()
plt.close()
```



In [8]:
```python
    #raise NotImplementedError
    # plot scatter of y_te vs x_te:
    #colors = (0,10,10)
    area = np.pi*3
    plt.title('y_te vs x_te')
    plt.xlabel('x')
    plt.ylabel('y')
    x = Ch3PartA["x_te"]
    y = Ch3PartA["y_te"]
    plt.scatter(x, y, s=area, alpha=0.5)
    plt.show()
    plt.close()
```



What is the maximum value of `y` in the training set and in the test set? Please store these variables as `max_y_train` and `max_y_test` below and run the first grader cell!

If you get 1 point, it means that you got both right. If you receive 0.5 points, you had only one right, and if you receive 0.25 points, then you correctly entered a tuple but both values were incorrect.

In [9]:
```python
max_y_train = (Ch3PartA["y_tr"].max())
```

```
max_y_test = (Ch3PartA["y_te"].max())
```

In [10]:
```
# View the results here before you submit
print(max_y_train, max_y_test)
```

```
0.95500282089616 0.8655959436070828
```

In [11]:
```
grader.grade(test_case_id = 'test_y_train_test', answer = (max_y_train,
max_y_test))
```

```
Correct! You earned 1.0/1 points. You are a star!

Your submission has been successfully recorded in the gradebook.
```

Now, comment on the plot differences below. Please record your response into the multiline string named `plot_diffs_string` and then submit it to us via the grader cell!

In [12]:
```
plot_diffs_string = '''
    train set has higher y values on average, and appears to have less
 dispersion around its mean
    then the test set.

'''
```

In [13]:
```
grader.grade(test_case_id = 'test_plot_diff_test', answer = plot_diffs_
string)
```

```
Correct! You earned 1/1 points. You are a star!

Your submission has been successfully recorded in the gradebook.
```

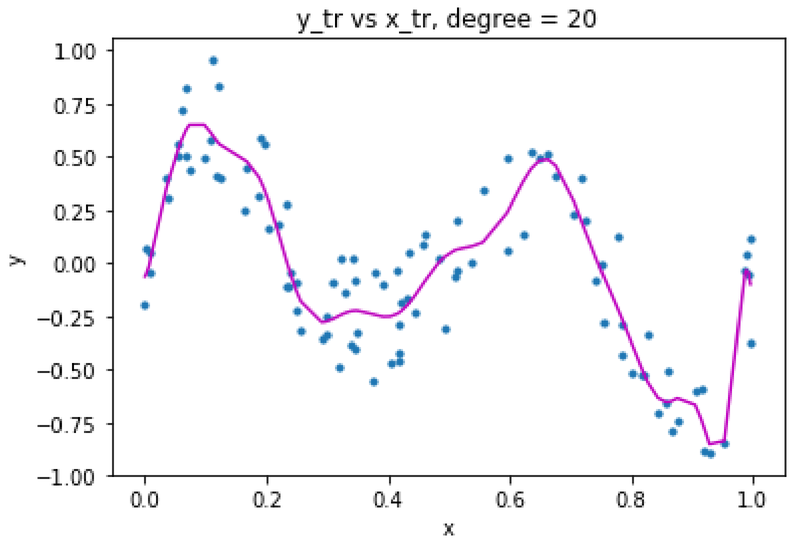
A3.

Generate the necessary features to fit polynomial regressions up to the 20th degree (up to and including the x_{20} term) on the training data. Hint: You will be fitting multi-variate linear regression models with polynomial features of x. Familiarize yourself with `sklearn.preprocessing.PolynomialFeatures`.

Here, we're just asking you to practice generating the features. You'll pass one of them into the autograder for a quick check (although the autograder will not be very strict, so if you end up failing the next test case definitely make sure your work here is correct!)

In [14]:
```
from sklearn import preprocessing
from sklearn.preprocessing import PolynomialFeatures
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score
import operator

x =Ch3PartA["x_tr"]
y= Ch3PartA["y_tr"]
x = x[:, np.newaxis]

y = y[:, np.newaxis]

deg = 20 # cx to 20 when working:
polyFeats = PolynomialFeatures(degree=deg)

polyX = polyFeats.fit_transform(x)
model = LinearRegression(fit_intercept = True)

model.fit(polyX, y)
y_poly_pred = model.predict(polyX)

rmse = np.sqrt(mean_squared_error(y,y_poly_pred))
r2 = r2_score(y,y_poly_pred)

plt.scatter(x, y, s=10)
# sort the values of x before line plot
sort_axis = operator.itemgetter(0)
sorted_zip = sorted(zip(x,y_poly_pred), key=sort_axis)
x, y_poly_pred = zip(*sorted_zip)
plt.plot(x, y_poly_pred, color='m')
title =  'y_tr vs x_tr, degree = ' + str(deg)
```

