5-Copy1

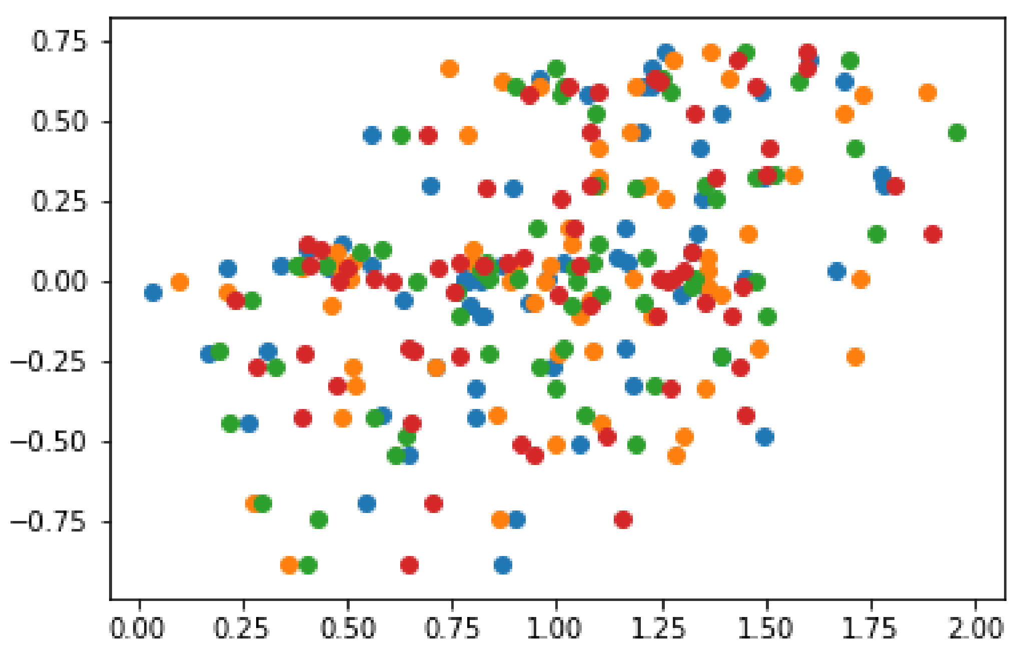
October 7, 2018

```
In [1]: ## Please use the Github output.
        ## My LaTex pdf graph outputs are mixing up cache or something
```

```
In [2]: from sklearn.linear_model import LinearRegression
        import numpy as np
        import matplotlib.pylab as plt
        %matplotlib inline
        from sklearn import linear_model
```

```
In [3]: n = 10000
        x = np.linspace(0.01, 1, n).reshape(-1, 1)
        y = np.linspace(0.01, 1, n)  + np.random.rand(n) - .5

        plt.scatter(x,y)
```

```
Out[3]: <matplotlib.collections.PathCollection at 0x1a12efcda0>
```



1 Assignment 5

1.1 1. Create and fit a Linear Regression Model

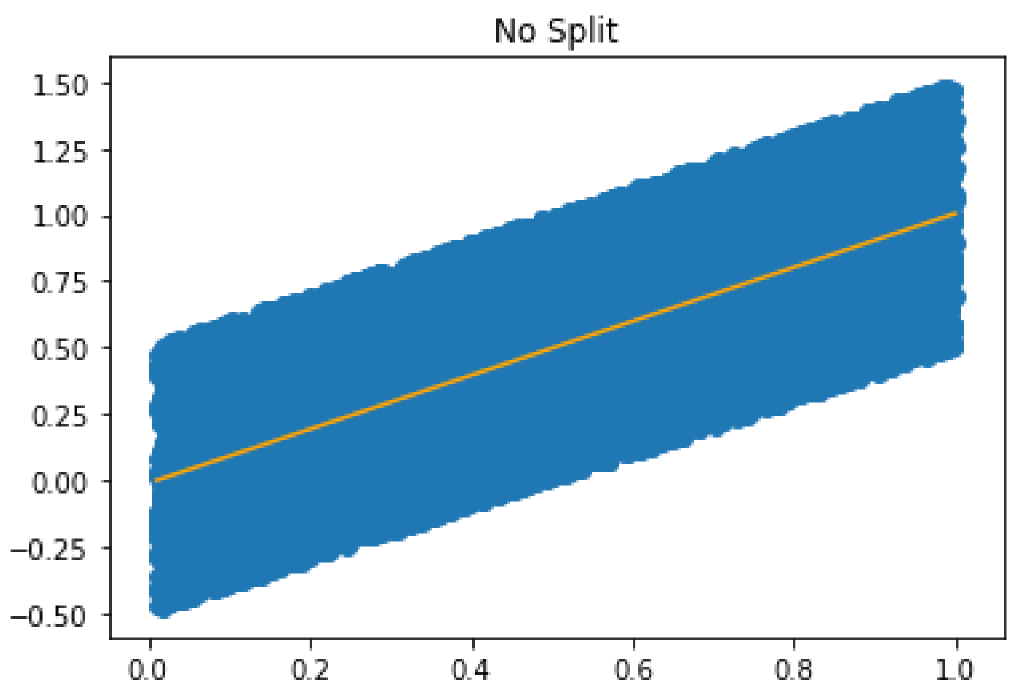
1.2 Calculate the Training error and Testing error using sklearn with a .50 split

For error, use `mean_squared`, but if you want to experiment with other mean errors, please do!

```
In [4]: # First, display no split at all
        from IPython.display import display

        model = LinearRegression()
        model.fit(x, y)
        nosplit = model.coef_, model.intercept_
        print("Fit:" , model.coef_, model.intercept_)

        plt.scatter(x,y)
        plt.plot(x, np.dot(x, model.coef_) + model.intercept_, c="orange")
        plt.title("No Split")
```

```
Fit: [1.01634993] -0.011496060745088466
```

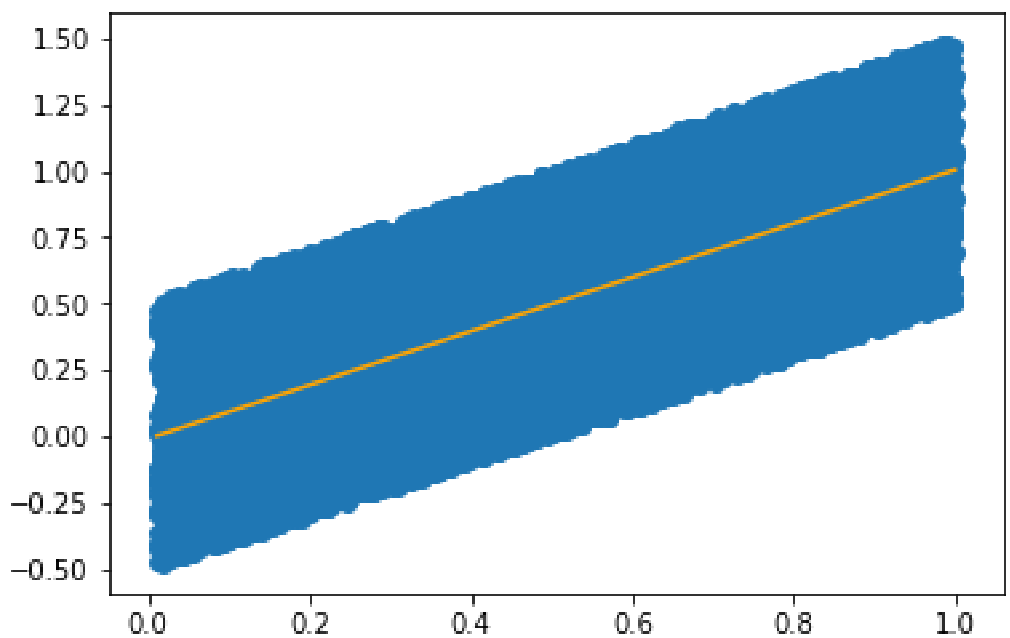
```
Out[4]: Text(0.5,1,'No Split')
```




```
In [5]: from sklearn.model_selection import train_test_split
        from sklearn.metrics import mean_squared_error
```

```
print("slope N y intercept:", model.coef_, model.intercept_)
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=.5)
model = LinearRegression()
model.fit(x_train, y_train)

plt.scatter(x,y)
plt.plot(x, np.dot(x, model.coef_) + model.intercept_, c="orange")
print("")
print("MSE test set: ", mean_squared_error(y_test, np.dot(x_test, model.coef_) + model.i
print("MSE train set:", mean_squared_error(y_train, np.dot(x_train, model.coef_) + model

print("")
print("Test set and train set error values go up and down variably such that one is high
```

```
slope N y intercept: [1.01634993] -0.011496060745088466

MSE test set:  0.08411920591890769
MSE train set: 0.08484675848898185

Test set and train set error values go up and down variably such that one is higher over the oth
```




```
In [6]: print("Of course, the lengthy, thought-provoking methods:")

        # Dissect halves mehtod
```

```
        model = LinearRegression()
        model.fit(x[:5000], y[:5000])
        print("First half:", model.coef_, model.intercept_)

        # Unsurprisingly, it looks about the same without the split
        print("w/o split: ", nosplit)
        # ...which means y is not negative 1 at x with nearly 1:1 slope"



        # Shuffle method
        print("")
        print("")
        def shuffle(a, b):
            assert len(a) == len(b)
            p = np.random.permutation(len(a))
            return p
        p = shuffle(x, y)
        print("Let's see it shuffled: ", p)
        model = LinearRegression()
        model.fit(x[p][:5000], y[p][:5000])
        print("For first half:", model.coef_, model.intercept_)
        print("training:      ", np.sum(np.square(y[p][:5000] - (np.dot(x[p][:5000], model.coef_
        model.fit(x[p][5000:], y[p][5000:])
        print("test:          ", np.sum(np.square(y[p][5000:] - (np.dot(x[p][5000:], model.coef_
```

```
Of course, the lengthy, thought-provoking methods:
First half: [0.99467504] -0.007533522564980383
w/o split:  (array([1.01634993]), -0.011496060745088466)


Let's see it shuffled:  [4750 1448 3279 ...  423 3847 5634]
For first half: [1.01599514] -0.014852174695210552
training:       0.0856303720129576
test:           0.08330515689032791
```

```
In [7]: # Cool. Everything looks about the same. Probably because of the
        # dense data
```

1.3 2. Repeat #1 for a Ridge Regression

```
In [8]: from sklearn.linear_model import Ridge

        linear_model = Ridge()
        model.fit(x_train, y_train)
        model.coef_, model.intercept_
```

```
print("training set:", mean_squared_error(y_train, np.dot(x_train, model.coef_) + model.
print("test set:    ", mean_squared_error(y_test, np.dot(x_test, model.coef_) + model.in
```

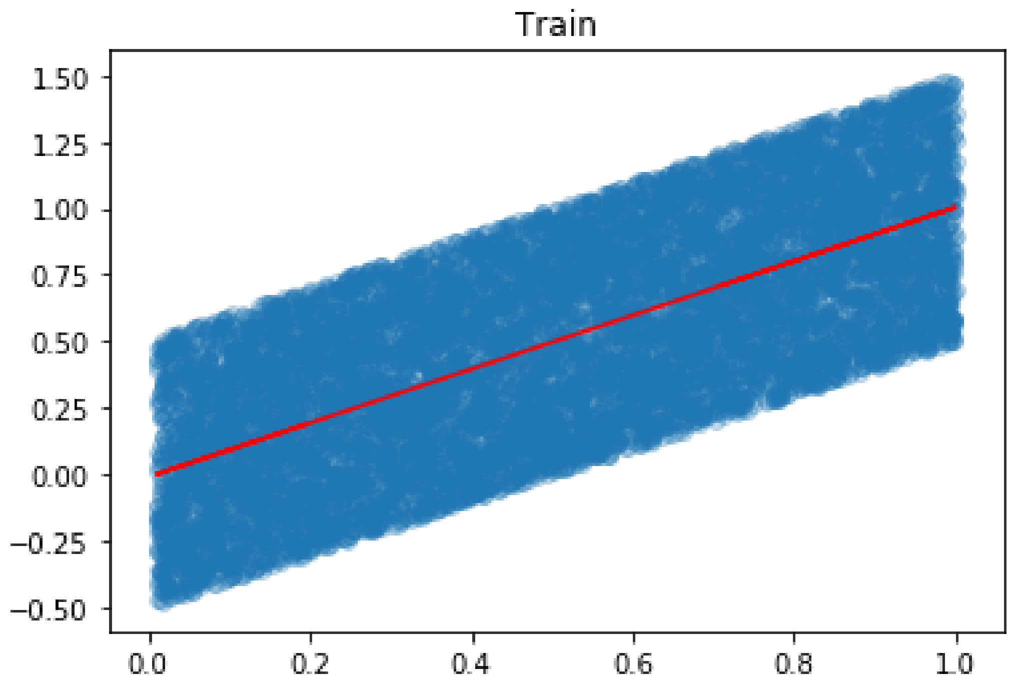
```
training set: 0.08484675848898185
test set:     0.08411920591890769
```

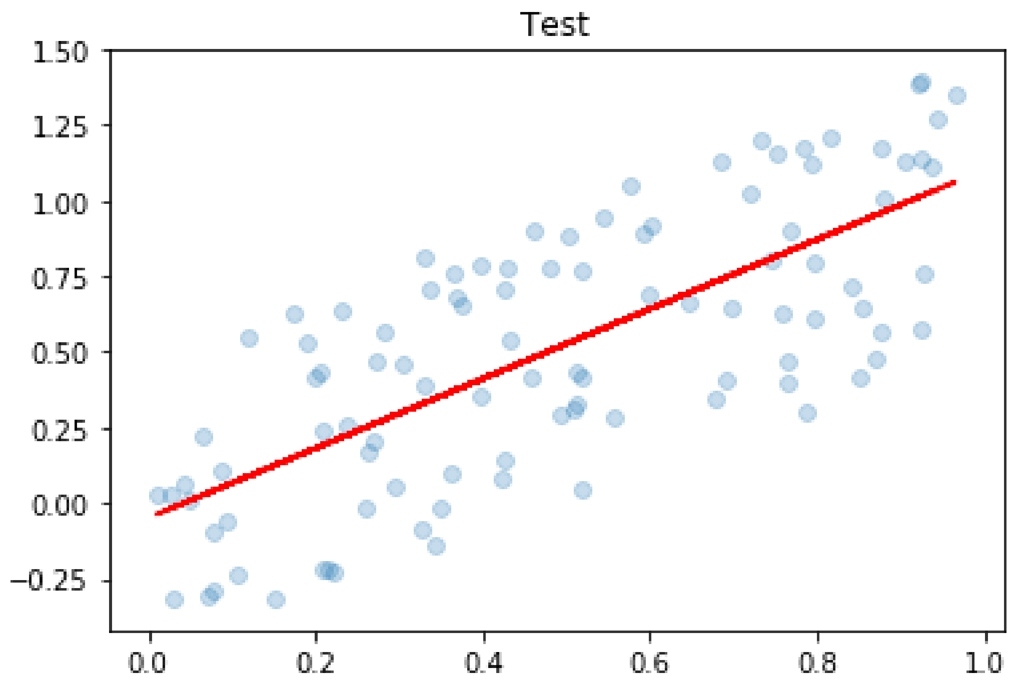
In [9]:
```
# considering the dense data, this makes sense
```