```
    plt.title(title)
    plt.xlabel('x')
    plt.ylabel('y')
    plt.show()
    plt.close()
```



Now, let's check to make sure your highest-degree polynomial features are correct; namely the set of features that includes x_{20}, or `PolynomialFeatures(degree = 20)`. Please set `polynomial_features_test_df` as this **dataframe**.

If you do not receive full points, that means that either you have the wrong number of columns or some column values aren't correct!

In [15]:
```
#polynomial_features_test_df = pd.DataFrame(PolynomialFeatures(degree=2
0).fit_transform(Ch3PartA["x_tr"]))
polynomial_features_test_df = pd.DataFrame(polyX)
```

In [16]:
```
grader.grade(test_case_id = 'test_poly_coefficients_setups', answer = p
olynomial_features_test_df)
```

```
Correct! You earned 1.0/1 points. You are a star!
```

```
Your submission has been successfully recorded in the gradebook.
```

A4.

Calculate the training MSE and the test MSE for 20 polynomial models up to degree 20. Store these as lists named `mse_train` and `mse_test` respectively. Hint: Familiarize yourself with the `sklearn.metrics.mean_squared_error` package and try to automate the process.

In [17]:

```
    """
    The issue is that your function calcRSMER2 takes in parameters x an
d y each time it is run.
    Thus, when you are calculating the test MSE, you are training your
 model on the test data.
    You can simply take out the function and use the code as-is.
    Just run model.fit() on the training data and run model.predict() o
n both the training data and test data.
    """


    def calcRMSER(runs, deg, x, y):
        rmseList = []
        r2List = []
        x = x[:, np.newaxis]
        y = y[:, np.newaxis]
        for i in range(runs):
          polyFeats = PolynomialFeatures(degree=deg)
          polyX = polyFeats.fit_transform(x)
          model = LinearRegression()
          model.fit(polyX, y)
          y_poly_pred = model.predict(polyX)
          #rmse = np.sqrt(mean_squared_error(y,y_poly_pred))
          rmse = (mean_squared_error(y, y_poly_pred))
          r2 = r2_score(y, y_poly_pred)
          #print('\n With polynomial degree ', deg, ' the performance p
arameters are: ')
          #print('    rmse: ', rmse)
          #print('    r2: ', r2, '\n')
```

```
            rmseList.append(rmse)
            r2List.append(r2)
            deg += 1
        #print('last degree used: ', str(deg-1))
        return rmseList, r2List

    def calcRMSER2(runs, deg, xTr, yTr, xTe, yTe):
        rmseList = []
        r2List = []
        xTr = xTr[:, np.newaxis]
        yTr = yTr[:, np.newaxis]
        xTe = xTe[:, np.newaxis]
        yTe = yTe[:, np.newaxis]
        for i in range(runs):
            polyFeats = PolynomialFeatures(degree=deg)
            polyXTr = polyFeats.fit_transform(xTr)
            polyXTe = polyFeats.fit_transform(xTe)
            model = LinearRegression()
            model.fit(polyXTr, yTr)
            y_poly_predTr = model.predict(polyXTr)
            y_poly_predTe = model.predict(polyXTe)
            #rmse = np.sqrt(mean_squared_error(y,y_poly_pred))
            rmse = (mean_squared_error(yTe, y_poly_predTe))
            r2 = r2_score(yTe, y_poly_predTe)
            #print('\n With polynomial degree ', deg, ' the performance p
arameters are: ')
            #print('    rmse: ', rmse)
            #print('    r2: ', r2, '\n')
            rmseList.append(rmse)
            r2List.append(r2)
            deg += 1
        #print('last degree used: ', str(deg-1))
        return rmseList, r2List

    mse_train = []
    mse_test = []
    xTr = Ch3PartA["x_tr"]
    yTr = Ch3PartA["y_tr"]
    xTe = Ch3PartA["x_te"]
    yTe = Ch3PartA["y_te"]
    runs = 20
    initDeg = 1
    deg = initDeg

    mse_train, r2TrainList = calcRMSER(runs, initDeg, xTr, yTr)
    print('\n mse_train: ')
    print( mse_train, '\n')
    #print('\nr2TrainList: ')
    #print(r2TrainList, '\n')

    mse_test, r2TestList = calcRMSER2(runs, initDeg, xTr, yTr, xTe, yTe
)
    print('\n mse_test: ')
    print( mse_test, '\n')
    #print('\nr2TrainList: ')
    #print(r2TestList, '\n')
    #raise NotImplementedError
```