1.4 3. Vary the `split` size from .01 to .99 with at least 10 values (the more the merrier!). Plot the resulting Training error and Testing error vs. `split` size. Create separate plots for Linear and Ridge

In [10]:
```
model = LinearRegression()
model.fit(x_train, y_train)

x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=.01)
print("Train .01: ", mean_squared_error(y_train, np.dot(x_train, model.coef_) + model.i
print("Test .01:  ", mean_squared_error(y_test, np.dot(x_test, model.coef_) + model.int
model.fit(x_train, y_train)
model.coef_, model.intercept_
plt.scatter(x_train,y_train, alpha=.25)
plt.plot(x_train, np.dot(x_train, model.coef_) + model.intercept_, c="red")
plt.title("Train")
plt.show()

model.fit(x_test, y_test)
model.coef_, model.intercept_
plt.scatter(x_test,y_test, alpha=.25)
plt.plot(x_test, np.dot(x_test, model.coef_) + model.intercept_, c="red")
plt.title("Test")
plt.show()
```

```
Train .01:  0.0844056000633266
Test .01:   0.09214381412514278
```

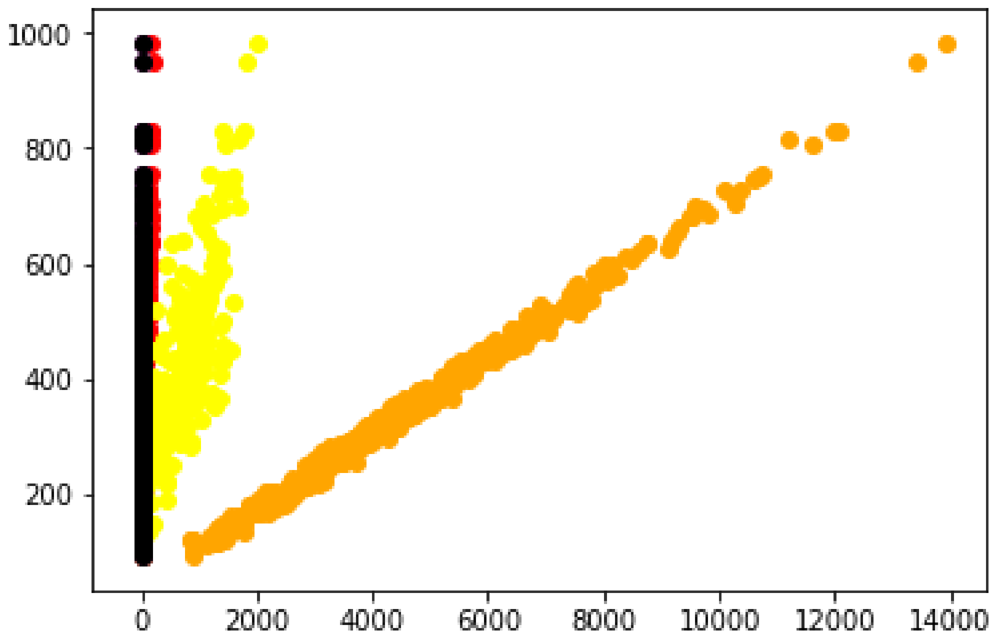


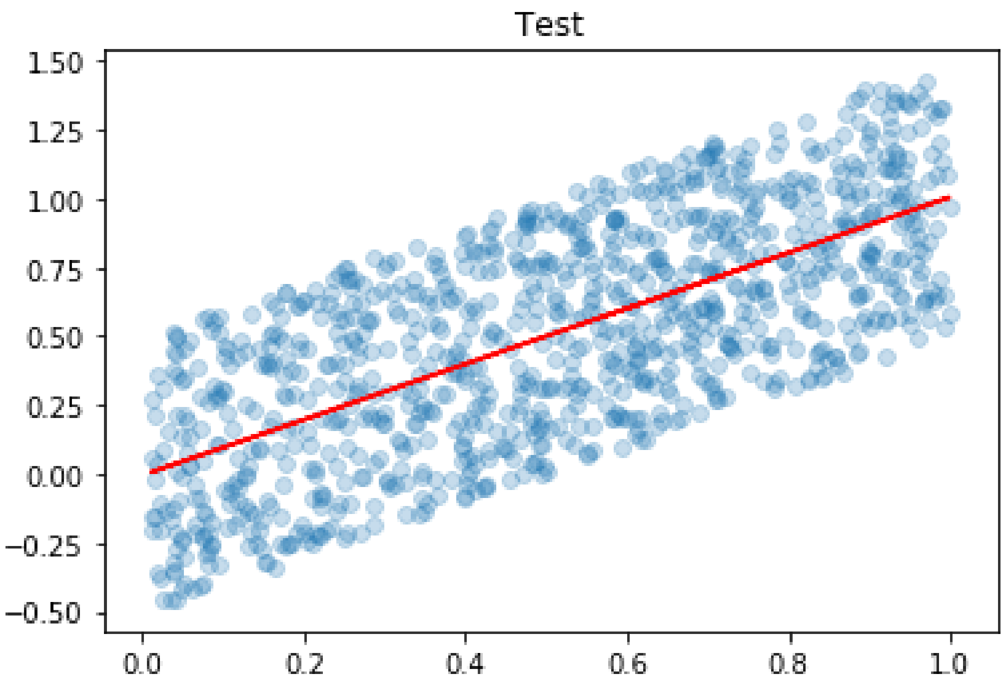



```
In [11]: x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=.1)
         print("Train .1: ", mean_squared_error(y_train, np.dot(x_train, model.coef_) + model.in
```

```
print("Test .1:  ", mean_squared_error(y_test, np.dot(x_test, model.coef_) + model.inte
model.fit(x_train, y_train)
model.coef_, model.intercept_
plt.scatter(x_train,y_train, alpha=.25)
plt.plot(x_train, np.dot(x_train, model.coef_) + model.intercept_, c="red")
plt.title("Train")
plt.show()

model.fit(x_test, y_test)
model.coef_, model.intercept_
plt.scatter(x_test,y_test, alpha=.25)
plt.plot(x_test, np.dot(x_test, model.coef_) + model.intercept_, c="red")
plt.title("Test")
plt.show()
print("")
```

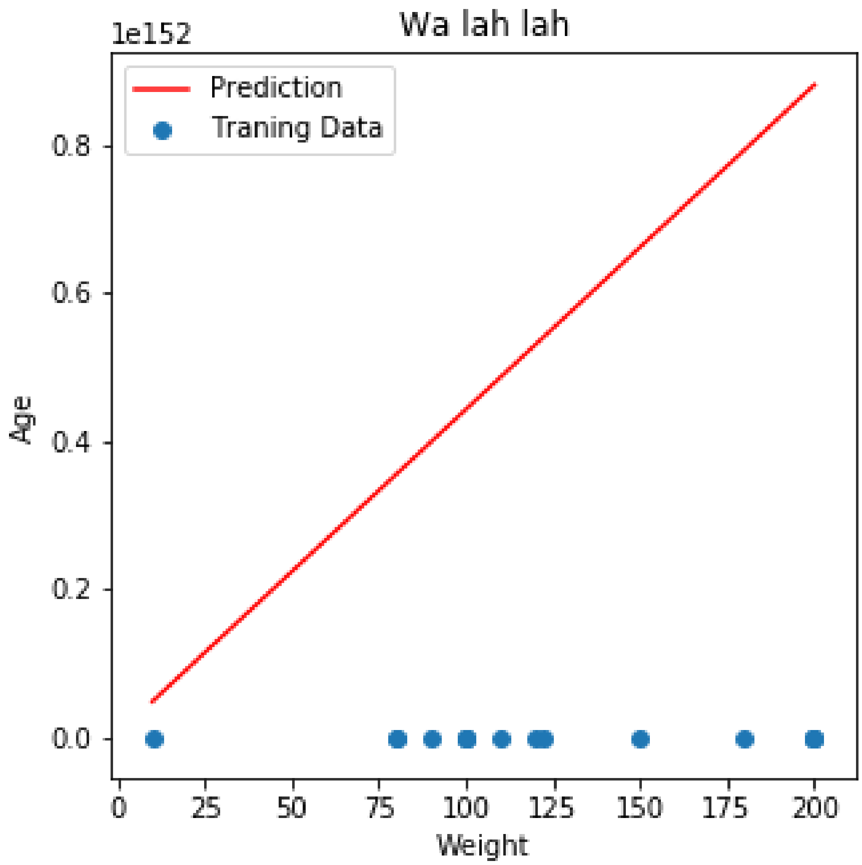
```
Train .1:   0.08674555454991946
Test .1:   0.08798800353527304
```

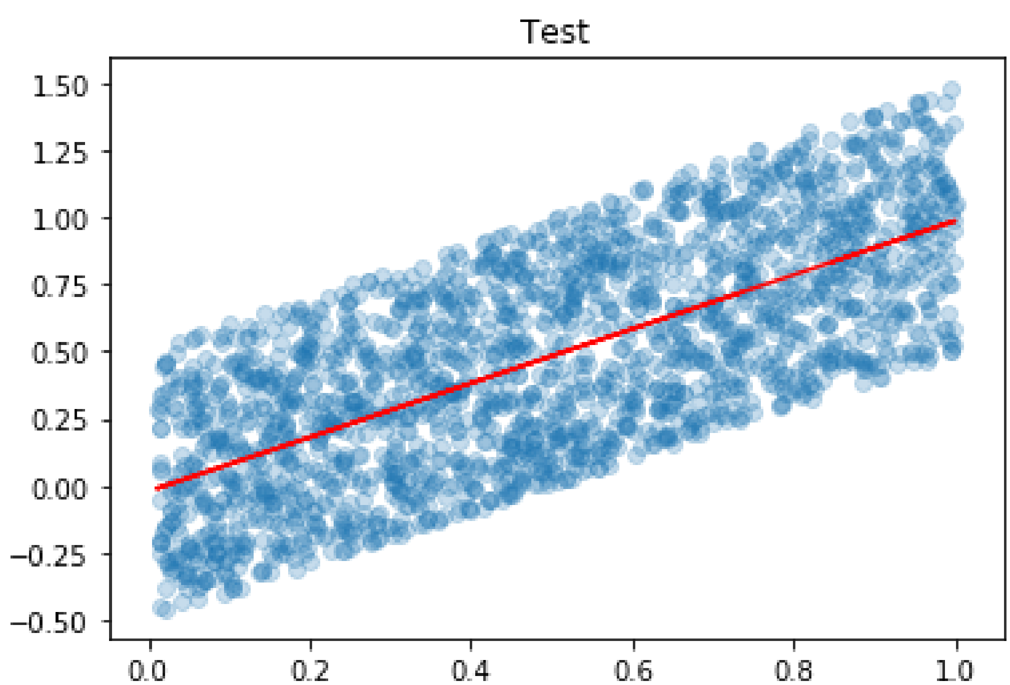





```
In [12]: x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=.2)
         print("Train .2: ", mean_squared_error(y_train, np.dot(x_train, model.coef_) + model.in
         print("Test .2:  ", mean_squared_error(y_test, np.dot(x_test, model.coef_) + model.inte
         model.fit(x_train, y_train)
         model.coef_, model.intercept_
         plt.scatter(x_train,y_train, alpha=.25)
         plt.plot(x_train, np.dot(x_train, model.coef_) + model.intercept_, c="red")
         plt.title("Train")
         plt.show()

         model.fit(x_test, y_test)
         model.coef_, model.intercept_
         plt.scatter(x_test,y_test, alpha=.25)
         plt.plot(x_test, np.dot(x_test, model.coef_) + model.intercept_, c="red")
         plt.title("Test")
         plt.show()
```

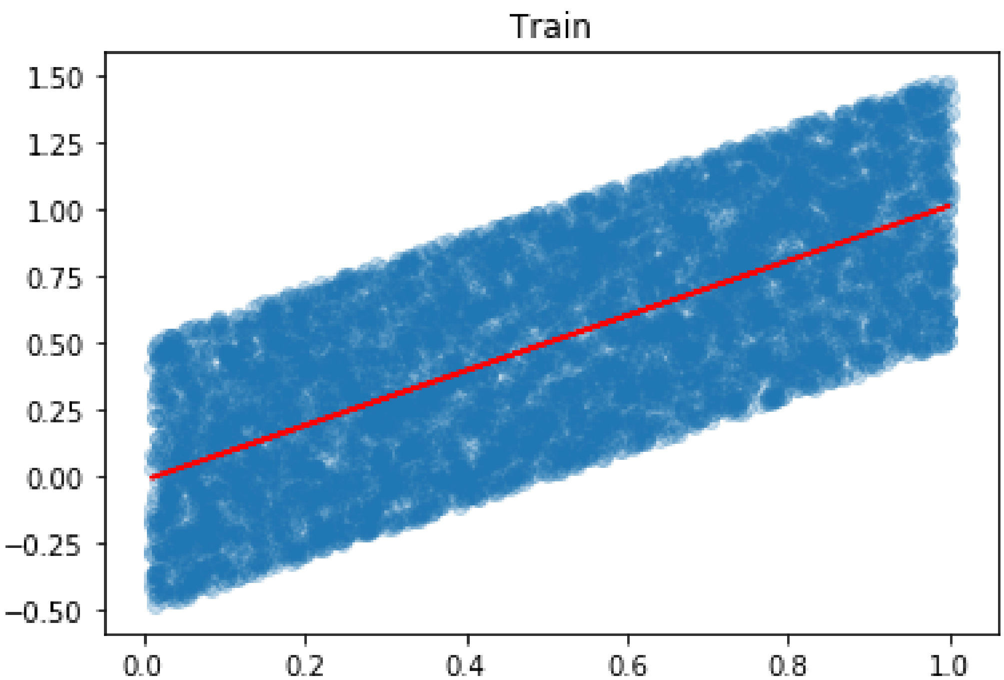
```
Train .2:  0.08480825126374282
Test .2:   0.08325645377811006
```

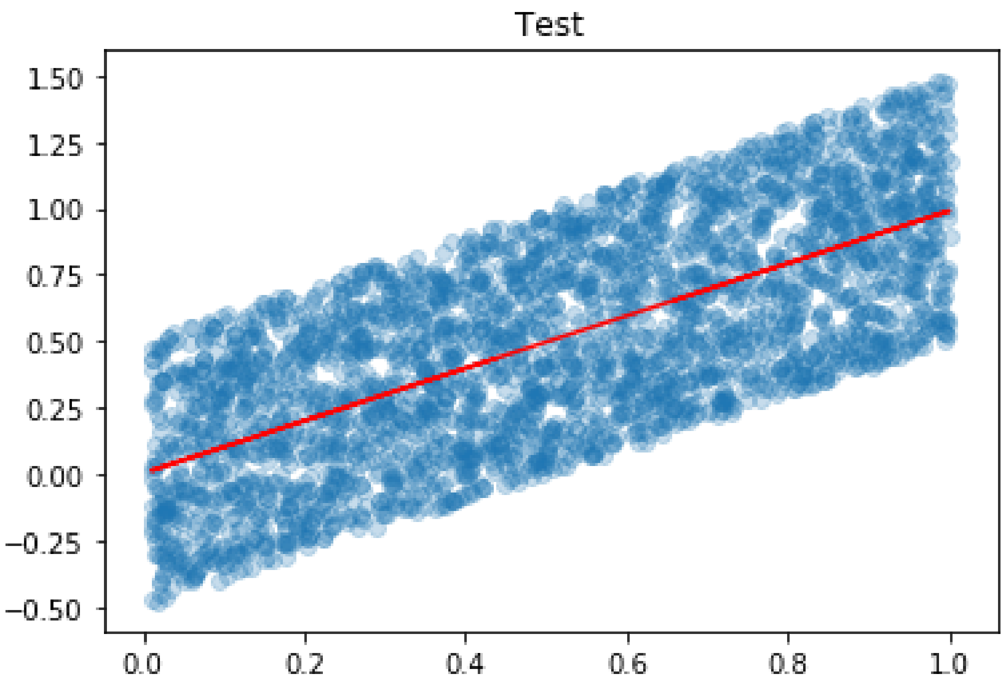

Wa lah lah
1e152
Prediction
Traning Data
0.8
0.6
0.4
0.2
0.0
Age
0
25
50
75
100
125
150
175
200
Weight


Test
1.50
1.25
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
0.0
0.2
0.4
0.6
0.8
1.0

In [13]:
```
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=.3)
print("Train .3: ", mean_squared_error(y_train, np.dot(x_train, model.coef_) + model.in
print("Test .3:  ", mean_squared_error(y_test, np.dot(x_test, model.coef_) + model.inte
model.fit(x_train, y_train)
model.coef_, model.intercept_
plt.scatter(x_train,y_train, alpha=.25)
plt.plot(x_train, np.dot(x_train, model.coef_) + model.intercept_, c="red")
plt.title("Train")
plt.show()

model.fit(x_test, y_test)
model.coef_, model.intercept_
plt.scatter(x_test,y_test, alpha=.25)
plt.plot(x_test, np.dot(x_test, model.coef_) + model.intercept_, c="red")
plt.title("Test")
plt.show()
```

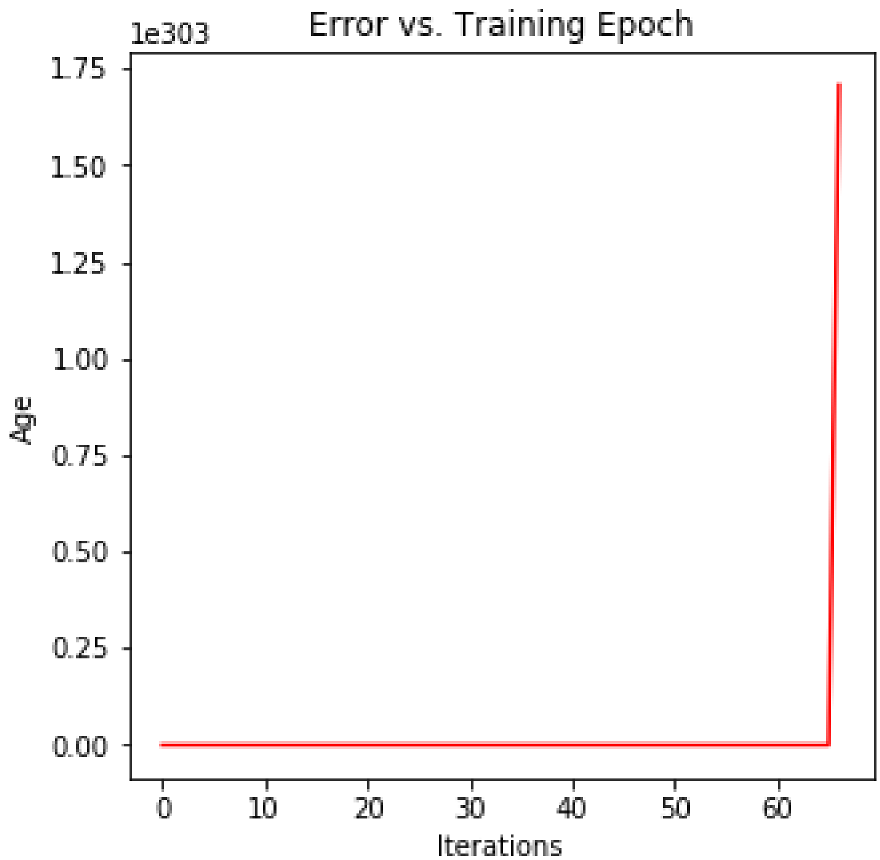
```
Train .3:  0.08420534211880233
Test .3:   0.08576553151299592
```



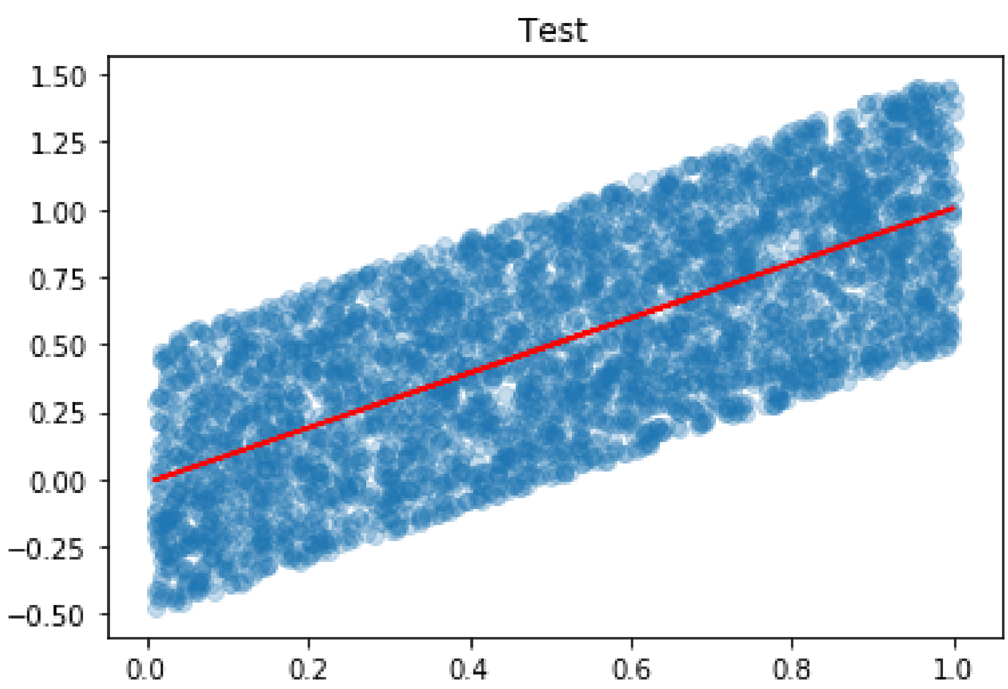



```
In [14]: x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=.4)
         print("Train .4: ", mean_squared_error(y_train, np.dot(x_train, model.coef_) + model.in
         print("Test .4:  ", mean_squared_error(y_test, np.dot(x_test, model.coef_) + model.inte
         model.fit(x_train, y_train)
         model.coef_, model.intercept_
         plt.scatter(x_train,y_train, alpha=.25)
         plt.plot(x_train, np.dot(x_train, model.coef_) + model.intercept_, c="red")
         plt.title("Train")
         plt.show()

         model.fit(x_test, y_test)
         model.coef_, model.intercept_
         plt.scatter(x_test,y_test, alpha=.25)
         plt.plot(x_test, np.dot(x_test, model.coef_) + model.intercept_, c="red")
         plt.title("Test")
         plt.show()
```

```
Train .4:  0.08475356274018869
Test .4:   0.08422938947966487
```



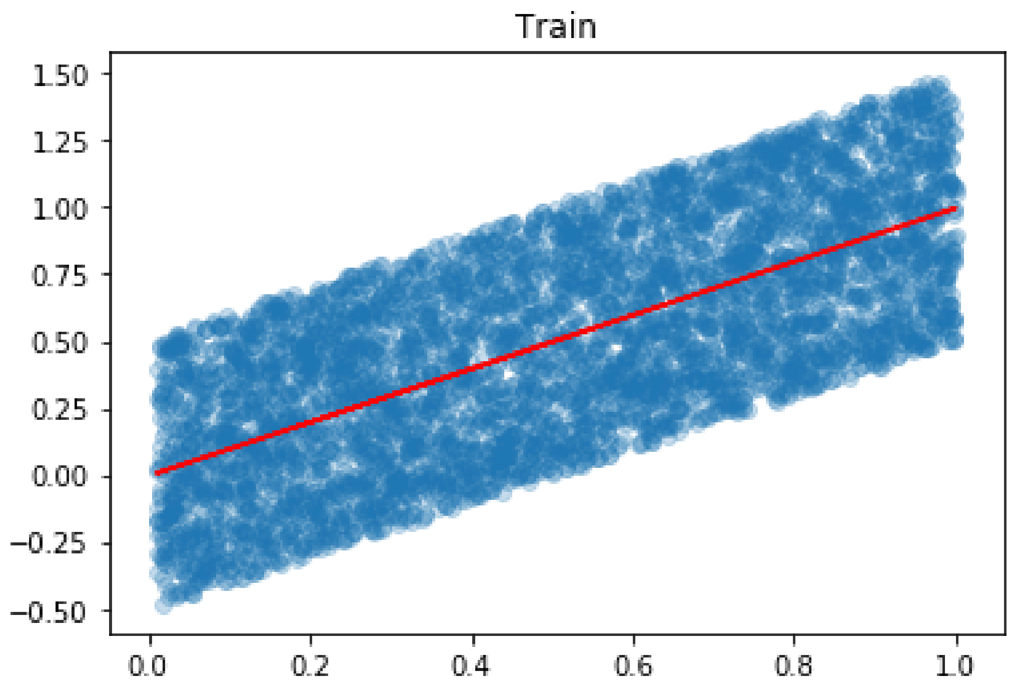
Error vs. Training Epoch
1e303
1.75
1.50
1.25
1.00
0.75
0.50
0.25
0.00
Age
0
10
20
30
40
50
60
Iterations

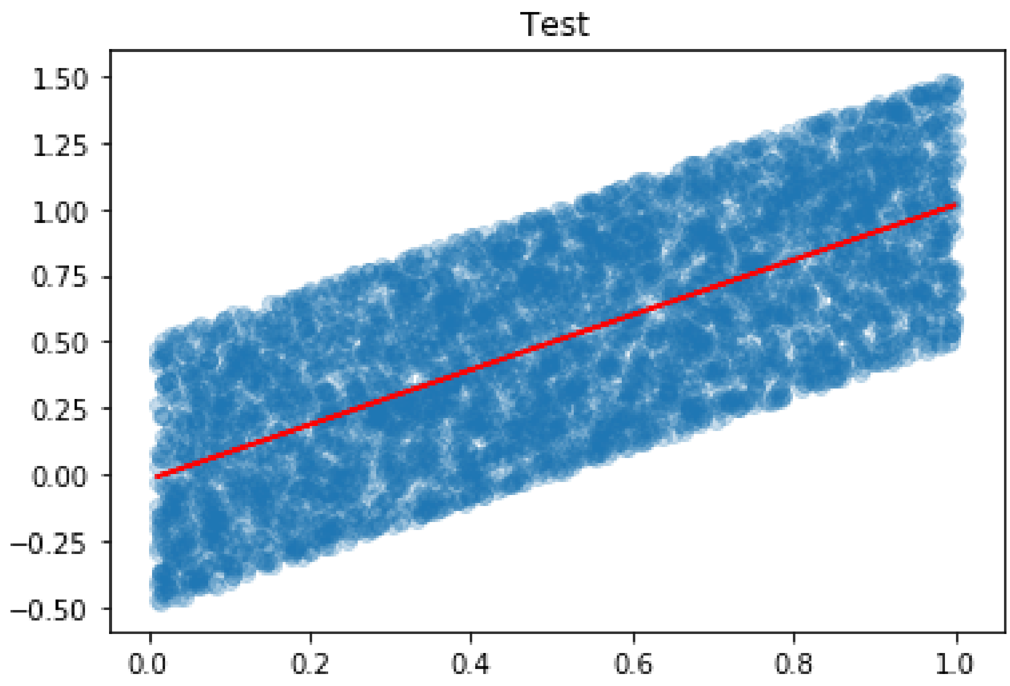


Test
1.50
1.25
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
0.0
0.2
0.4
0.6
0.8
1.0

```
In [15]: x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=.5)
         print("Train .5: ", mean_squared_error(y_train, np.dot(x_train, model.coef_) + model.in
         print("Test .5:  ", mean_squared_error(y_test, np.dot(x_test, model.coef_) + model.inte
         model.fit(x_train, y_train)
         model.coef_, model.intercept_
         plt.scatter(x_train,y_train, alpha=.25)
         plt.plot(x_train, np.dot(x_train, model.coef_) + model.intercept_, c="red")
         plt.title("Train")
         plt.show()

         model.fit(x_test, y_test)
         model.coef_, model.intercept_
         plt.scatter(x_test,y_test, alpha=.25)
         plt.plot(x_test, np.dot(x_test, model.coef_) + model.intercept_, c="red")
         plt.title("Test")
         plt.show()
```

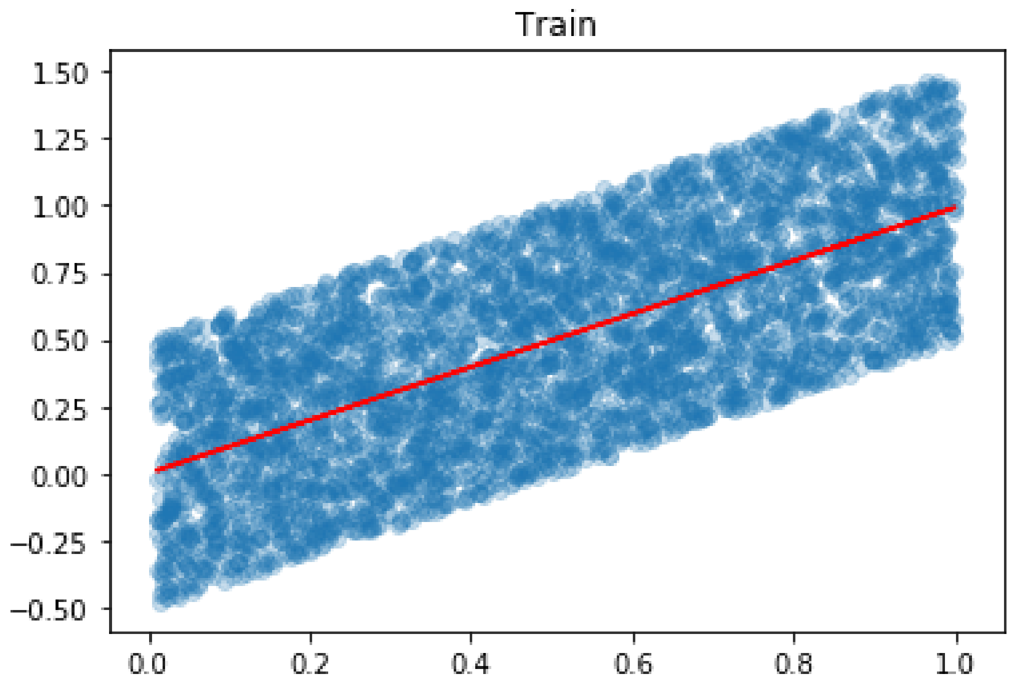
```
Train .5:  0.08534025901490036
Test .5:   0.0836236563606574
```