```
 mse_train:
[0.12125185501925011, 0.12033129619890895, 0.11621071946843699, 0.11619
473366750084, 0.029527712553060435, 0.02940354220219511, 0.029316808566
589154, 0.029298072759654174, 0.029210396487502864, 0.028687978598350076
8, 0.028055308947000024, 0.027906972308051982, 0.027285261188199277, 0.0
27213013718522708, 0.026275691445044458, 0.026269671462221096, 0.0262693
09643829085, 0.025325292653535406, 0.025323673483406185, 0.024833781487
058403]


 mse_test:
[0.09818117822153999, 0.10019434223965008, 0.0867006932612044, 0.086872
44868018128, 0.03234864152603839, 0.0328270268142006, 0.032912940369083
75, 0.03288579488513143, 0.032847511775612175, 0.03374299946459762, 0.0
35538802394713756, 0.03542804470014921, 0.03735684810027347, 0.03775209
2209504026, 0.039213085157553264, 0.03945116797042994, 0.03947025182139
1314, 0.04263571410440089, 0.04270095983455204, 0.042976305958051075]
```

Run the grader cells for both `mse_train` and `mse_test` in order; please make sure you don't put the wrong cell in!

In [18]:
```
grader.grade(test_case_id = 'test_mse_polynomials_train', answer = mse_
train)
```

```
Correct! You earned 1.5/1.5 points. You are a star!

Your submission has been successfully recorded in the gradebook.
```

In [19]:
```
grader.grade(test_case_id = 'test_mse_polynomials_test', answer = mse_t
est)
```

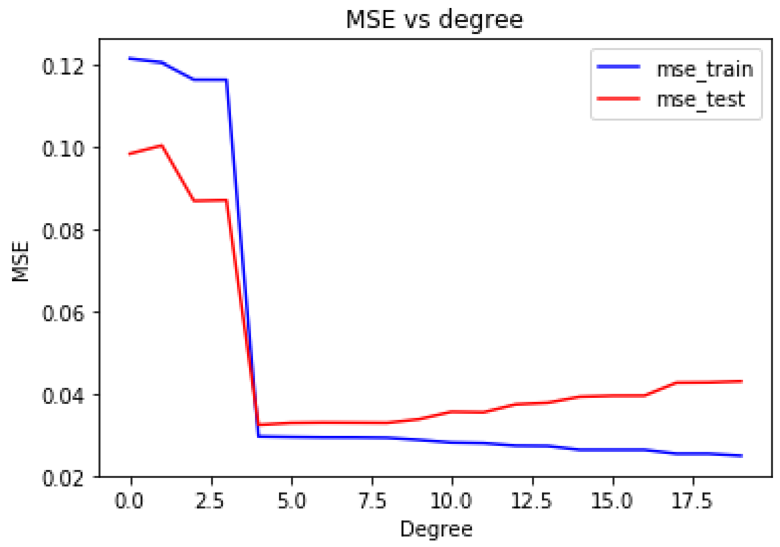
```
Correct! You earned 1.5/1.5 points. You are a star!

Your submission has been successfully recorded in the gradebook.
```

A5.

Generate a plot of both the training MSE and test MSE against flexibility (polynomial degree) for degrees 1 to 20.

Find the minimum training and testing MSEs and set them to `min_train_mse` and `min_test_mse` respectively.

In [20]:
```
    lastDeg = 20
    plotX = np.arange(lastDeg)
    y1 = np.asarray(mse_train)
    y2 = np.array(mse_test)
    plt.plot(plotX, y1, "-b", label="mse_train")
    plt.plot(plotX, y2, "-r", label="mse_test")
    #plt.plot(plotX, y1, y2)
    title =  'MSE vs degree'
    plt.title(title)
    plt.xlabel('Degree')
    plt.ylabel('MSE')
    plt.legend(loc="upper right")
    plt.show()
    plt.close()
    min_train_mse = min(mse_train)
    min_test_mse = min(mse_test)
    #raise NotImplementedError
```



In [21]:
```
print(min_train_mse, min_test_mse)
```

```
0.024833781487058403 0.03234864152603839
```

Hint: You should see the `min_train_mse < min_test_mse` since we have a bit of overfitting. Run the grader cell below! Each of the variables is worth 1 point; we assign points based on how close you are to the true answer

In [22]:
```
grader.grade(test_case_id = 'test_min_mses_test', answer = (min_train_m
se, min_test_mse))
```

```
Correct! You earned 2.0/2 points. You are a star!