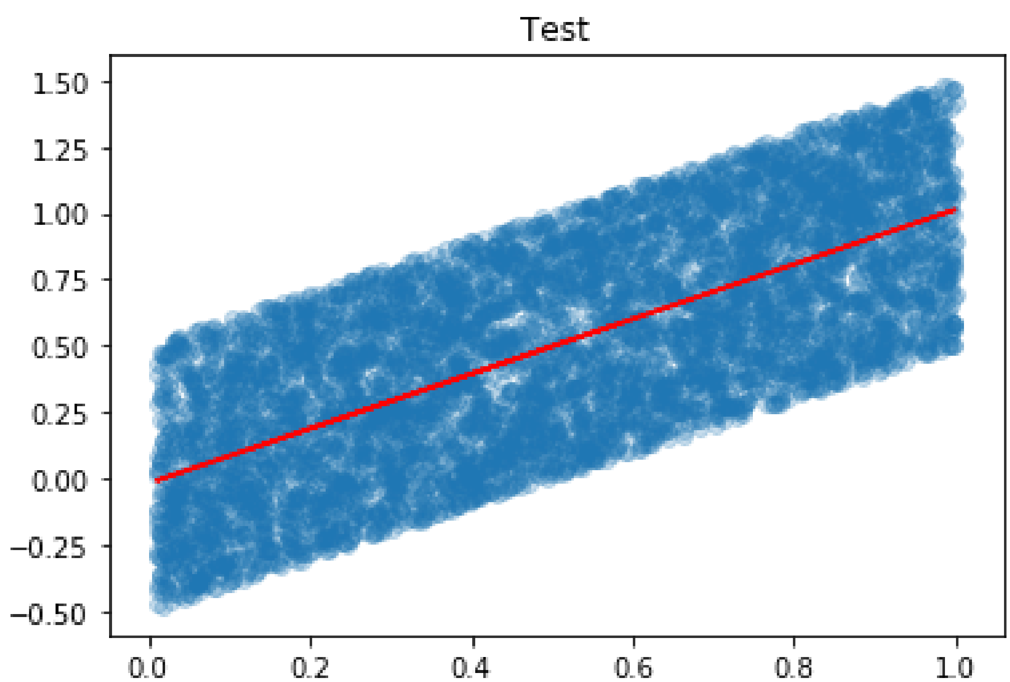




In [16]:
```
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=.6)
print("Train .6: ", mean_squared_error(y_train, np.dot(x_train, model.coef_) + model.in
print("Test .6:  ", mean_squared_error(y_test, np.dot(x_test, model.coef_) + model.inte
model.fit(x_train, y_train)
model.coef_, model.intercept_
plt.scatter(x_train,y_train, alpha=.25)
plt.plot(x_train, np.dot(x_train, model.coef_) + model.intercept_, c="red")
plt.title("Train")
plt.show()

model.fit(x_test, y_test)
model.coef_, model.intercept_
plt.scatter(x_test,y_test, alpha=.25)
plt.plot(x_test, np.dot(x_test, model.coef_) + model.intercept_, c="red")
plt.title("Test")
plt.show()
```

```
Train .6:  0.0839982804721052
Test .6:   0.08484097395214141
```

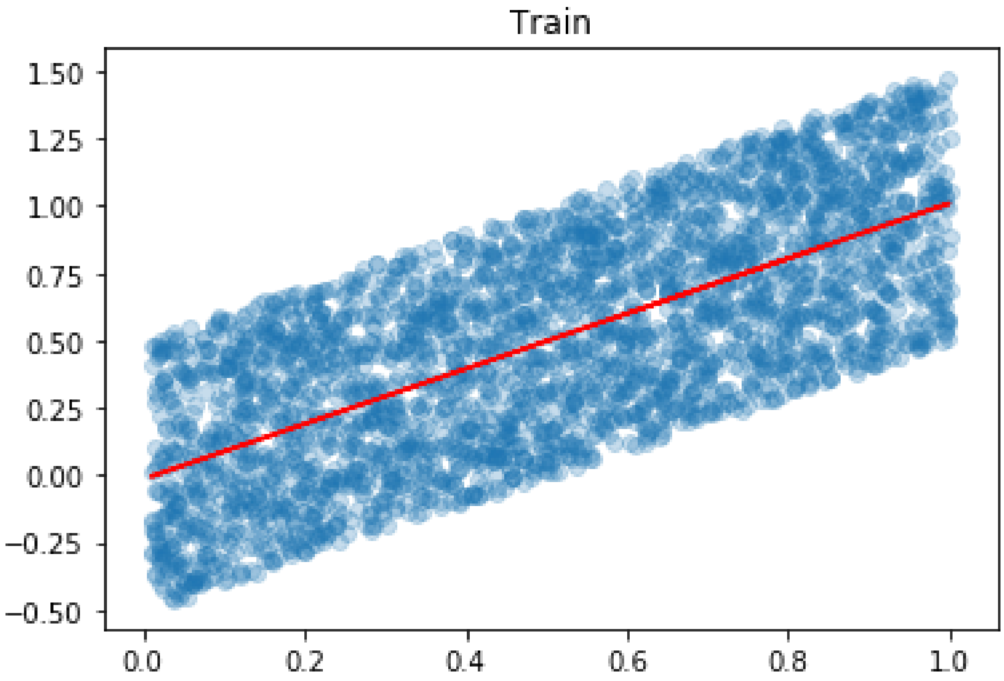


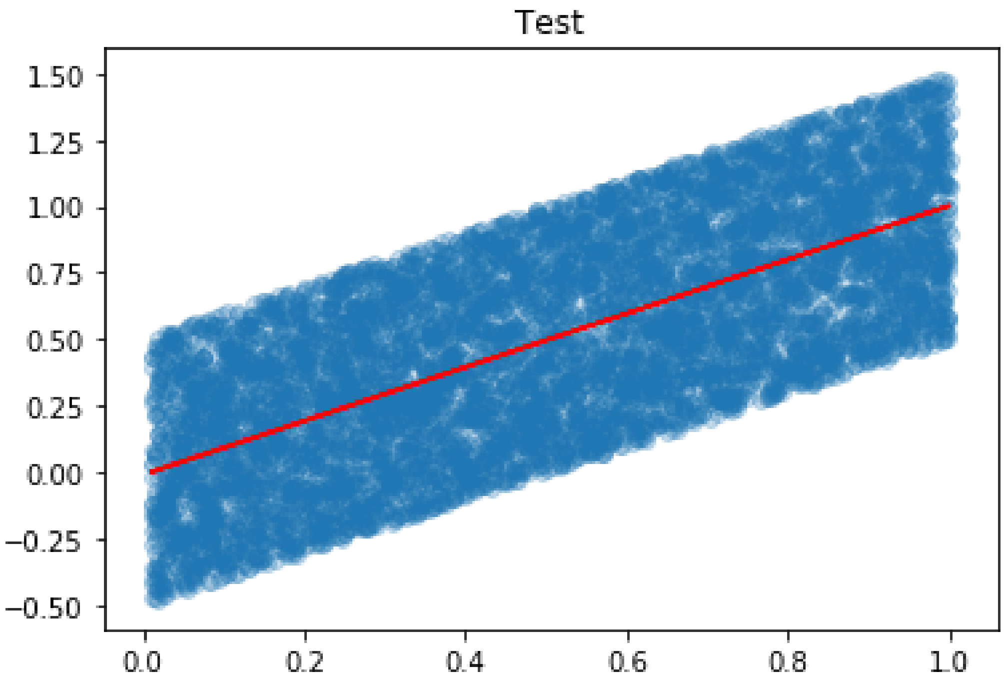



```
In [17]: x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=.7)
         print("Train .7: ", mean_squared_error(y_train, np.dot(x_train, model.coef_) + model.in
```

```python
print("Test .7:  ", mean_squared_error(y_test, np.dot(x_test, model.coef_) + model.inte
model.fit(x_train, y_train)
model.coef_, model.intercept_
plt.scatter(x_train,y_train, alpha=.25)
plt.plot(x_train, np.dot(x_train, model.coef_) + model.intercept_, c="red")
plt.title("Train")
plt.show()

model.fit(x_test, y_test)
model.coef_, model.intercept_
plt.scatter(x_test,y_test, alpha=.25)
plt.plot(x_test, np.dot(x_test, model.coef_) + model.intercept_, c="red")
plt.title("Test")
plt.show()
```

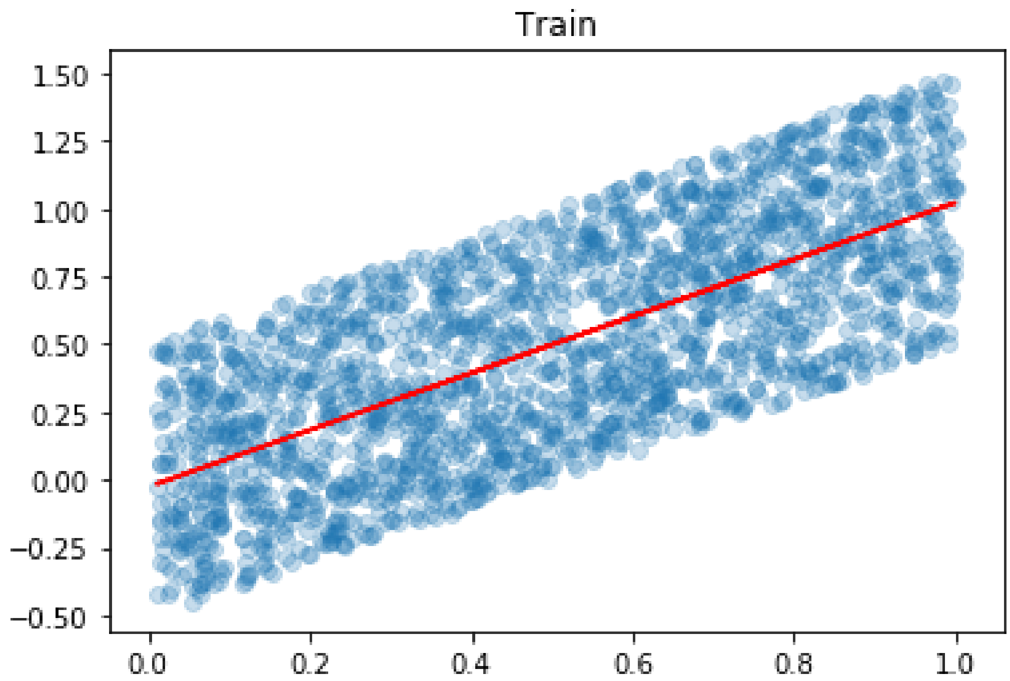
```
Train .7:  0.0853398645767253
Test .7:   0.08414371743149195
```

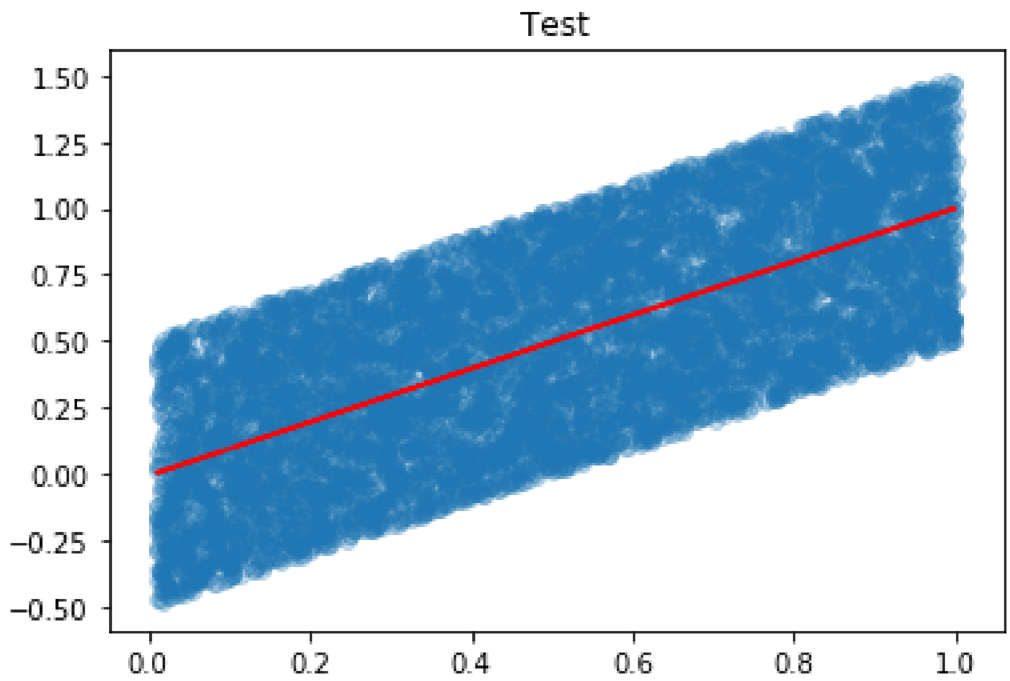




In [18]:
```
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=.8)
print("Train .8: ", mean_squared_error(y_train, np.dot(x_train, model.coef_) + model.in
print("Test .8:  ", mean_squared_error(y_test, np.dot(x_test, model.coef_) + model.inte
model.fit(x_train, y_train)
model.coef_, model.intercept_
plt.scatter(x_train,y_train, alpha=.25)
plt.plot(x_train, np.dot(x_train, model.coef_) + model.intercept_, c="red")
plt.title("Train")
plt.show()

model.fit(x_test, y_test)
model.coef_, model.intercept_
plt.scatter(x_test,y_test, alpha=.25)
plt.plot(x_test, np.dot(x_test, model.coef_) + model.intercept_, c="red")
plt.title("Test")
plt.show()
```

```
Train .8:  0.08450351485453626
Test .8:   0.08447590275806367
```

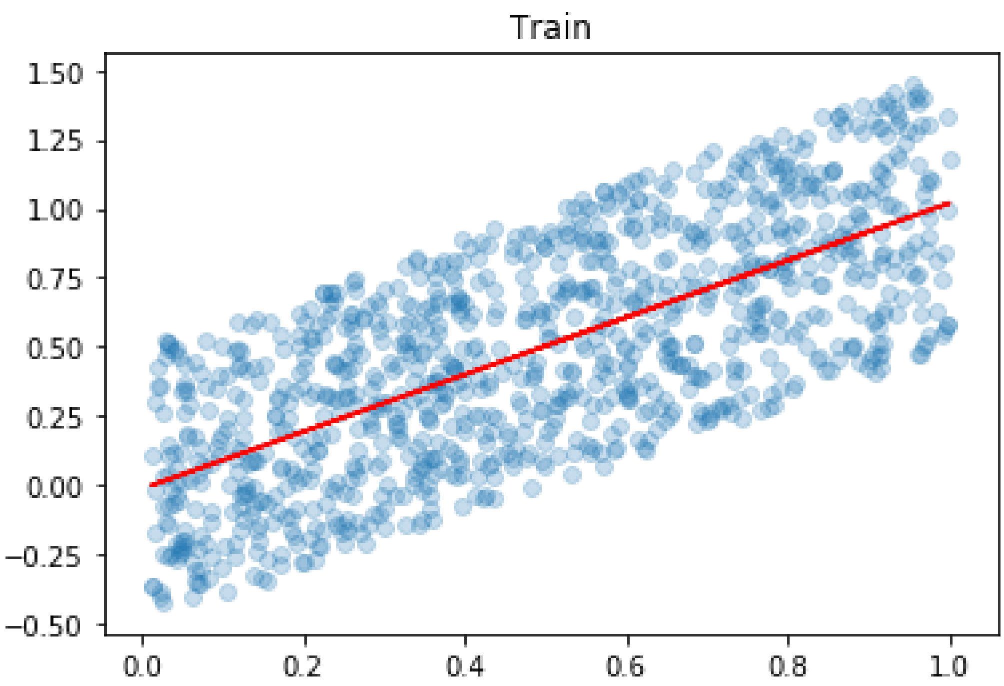


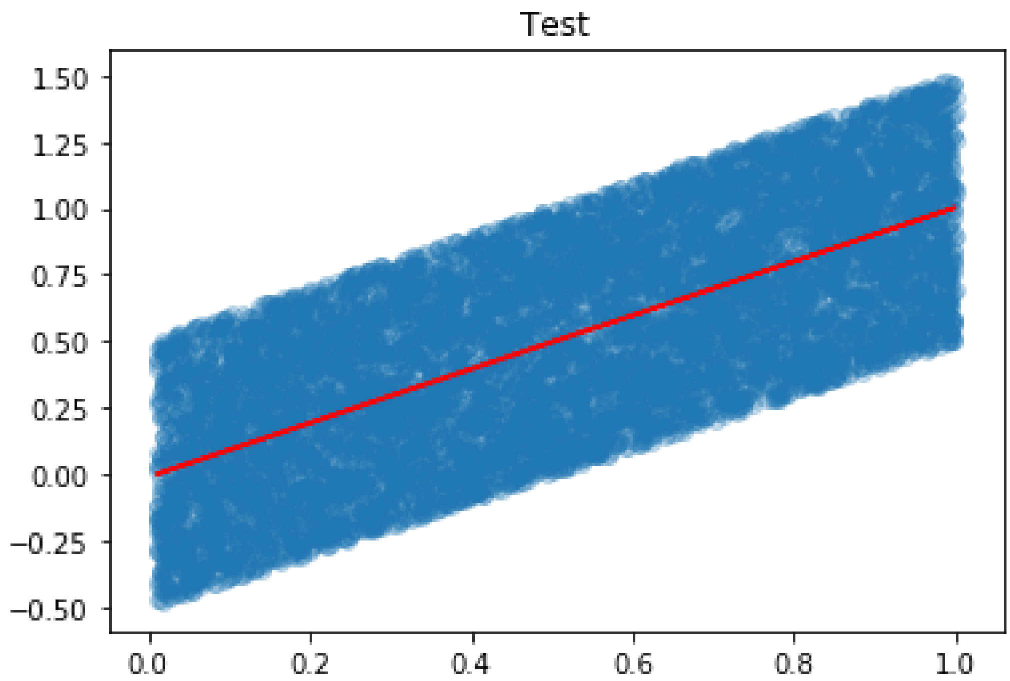


In [19]:
```
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=.9)
print("Train .9: ", mean_squared_error(y_train, np.dot(x_train, model.coef_) + model.in
```

```
print("Test .9:  ", mean_squared_error(y_test, np.dot(x_test, model.coef_) + model.inte
model.fit(x_train, y_train)
model.coef_, model.intercept_
plt.scatter(x_train,y_train, alpha=.25)
plt.plot(x_train, np.dot(x_train, model.coef_) + model.intercept_, c="red")
plt.title("Train")
plt.show()

model.fit(x_test, y_test)
model.coef_, model.intercept_
plt.scatter(x_test,y_test, alpha=.25)
plt.plot(x_test, np.dot(x_test, model.coef_) + model.intercept_, c="red")
plt.title("Test")
plt.show()
```

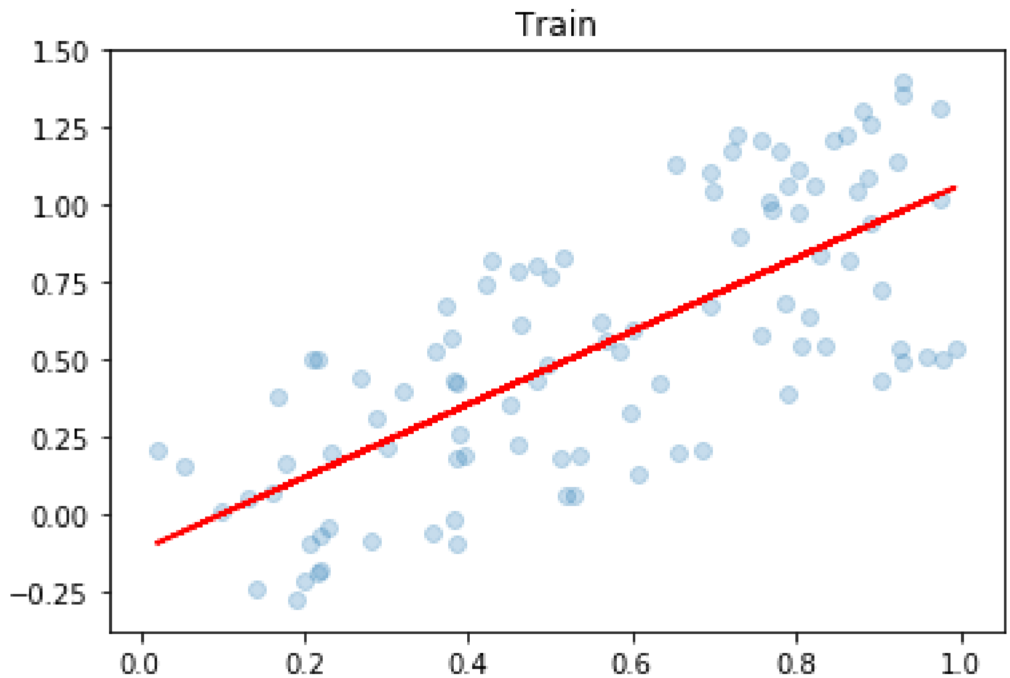
```
Train .9:  0.0838451728371693
Test .9:   0.0845568940578688
```

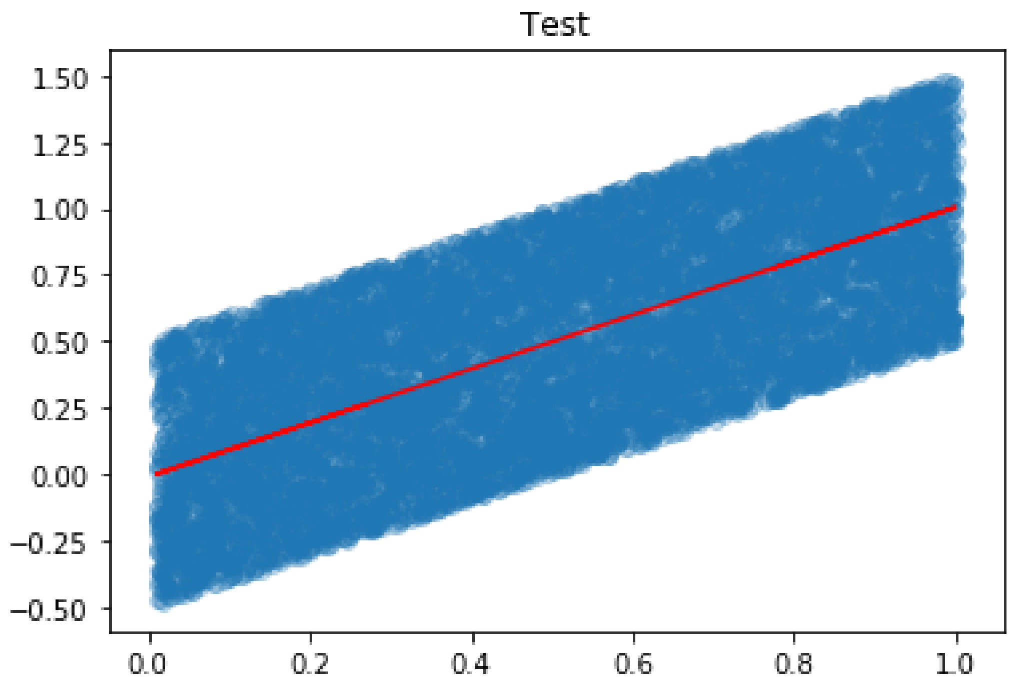





```
In [20]: x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=.99)
         print("Train .99: ", mean_squared_error(y_train, np.dot(x_train, model.coef_) + model.i
         print("Test .99:  ", mean_squared_error(y_test, np.dot(x_test, model.coef_) + model.int
         model.fit(x_train, y_train)
         model.coef_, model.intercept_
         plt.scatter(x_train,y_train, alpha=.25)
         plt.plot(x_train, np.dot(x_train, model.coef_) + model.intercept_, c="red")
         plt.title("Train")
         plt.show()

         model.fit(x_test, y_test)
         model.coef_, model.intercept_
         plt.scatter(x_test,y_test, alpha=.25)
         plt.plot(x_test, np.dot(x_test, model.coef_) + model.intercept_, c="red")
         plt.title("Test")
         plt.show()
```

```
Train .99:  0.09081853855306442
Test .99:   0.08441691699824076
```

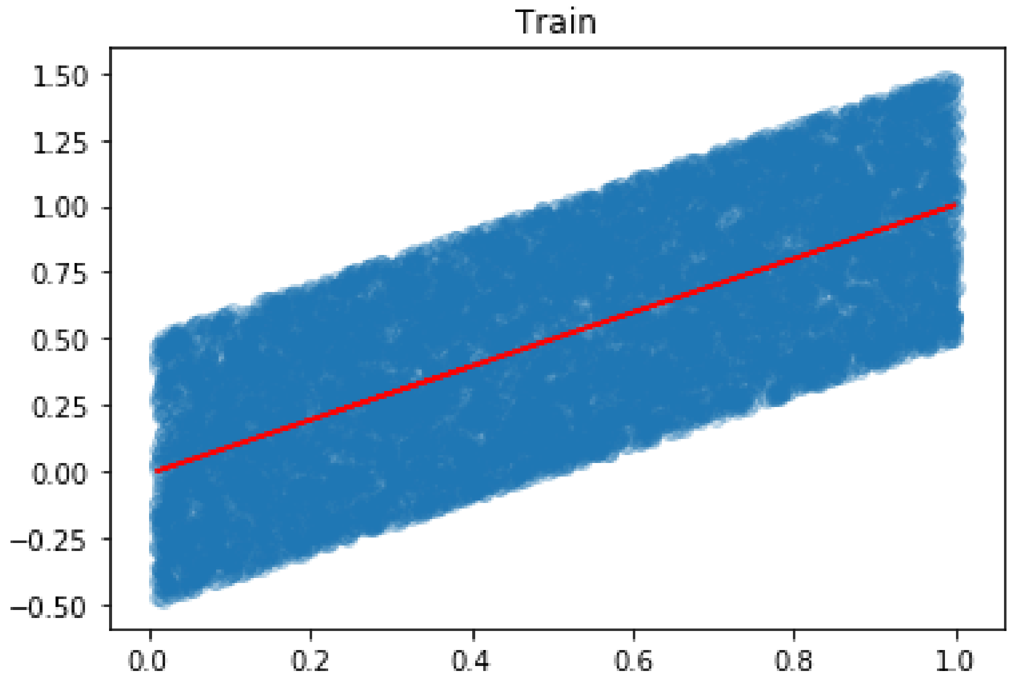




In [41]:
```
# Plotting only three ridges because I think they will look
# the same
```

```
ridge = Ridge()
ridge.fit(x_train, y_train)

x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=.01)
print("Train .01: ", mean_squared_error(y_train, np.dot(x_train, ridge.coef_) + ridge.i
print("Test .01:  ", mean_squared_error(y_test, np.dot(x_test, ridge.coef_) + ridge.int
ridge.fit(x_train, y_train)
ridge.coef_, ridge.intercept_

plt.scatter(x_train,y_train, alpha=.25)
plt.plot(x_train, np.dot(x_train, ridge.coef_) + ridge.intercept_, c="red")
plt.title("Train")
plt.show()

model.fit(x_test, y_test)
model.coef_, ridge.intercept_
plt.scatter(x_test,y_test, alpha=.25)
plt.plot(x_test, np.dot(x_test, ridge.coef_) + ridge.intercept_, c="red")
plt.title("Test")
plt.show()
```

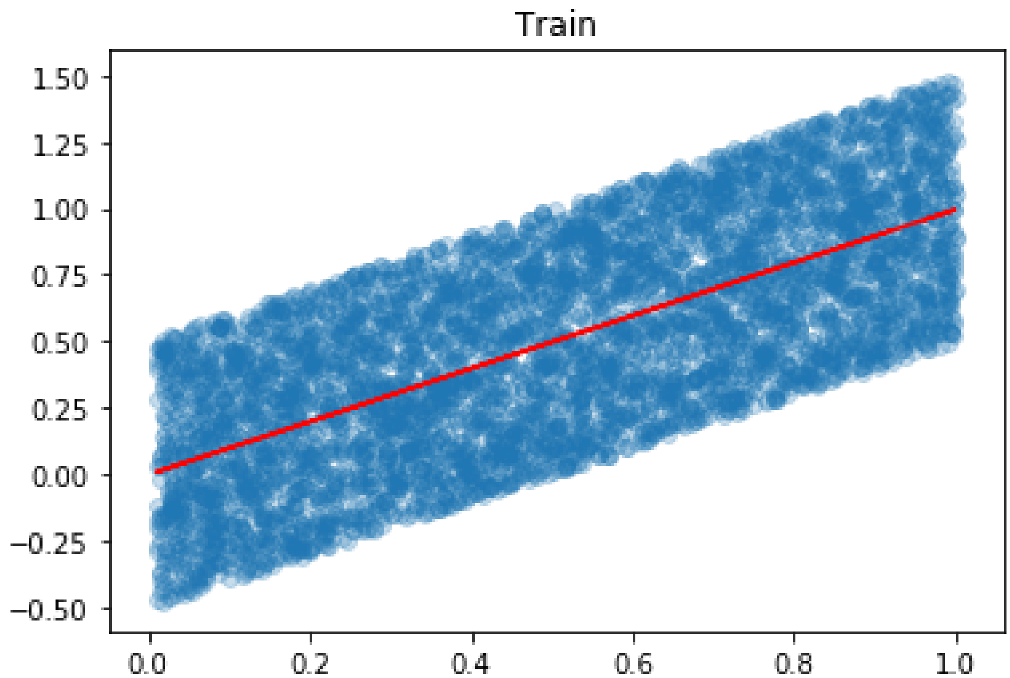
```
Train .01:  0.08438392796689934
Test .01:   0.09404690385420374
```