Your submission has been successfully recorded in the gradebook.
```

A6.

From your plot, make an educated guess about the polynomial degree of the function that was used to generate the data. Then, give an estimate of the irreducible error $Var(\epsilon)$ for the optimal model on both the training set and test set.

Hint: The optimal model is obtained when we use the maximal degree polynomial that does not overfit. Revisit the section on hypothesis testing and think about the relationship between MSE, RSS, and RSE to calculate the irreducible error.

In [24]:
```
min_train_mseDeg = mse_train.index(min_train_mse) + 1
min_test_mseDeg = mse_test.index(min_test_mse) + 1
#print('min_train_mse: ', Demin_train_mseDeg, 'min_test_mse: ', Degmin_
test_mseDeg, '\n')
# is the degree asked for from the train set or test set?
# if from train set:
#degree = (min_train_mseDeg)
# if from test set:
degree = (min_test_mseDeg) # most likely this one
# residual sum of squares: model.ssr fails
# compute model results for the test data optimal degree = 5:
xTr = Ch3PartA["x_tr"]
yTr = Ch3PartA["y_tr"]
xTe = Ch3PartA["x_te"]
yTe = Ch3PartA["y_te"]
xTr = xTr[:, np.newaxis]
yTr = yTr[:, np.newaxis]
xTe = xTe[:, np.newaxis]
yTe = yTe[:, np.newaxis]
polyFeats = PolynomialFeatures(degree=5)
polyXTr = polyFeats.fit_transform(xTr)
polyXTe = polyFeats.fit_transform(xTe)
model = LinearRegression()
model.fit(polyXTr, yTr)
y_poly_predTr = model.predict(polyXTr)
y_poly_predTe = model.predict(polyXTe)
rss_adjTrain = np.sum((y - y_poly_predTr)**2)
rse = np.sqrt(rss_adjTrain/(len(y)-2))
```

```
RSE_train_sq = rse**2

#RSE_train_sq = (sum((np.sqrt(mse_train))))/(len(mse_train))

rss_adjTest = np.sum((y - y_poly_predTe)**2)
rse = np.sqrt(rss_adjTest/(len(y)-2))
RSE_test_sq =  rse**2
#RSE_test_sq = (sum((np.sqrt(mse_test))))/(len(mse_test))

#raise NotImplementedError
```

Please set your respective irreducible errors as `RSE_train_sq` and `RSE_test_sq` respectively, and set the number of polynomial features as `degree`. Then, run the grader cell below. It grades similar to above, but we add 1 point for the `degree` variable!

In [25]:
```
print("Desired degree for best model: ", degree)
print("Irreducible error (training): ", RSE_train_sq)
print("Irreducible error (test): ", RSE_test_sq)
```

```
Desired degree for best model:  5
Irreducible error (training):  0.030130318931694326
Irreducible error (test):  0.3087367893694604
```

In [26]:
```
grader.grade(test_case_id = 'test_irreducible', answer = (degree, RSE_t
rain_sq, RSE_test_sq))
```

```
You earned 1.75/3 points.