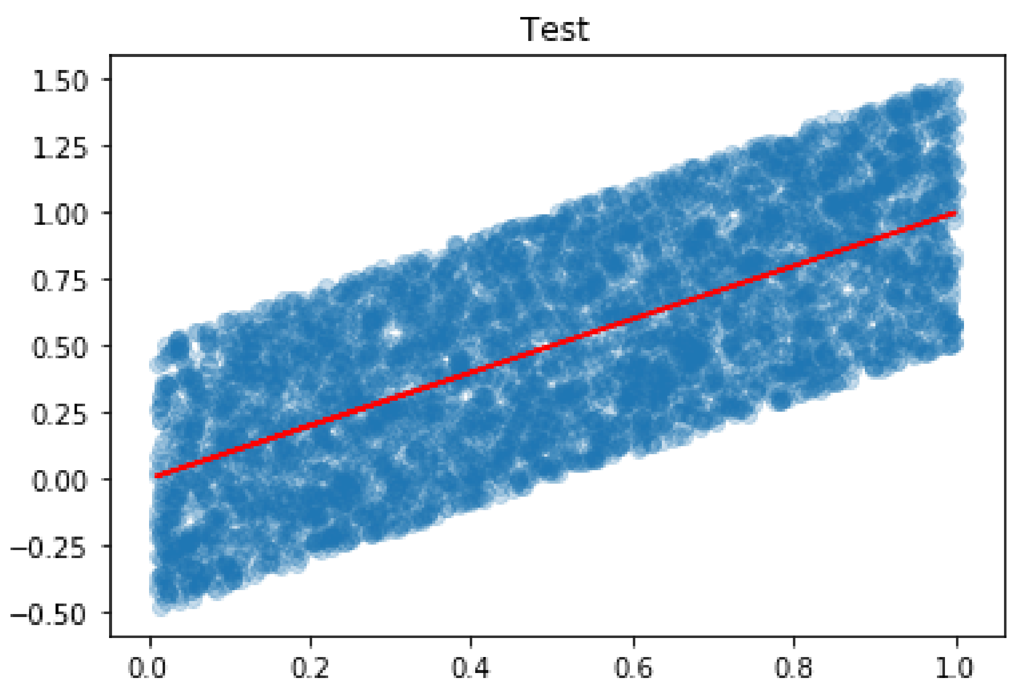



```
In [22]: # The graph may look the same as the graph, but
         # error on the test is more variable than other models
```

```
In [23]: x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=.5)
         print("Train .5: ", mean_squared_error(y_train, np.dot(x_train, ridge.coef_) + ridge.in
         print("Test .5:  ", mean_squared_error(y_test, np.dot(x_test, ridge.coef_) + ridge.inte
         ridge.fit(x_train, y_train)
         ridge.coef_, ridge.intercept_

         plt.scatter(x_train,y_train, alpha=.25)
         plt.plot(x_train, np.dot(x_train, ridge.coef_) + ridge.intercept_, c="red")
         plt.title("Train")
         plt.show()

         model.fit(x_test, y_test)
         model.coef_, ridge.intercept_
         plt.scatter(x_test,y_test, alpha=.25)
         plt.plot(x_test, np.dot(x_test, ridge.coef_) + ridge.intercept_, c="red")
         plt.title("Test")
         plt.show()
```

```
Train .5:  0.08490168433824476
Test .5:   0.08405952400140855
```

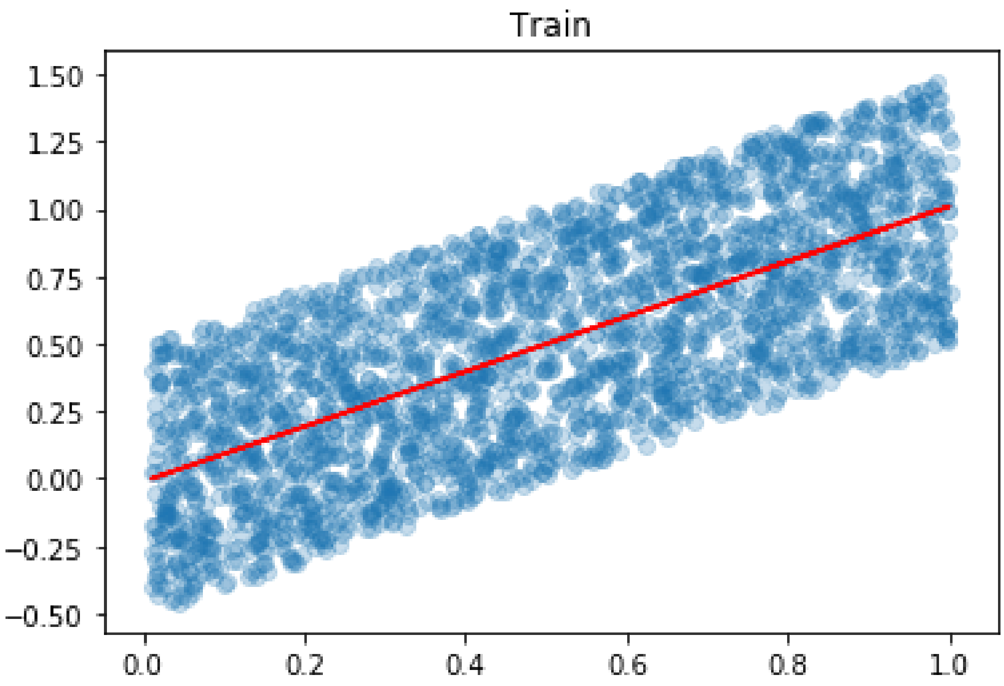


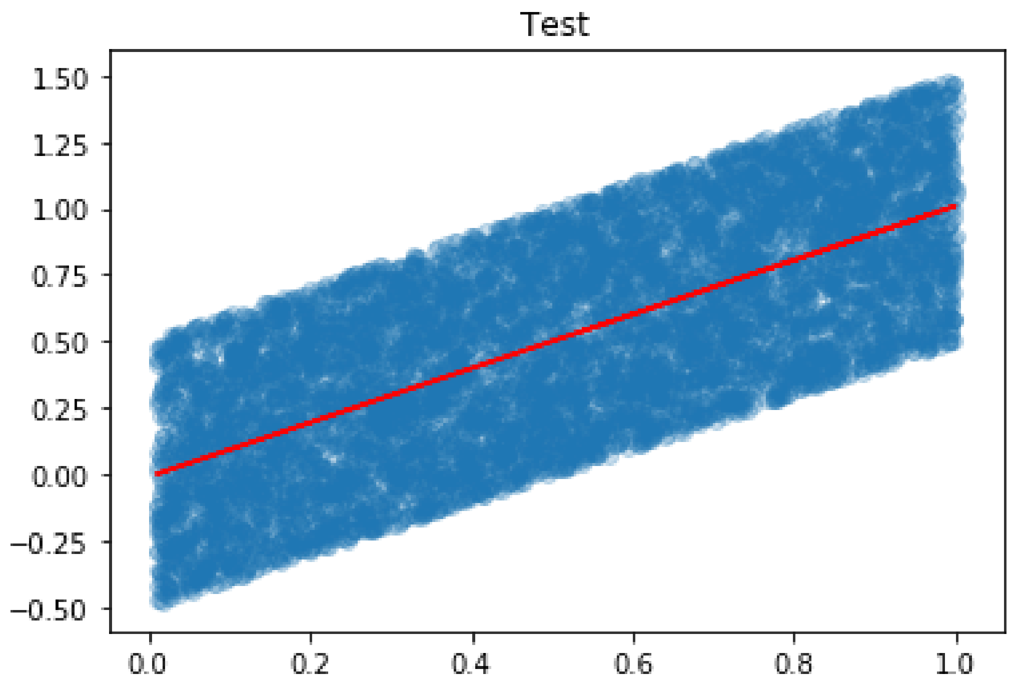


In [24]:
```
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=.75)
print("Train .75: ", mean_squared_error(y_train, np.dot(x_train, ridge.coef_) + ridge.i
```

```
print("Test .75:  ", mean_squared_error(y_test, np.dot(x_test, ridge.coef_) + ridge.int
ridge.fit(x_train, y_train)
ridge.coef_, ridge.intercept_

plt.scatter(x_train,y_train, alpha=.25)
plt.plot(x_train, np.dot(x_train, ridge.coef_) + ridge.intercept_, c="red")
plt.title("Train")
plt.show()

model.fit(x_test, y_test)
model.coef_, ridge.intercept_
plt.scatter(x_test,y_test, alpha=.25)
plt.plot(x_test, np.dot(x_test, ridge.coef_) + ridge.intercept_, c="red")
plt.title("Test")
plt.show()
```

```
Train .75:  0.08443771798692766
Test .75:   0.08453123120789502
```





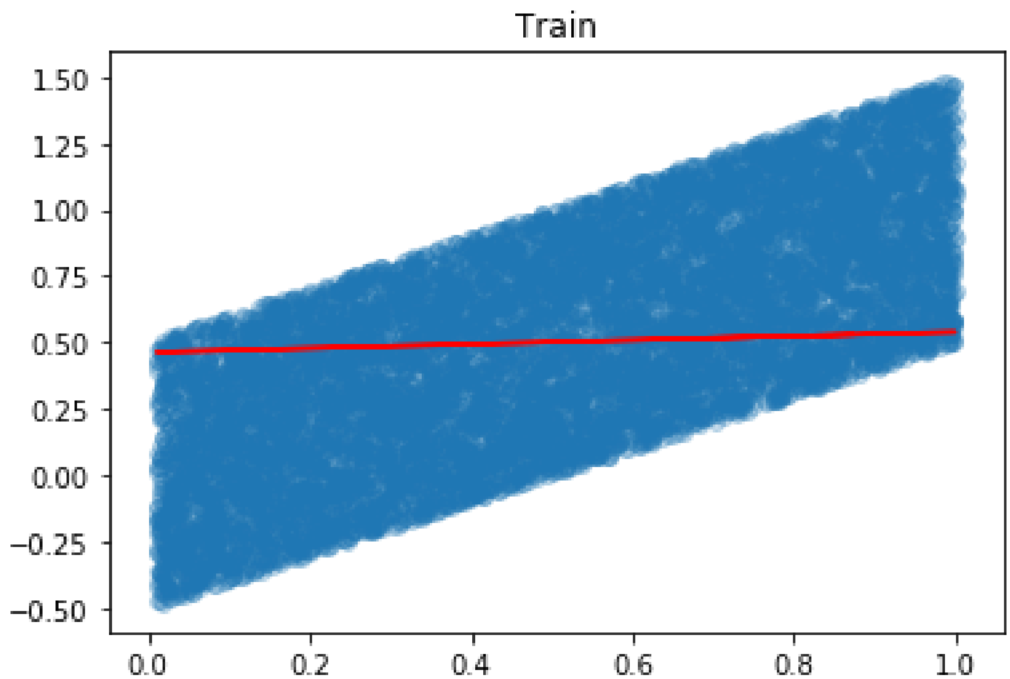
1.5 4. Chose an ideal split size based on the previous plot for Ridge.

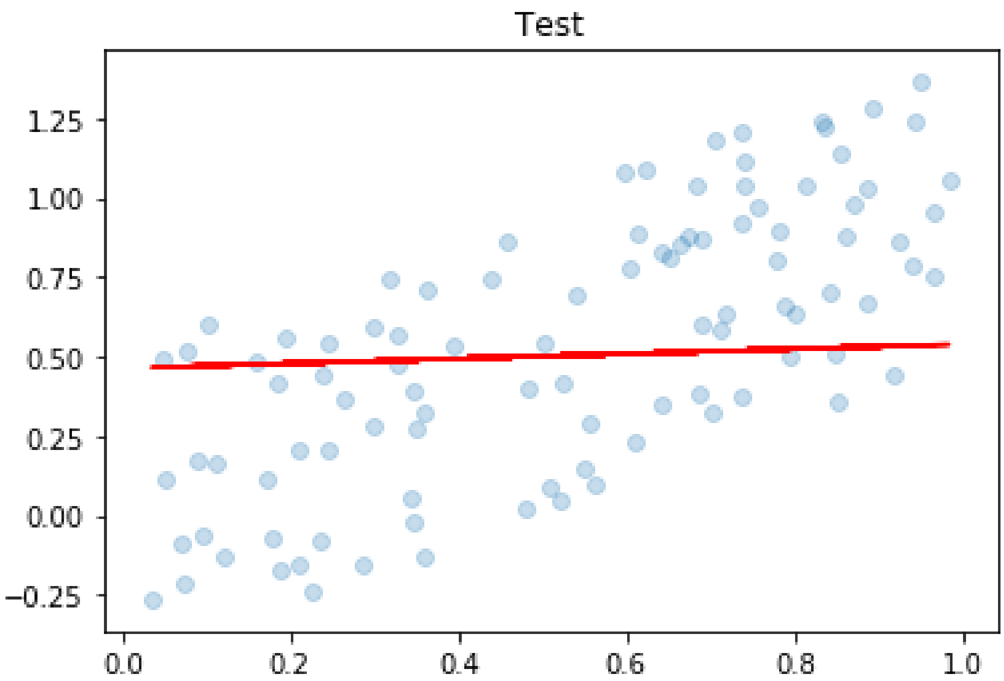
1.6 Vary the Ridge parameter alpha from 0 to any value you'd like above 1. Plot the Train and Test error. Describe what you see based on the alpha parameter's stiffness.