But, don't worry you can re-submit and we will keep only your latest sc
ore.
```

Part B

Next, we will use the `Ch3PartB` dataset to observe the effects of collinearity using `statsmodels`. This dataset contains 100 observations of points $(x1, x2)$, and y, the response variable.

B1.

Load the data from `Ch3PartB.csv` into a pandas DataFrame.

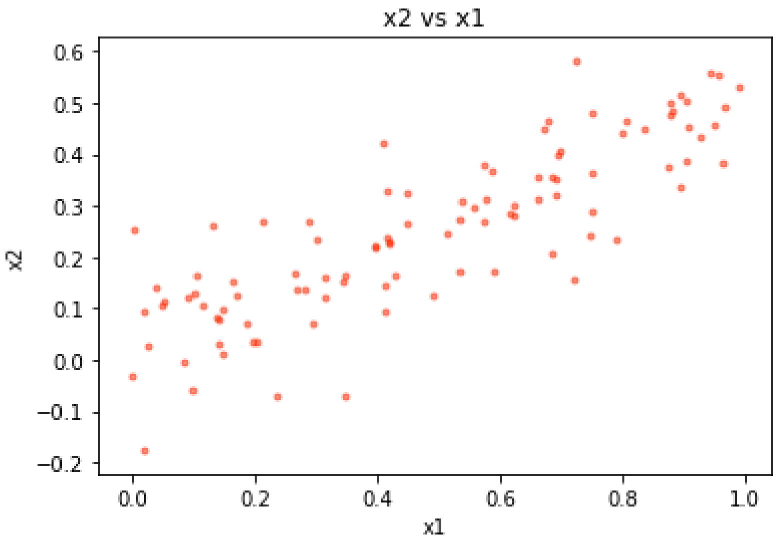
In [27]:
```
Ch3PartBFile = pd.read_csv('Ch3PartB.csv').copy()
#file_path = r'C:\Users\alexi\Desktop\Ch3PartA.csv'
#Ch3PartAfile = pd.read_csv(file_path)
Ch3PartB = pd.DataFrame(Ch3PartBFile)
```

B2.

Show a scatterplot displaying the relationship between $x1$ and $x2$

What is the correlation coefficient between $x1$ and $x2$? Compute the answer and store it as `correlation_variable` -- it should be a single floating-point number

In [28]:
```
    blue = '#008FD5'
    red = '#FF2700'
    green = '#77AB43'
    area = np.pi*3
    color =(red)
    plt.title('x2 vs x1')
    plt.xlabel('x1')
    plt.ylabel('x2')
    x = Ch3PartB["x1"]
    y = Ch3PartB["x2"]
    plt.scatter(x, y, s=area, c=color, alpha=0.5)
    plt.show()
    plt.close()
```



In [29]:
```python
r = np.corrcoef(x, y)
correlation_variable = (r[0, 1])
print('correlation_variable: ', correlation_variable)
```

```
correlation_variable:  0.8390596222844913
```

In [30]:
```python
grader.grade(test_case_id = 'test_correlation', answer = correlation_va
riable)
```

```
Correct! You earned 0.5/0.5 points. You are a star!

Your submission has been successfully recorded in the gradebook.
```

B3.

Using the data, fit a least squares regression to predict y using $x1$ and $x2$. Describe your results in a Markdown cell.

Hint: Familiarize yourself with `statsmodels.formula.api.ols`.

We have several questions here as well:

(a) What are the estimates $\hat{\beta}_0, \hat{\beta}_1, \hat{\beta}_2$?

(b) At a 95% confidence level, can you reject the null hypothesis $H_0 : \beta_1 = 0$?

(c) What about $H_0 : \beta_2 = 0$?

In [38]:
```
    #raise NotImplementedError
    # X = df[["RM", "LSTAT"]]
    X = Ch3PartB[["x1", "x2"]]
    X = sm.add_constant(X)
    y = Ch3PartB["y"]

    phrase = 'y ~ X'
    #results = smf.ols(y, X).fit()
    results = smf.ols(phrase, X).fit()

    RSS = np.sum(results.resid**2) #Residuals is y_i - \hat{y_i}

    RSE = np.sqrt(RSS/results.resid)

    TSS = np.sum((y-np.average(y))**2)
    Rsquared = (TSS-RSS)/TSS
```

```
/usr/local/lib/python3.6/dist-packages/numpy/core/fromnumeric.py:2389:
FutureWarning: Method .ptp is deprecated and will be removed in a futur
e version. Use numpy.ptp instead.
  return ptp(axis=axis, out=out, **kwargs)
/home/codio/.local/lib/python3.6/site-packages/pandas/core/series.py:85
6: RuntimeWarning: invalid value encountered in sqrt
  result = getattr(ufunc, method)(*inputs, **kwargs)
```

For your answers to $\hat{\beta}_0, \hat{\beta}_1, \hat{\beta}_2$, please input them either using code or typing the numbers from `statsmodels`' output into the variables below. You should include at least 4 digits after the decimal point.

In [39]:
```
beta_0 = (1.0946)
```

```
beta_1 = (0.7046)
beta_2 = (2.5024)
```

Please run but *do not change* the cell below to set up your autograder. Afterr, run the first grader cell with these variables! You will receive 0.5 points for each variable.

In [40]:
```
# DO NOT CHANGE THIS CELL!
answer_dict = {
    0: beta_0,
    1: beta_1,
    2: beta_2
}
```

In [41]:
```
grader.grade(test_case_id = 'test_betas', answer = answer_dict)
```

```
You earned 1.0/1.5 points.