In [25]:
```
# I'm doing .01 just because the lower test error value is
# more prominant and variable than other ones

from sklearn import linear_model

ridge = linear_model.Ridge(alpha=10000) # This one ended up looking flat and super stif
ridge.fit(x, y)

x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=.01)
print("Train .5: ", mean_squared_error(y_train, np.dot(x_train, ridge.coef_) + ridge.in
print("Test .5:  ", mean_squared_error(y_test, np.dot(x_test, ridge.coef_) + ridge.inte
ridge.fit(x_train, y_train)
ridge.coef_, ridge.intercept_

plt.scatter(x_train,y_train, alpha=.25)
plt.plot(x_train, np.dot(x_train, ridge.coef_) + ridge.intercept_, c="red")
plt.title("Train")
plt.show()
```

```
model.fit(x_test, y_test)
model.coef_, ridge.intercept_
plt.scatter(x_test,y_test, alpha=.25)
plt.plot(x_test, np.dot(x_test, ridge.coef_) + ridge.intercept_, c="red")
plt.title("Test")
plt.show()
```

```
Train .5:  0.15656488231218008
Test .5:   0.1601065834076474
```



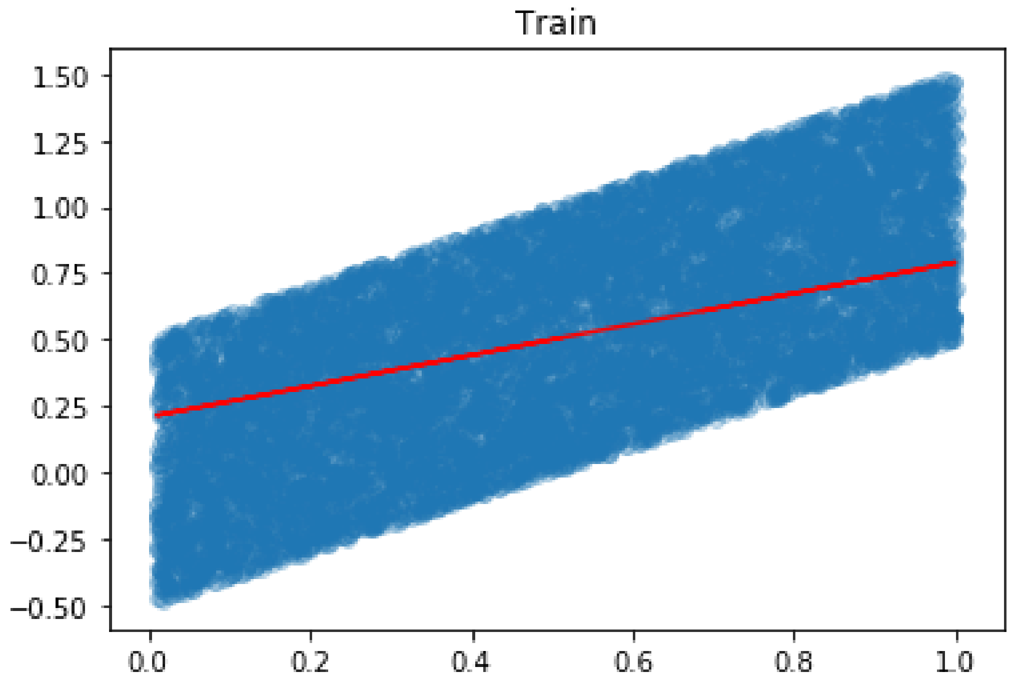


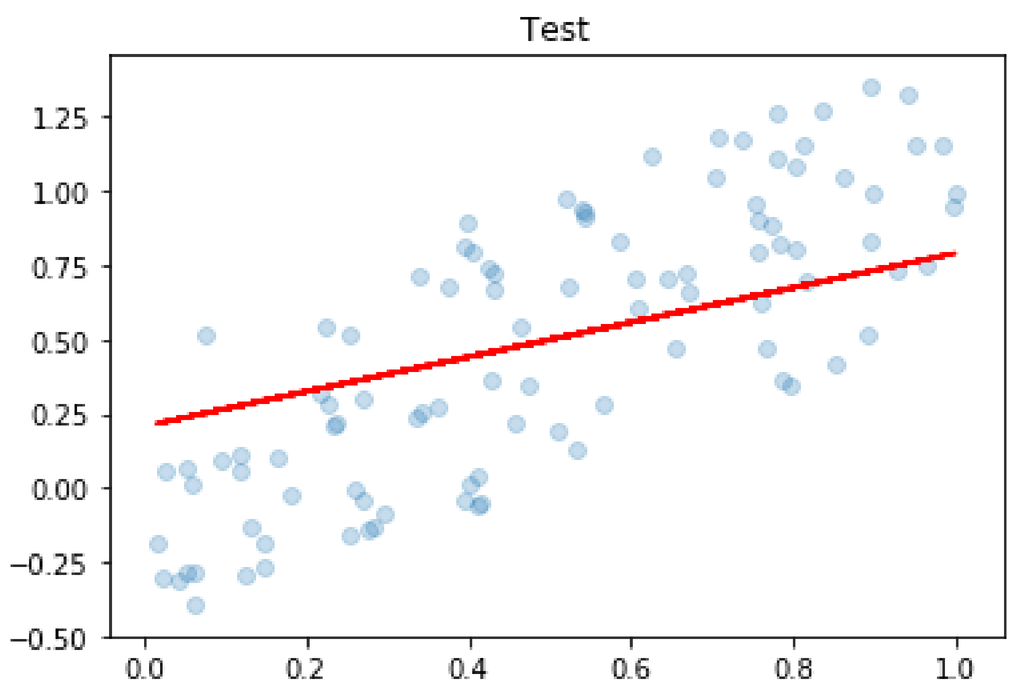

```
In [26]: # Well, that's interesting. the error value is much higher
         # of course.
```

```
In [27]: ridge = linear_model.Ridge(alpha=600)
         ridge.fit(x, y)

         x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=.01)
         print("Train .5: ", mean_squared_error(y_train, np.dot(x_train, ridge.coef_) + ridge.in
         print("Test .5:  ", mean_squared_error(y_test, np.dot(x_test, ridge.coef_) + ridge.inte
         ridge.fit(x_train, y_train)
         ridge.coef_, ridge.intercept_

         plt.scatter(x_train,y_train, alpha=.25)
         plt.plot(x_train, np.dot(x_train, ridge.coef_) + ridge.intercept_, c="red")
         plt.title("Train")
         plt.show()

         model.fit(x_test, y_test)
         model.coef_, ridge.intercept_
         plt.scatter(x_test,y_test, alpha=.25)
         plt.plot(x_test, np.dot(x_test, ridge.coef_) + ridge.intercept_, c="red")
         plt.title("Test")
         plt.show()
```

```
Train .5:  0.09940183835134718
```

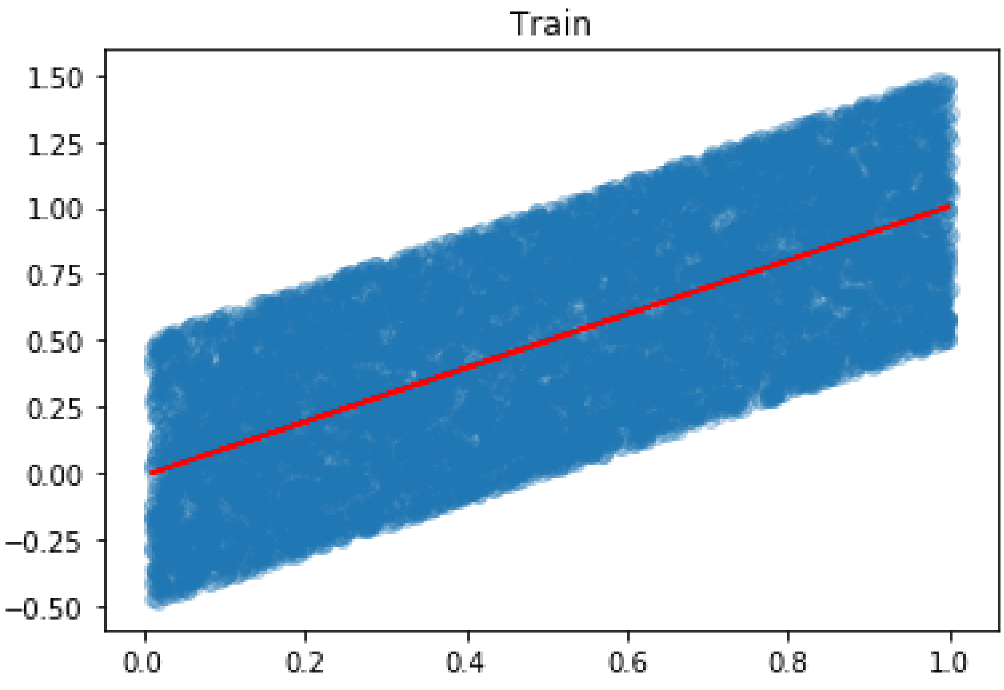
```
Test .5:    0.12038378103461717
```

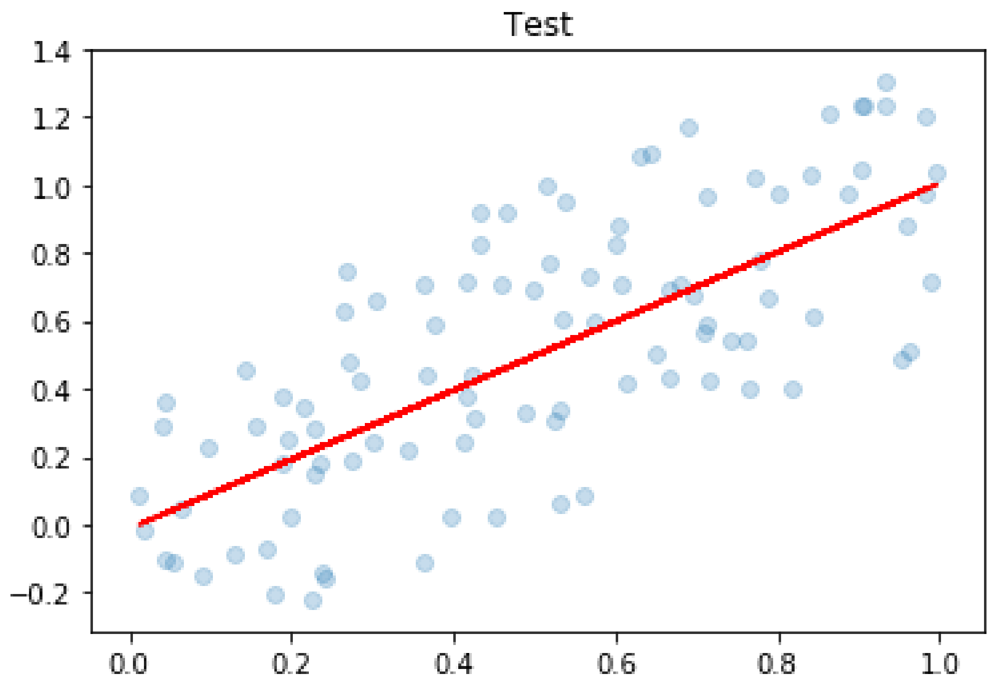




```
In [28]: ridge = linear_model.Ridge(alpha=0)
         ridge.fit(x, y)

         x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=.01)
         print("Train .5: ", mean_squared_error(y_train, np.dot(x_train, ridge.coef_) + ridge.in
         print("Test .5:  ", mean_squared_error(y_test, np.dot(x_test, ridge.coef_) + ridge.inte
         ridge.fit(x_train, y_train)
         ridge.coef_, ridge.intercept_

         plt.scatter(x_train,y_train, alpha=.25)
         plt.plot(x_train, np.dot(x_train, ridge.coef_) + ridge.intercept_, c="red")
         plt.title("Train")
         plt.show()

         model.fit(x_test, y_test)
         model.coef_, ridge.intercept_
         plt.scatter(x_test,y_test, alpha=.25)
         plt.plot(x_test, np.dot(x_test, ridge.coef_) + ridge.intercept_, c="red")
         plt.title("Test")
         plt.show()
```

```
Train .5:  0.08460000015290149
Test .5:   0.0726281198726732
```





1.7 Bonus. Either: Generate data with a polynomial shape or use real data that you find on your own. Choose whatever regression model and process you'd like (Ridge, polynomial, etc.) and plot the Train-Test errors vs. any parameter your Model depends on (e.g. `alpha`, `degree`, etc.)

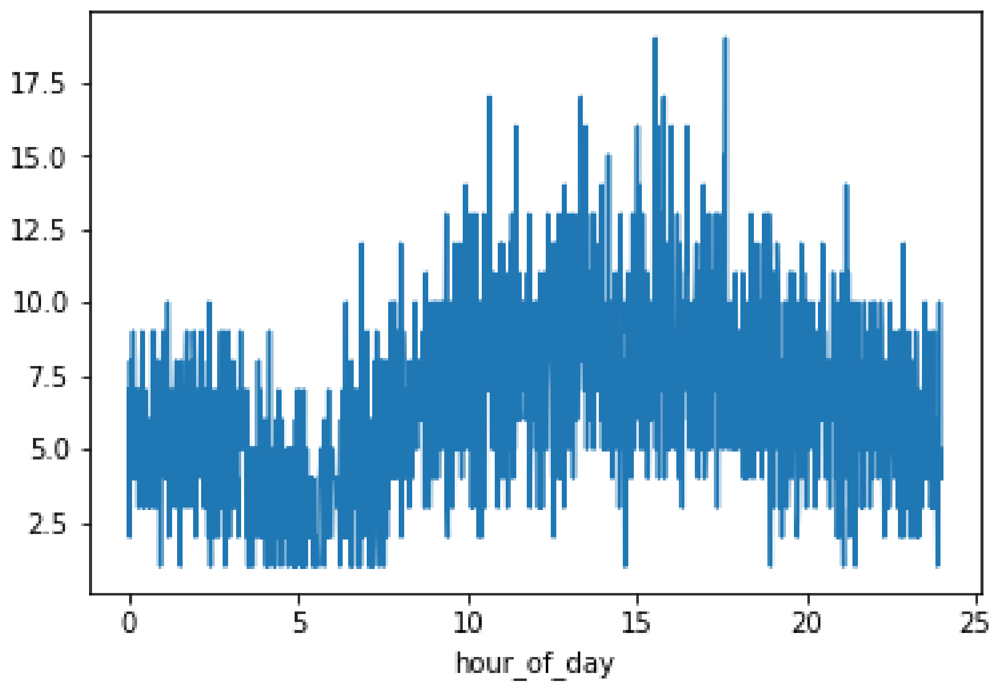
```
In [29]: # Fire dispatch data from the City of Tempe, AZ

         import pandas as pd

         fire = pd.read_csv("https://data.tempe.gov/dataset/7584b16e-aee8-48dc-9b22-e570a3b61c27
         fire.head()
         fire['dispatched'] = pd.to_datetime(fire['Dispatch_Datetime'])
         %time fire['end'] = pd.to_datetime(fire['Arrival_Datetime'])
```