But, don't worry you can re-submit and we will keep only your latest sc
ore.
```

Now, we want to evaluate the hypothesis test of $H_0 : \beta_1 = 0$ when both $x1$ and $x2$ are present.

Please compute either the t-value or p-value and set it as `test_statistic_b1`, then determine whether or not you reject the null hypothesiss at the 95% confidence level. Set that variable as a boolean (`True/False`) as `reject_null_b1`.

If your values are either incorrect or do not agree (i.e. you said the null would be rejected when it should not be), then you will not receive full points!

You will receive 0.5 points for getting the correct statistics as well as 1 points for your in-context evaluation of the null hypothesis.

In [46]:
```
test_statistic_b1 = (2.712146e-01) # use t or p, to at least 3 signific
ant digits
# 0.05 confidence level < p-value of 0.2712
```

```
reject_null_b1 = (False)
#raise NotImplementedError
```

In [47]:
```
grader.grade(test_case_id = 'test_beta1_hypothesis', answer = (test_sta
tistic_b1, reject_null_b1))
```

```
Correct! You earned 1.5/1.5 points. You are a star!

Your submission has been successfully recorded in the gradebook.
```

Let's do the same for the other test, to evaluate the hypothesis test of $H_0 : \beta_2 = 0$ when both $x1$ and $x2$ are present.

Please compute either the t-value or p-value and set it as `test_statistic_b2`, then determine whether or not you reject the null hypothesiss at the 95% confidence level. Set that variable as a boolean (`True/False`) as `reject_null_b2`

You will receive 0.5 points for getting the correct statistics as well as 1 point for your in-context evaluation of the null hypothesis.

In [48]:
```
test_statistic_b2 = (3.060418e-02) # use t or p, to at least 3 signific
ant digits
# 0.05 confidence level > p-value of 0.03060
reject_null_b2 = (True)
```

In [49]:
```
grader.grade(test_case_id = 'test_beta2_hypothesis', answer = (test_sta
tistic_b2, reject_null_b2))
```

```
Correct! You earned 1.5/1.5 points. You are a star!

Your submission has been successfully recorded in the gradebook.
```

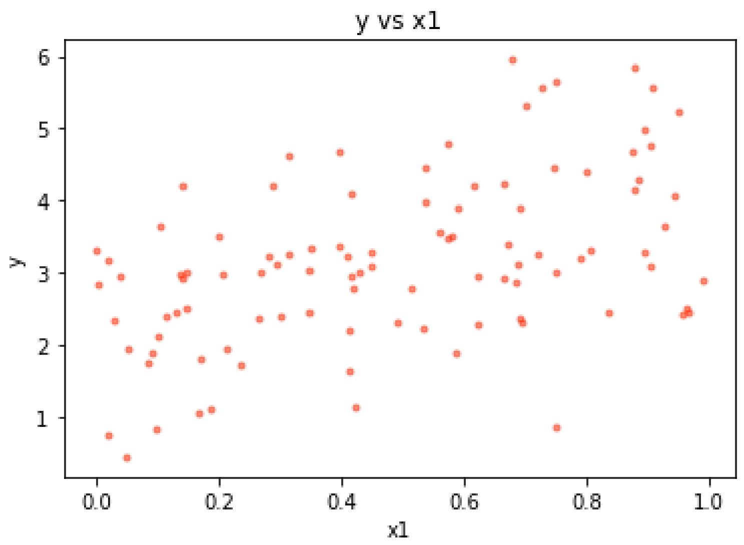
B4.

Now fit a least squares regression to predict y using only $x1$. Comment on your results.

Can you reject the null hypothesis $H_0 : \beta_1 = 0$?

In [55]:
```
    #raise NotImplementedError
    X = Ch3PartB["x1"]

    X = sm.add_constant(X)
    y = Ch3PartB["y"]

    blue = '#008FD5'
    red = '#FF2700'
    green = '#77AB43'
    area = np.pi*3
    color =(red)
    plt.title('y vs x1')
    plt.xlabel('x1')
    plt.ylabel('y')
    x = Ch3PartB["x1"]
    y = Ch3PartB["y"]
    plt.scatter(x, y, s=area, c=color, alpha=0.5)
    plt.show()
    plt.close()
    r = np.corrcoef(x, y)
    correlation_variable = (r[0, 1])
    print('correlation_variable: ', correlation_variable)


    phrase = 'y ~ X'
    #results = smf.ols(y, X).fit()
    results = smf.ols(phrase, X).fit()


    RSS = np.sum(results.resid**2) #Residuals is y_i - \hat{y_i}

    #RSE = np.sqrt(RSS/results.df_resid)
    RSE = np.sqrt(RSS/results.resid)

    #TSS = np.sum((data2['Grad_Rate']-np.average(data2['Grad_Rate']))**
2)
    TSS = np.sum((y-np.average(y))**2)
    Rsquared = (TSS-RSS)/TSS
```

```
/usr/local/lib/python3.6/dist-packages/numpy/core/fromnumeric.py:2389:
FutureWarning: Method .ptp is deprecated and will be removed in a futur
e version. Use numpy.ptp instead.
  return ptp(axis=axis, out=out, **kwargs)
```



```
correlation_variable:  0.4730772711650065
```

```
/home/codio/.local/lib/python3.6/site-packages/pandas/core/series.py:85
6: RuntimeWarning: invalid value encountered in sqrt
  result = getattr(ufunc, method)(*inputs, **kwargs)
```

When evaluating $H_0 : \beta_1 = 0$ when fitting with only $x1$ please do the following:

- Please compute either the t-value or p-value and set it as `test_statistic_b4`
- Determine whether or not you reject the null hypothesiss at the 95% confidence level; set that variable as a boolean (`True/False`) as `reject_null_b4`
- Determine if $x1$ is significant; set that result as a boolean (`True/False`) as `is_x1_significant`

Similar to previously, you'll receive points both for your test statistics and the evaluation.

In [56]:
```
test_statistic_b4 = (6.683125e-07) # use t or p, to at least 3 signific
ant digits
reject_b4_null_hypothesis = (True)
is_x1_significant = (True)
```

In [57]:
```
grader.grade(test_case_id = 'test_b4_x1_hypothesis', answer = (
    test_statistic_b4,
    reject_b4_null_hypothesis,
    is_x1_significant)
)
```

```
Correct! You earned 1.5/1.5 points. You are a star!

Your submission has been successfully recorded in the gradebook.
```

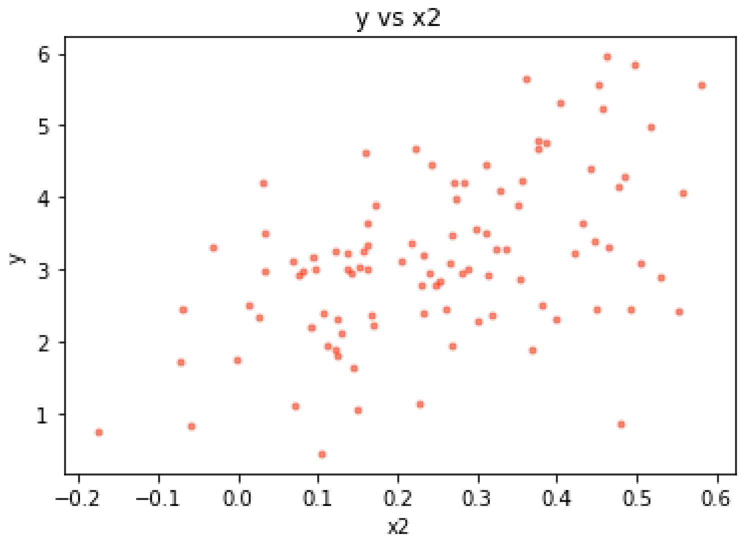
B5.

Now fit a least squares regression to predict y using only $x2$. Comment on your results.

Can you reject the null hypothesis $H_0 : \beta_2 = 0$?

In [50]:
```
    #raise NotImplementedError
    X = Ch3PartB["x2"]

    X = sm.add_constant(X)
    y = Ch3PartB["y"]

    blue = '#008FD5'
    red = '#FF2700'
    green = '#77AB43'
    area = np.pi*3
    color =(red)
    plt.title('y vs x2')
    plt.xlabel('x2')
    plt.ylabel('y')
    x = Ch3PartB["x2"]
    y = Ch3PartB["y"]
    plt.scatter(x, y, s=area, c=color, alpha=0.5)
    plt.show()
    plt.close()
    r = np.corrcoef(x, y)
    correlation_variable = (r[0, 1])
    print('correlation_variable: ', correlation_variable)

    #print('X: ')
    #print(X, '\n')
    #print('y: ')
    #print(y, '\n\n')
    phrase = 'y ~ X'
    #results = smf.ols(y, X).fit()
    results = smf.ols(phrase, X).fit()


    RSS = np.sum(results.resid**2) #Residuals is y_i - \hat{y_i}

    #RSE = np.sqrt(RSS/results.df_resid)
    RSE = np.sqrt(RSS/results.resid)

    #TSS = np.sum((data2['Grad_Rate']-np.average(data2['Grad_Rate']))**
2)
    TSS = np.sum((y-np.average(y))**2)
    Rsquared = (TSS-RSS)/TSS
```

```
/usr/local/lib/python3.6/dist-packages/numpy/core/fromnumeric.py:2389:
FutureWarning: Method .ptp is deprecated and will be removed in a futur
e version. Use numpy.ptp instead.
  return ptp(axis=axis, out=out, **kwargs)
```



```
correlation_variable:  0.5011715818873871
```

```
/home/codio/.local/lib/python3.6/site-packages/pandas/core/series.py:85
6: RuntimeWarning: invalid value encountered in sqrt
  result = getattr(ufunc, method)(*inputs, **kwargs)
```

When evaluating $H_0 : \beta_2 = 0$ when fitting with only $x1$ please do the following:

- Please compute either the t-value or p-value and set it as `test_statistic_b5`
- Determine whether or not you reject the null hypothesiss at the 95% confidence level; set that variable as a boolean (`True/False`) as `reject_null_b5`
- Determine if $x2$ is significant; set that result as a boolean (`True/False`) as `is_x2_significant`

Scoring is identical to B4 above!

In [51]:
```
test_statistic_b5 = 1.09e-07 # use t or p, to at least 3 significant di
gits
```

```
reject_b5_null_hypothesis = True
is_x2_significant = True
```

In [52]:
```
grader.grade(test_case_id = 'test_b5_x2_hypothesis', answer = (
    test_statistic_b5,
    reject_b5_null_hypothesis,
    is_x2_significant)
)
```

```
Correct! You earned 1.5/1.5 points. You are a star!

Your submission has been successfully recorded in the gradebook.
```

B6.

Do Part B Questions 3-5 contradict each other? Explain why or why not.

In [53]:
```
#Yes, the results of Q 3 and the results of Qs 4 and 5 seem contradicto
ry,
# because while we can confidently reject null hypothesis when relying
 upon x1 or x2 alone,
# we get mixed results when using both x1 and x2 together.
# You would think using two independent vars would explain more of the
 variance,
# but in this case an examination of the plots indicates that the patte
rns of x1 and x2
# are both sufficiently noisy that combining them does not reduce varia
nce but increases it
```

Now, comment on the apparent contradiction below. Enter a boolean (`True/False`) for whether or not the answers contradict as `is_contradiction` , and then record your explanation into the multiline string named `contradiction_string` and then submit it to us via the grader cell!

Please note that you'll need to have the right answer as well as have an explanation that has a reasonable set of keywords in order to get full credit!

Note: if you have an explanation that you think is reasonable but you aren't passing the autograder, let us know on Piazza!

In [54]:
```
is_contradiction = (True)
contradiction_string = '''
       Yes, the results of Q 3 and the results of Qs 4 and 5 seem contr
adictory,
because while we can confidently reject null hypothesis when relying up
on x1 or x2 alone,
we get mixed results when using both x1 and x2 together.
You would think using two independent vars would explain more of the va
riance,
but in this case an examination of the plots indicates that the pattern
s of x1 and x2
are both sufficiently noisy that combining them does not reduce varianc
e but increases it
'''
#raise NotImplementedError
```

In [55]:
```
grader.grade(test_case_id = 'test_contradiction_test', answer = (is_con
tradiction, contradiction_string))
```

```
Correct! You earned 1/1 points. You are a star!

Your submission has been successfully recorded in the gradebook.
```

Submit

You're done! Please make sure you've run all the PennGrader cells and count up your score to be sure (there are 20 points in total) and then make sure to submit this on Codio.