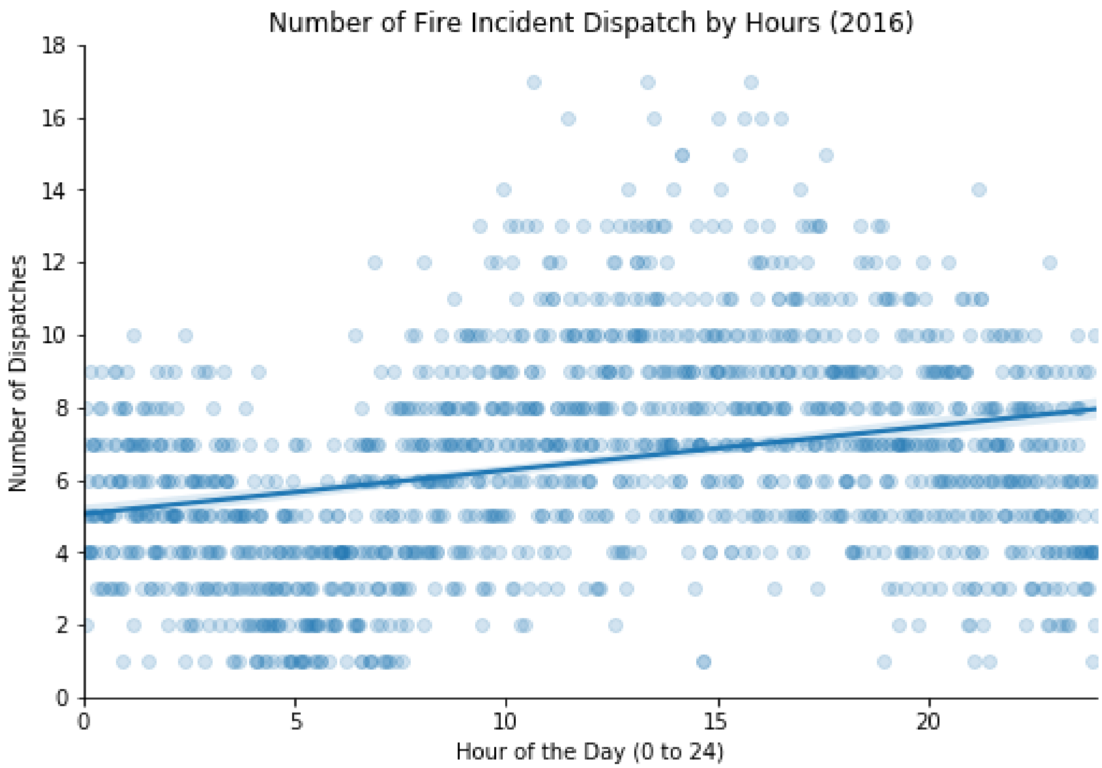
```
CPU times: user 1.09 s, sys: 5.67 ms, total: 1.1 s
Wall time: 1.11 s
```

```
In [30]: import seaborn as sns

         fire['hour_of_day'] = (fire.dispatched.dt.hour + (fire.dispatched.dt.minute/60).round(2

         hours = fire.groupby('hour_of_day').agg('count')
         hours['hour'] = hours.index
```

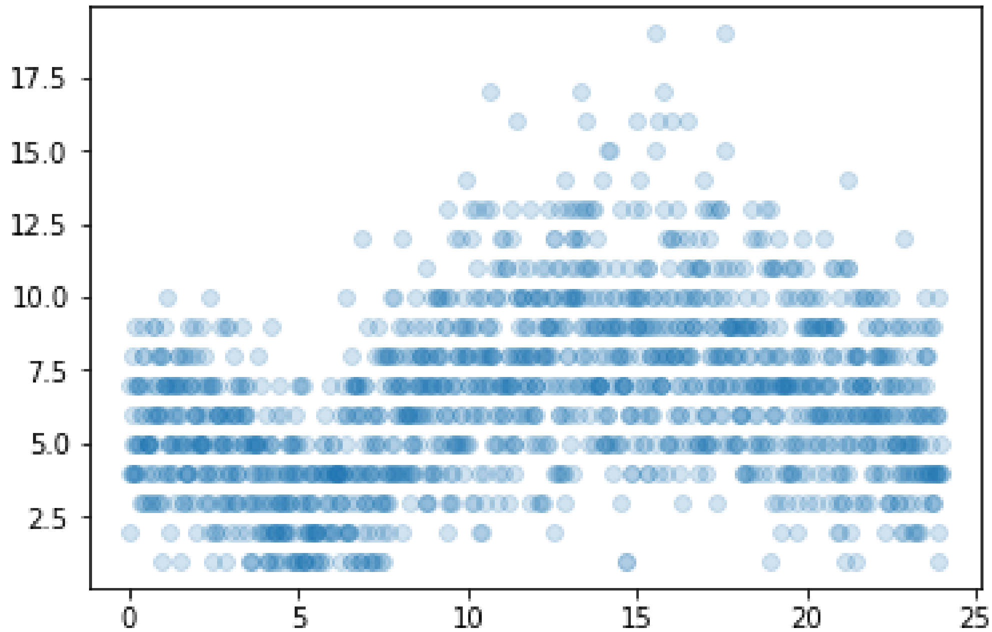
```
hours.dispatched.plot()
# import seaborn as sns

sns.lmplot(x='hour', y='dispatched', data=hours, aspect=1.5, scatter_kws={'alpha':0.2})
plt.title("Number of Fire Incident Dispatch by Hours (2016)")
plt.ylabel('Number of Dispatches')
plt.xlabel('Hour of the Day (0 to 24)')
plt.xlim(0,24)
plt.ylim(0,18)
```

```
Out[30]: (0, 18)
```

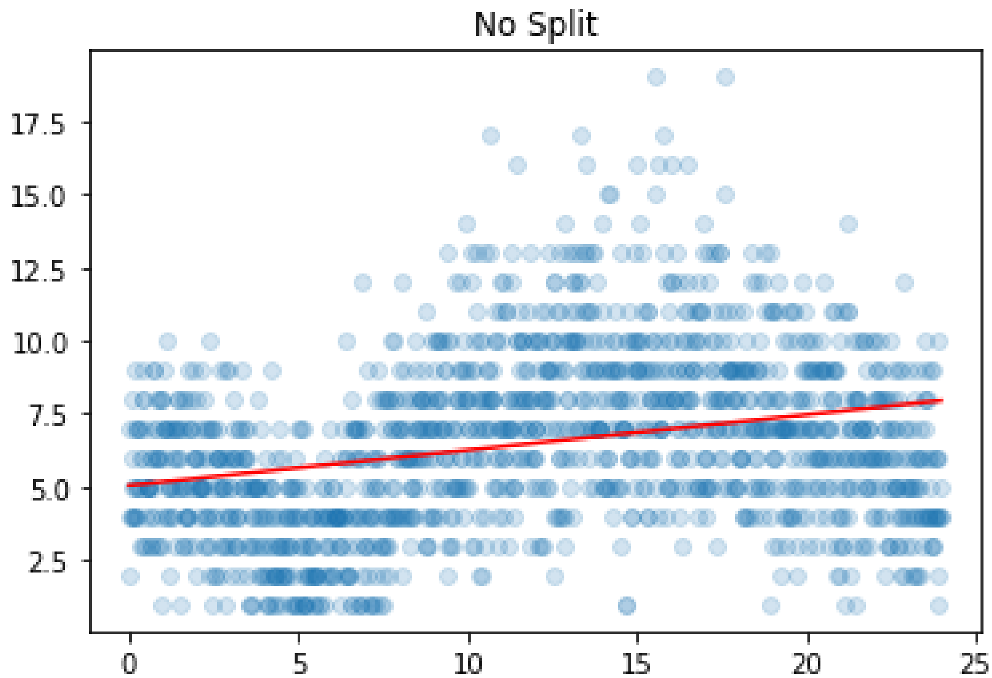





```
In [31]: dispatched = hours['dispatched']
         hour = hours[['hour']]

         plt.scatter(hour, dispatched, alpha = .2)

Out[31]: <matplotlib.collections.PathCollection at 0x1a175f5f98>
```

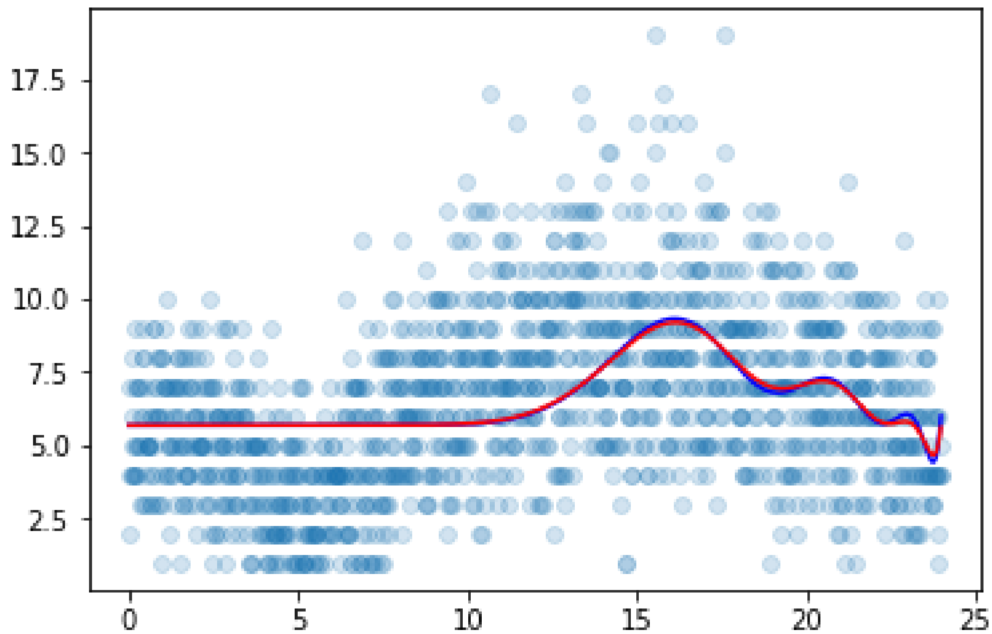



```
In [32]: model = LinearRegression()
         model.fit(hour, dispatched)

         nosplit = model.coef_, model.intercept_
         print("Fit:" , model.coef_, model.intercept_)

         plt.scatter(hour, dispatched, alpha=0.2)
         plt.plot(hour, np.dot(hour, model.coef_) + model.intercept_, c="red")
         plt.title("No Split")

Fit: [0.12082424] 5.051001323586405


Out[32]: Text(0.5,1,'No Split')
```



In [33]:
```
# polynomial linear regression x25 degree

from sklearn.preprocessing import PolynomialFeatures

poly_25 = PolynomialFeatures(degree=25)
hour_25 = poly_25.fit_transform(hour)

linear = linear_model.LinearRegression()
linear.fit(hour_25, dispatched)
linear.coef_, linear.intercept_

ridge = linear_model.Ridge()
ridge.fit(hour_25, dispatched)
ridge.coef_, ridge.intercept_

import numpy as np

plt.scatter(hour, dispatched, alpha=.2)
plt.plot(hour, np.dot(hour_25, linear.coef_) + linear.intercept_, c='b')
plt.plot(hour, np.dot(hour_25, ridge.coef_) + ridge.intercept_, c='r')
```

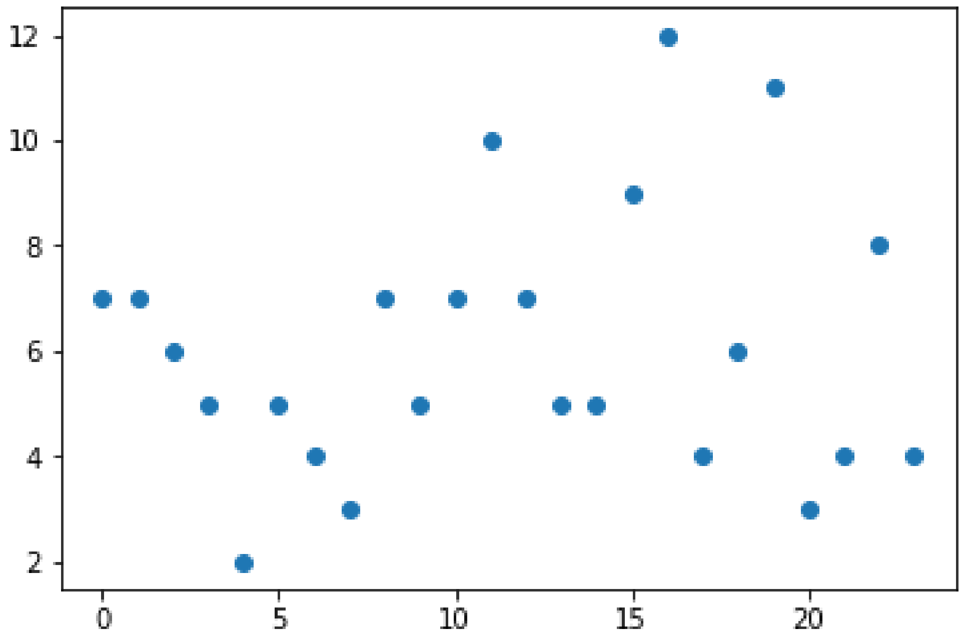
Out[33]: [<matplotlib.lines.Line2D at 0x1a17e13a58>]




```
In [34]: # I love it.

In [35]: # But, I think split size will be the most appropriate for this type of analysis
         dispatched = hours['dispatched']
         hour = hours['hour']

         def shuffle(a, b):
             assert len(a) == len(b)
             p = np.random.permutation(len(a))
             return p

         p = shuffle(hour, dispatched)

         print(p)

         plt.scatter(hour[p],dispatched[p])

[ 304  995  548 ...  202  836 1400]


/Users/dell/anaconda3/lib/python3.6/site-packages/pandas/core/series.py:841: FutureWarning:
Passing list-likes to .loc or [] with any missing label will raise
KeyError in the future, you can use .reindex() as an alternative.

See the documentation here:
https://pandas.pydata.org/pandas-docs/stable/indexing.html#deprecate-loc-reindex-listlike
```

```
  return self.loc[key]
```

Out[35]: <matplotlib.collections.PathCollection at 0x1a17576588>



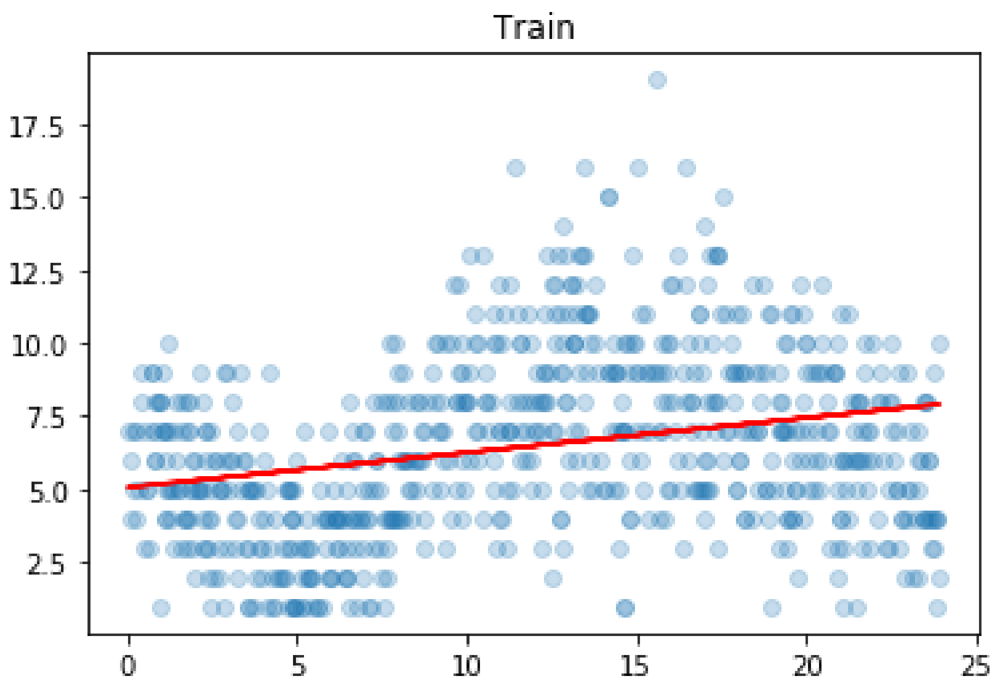
In [36]:
```
# My time is short, so I don't want to spend my time on an error...
```

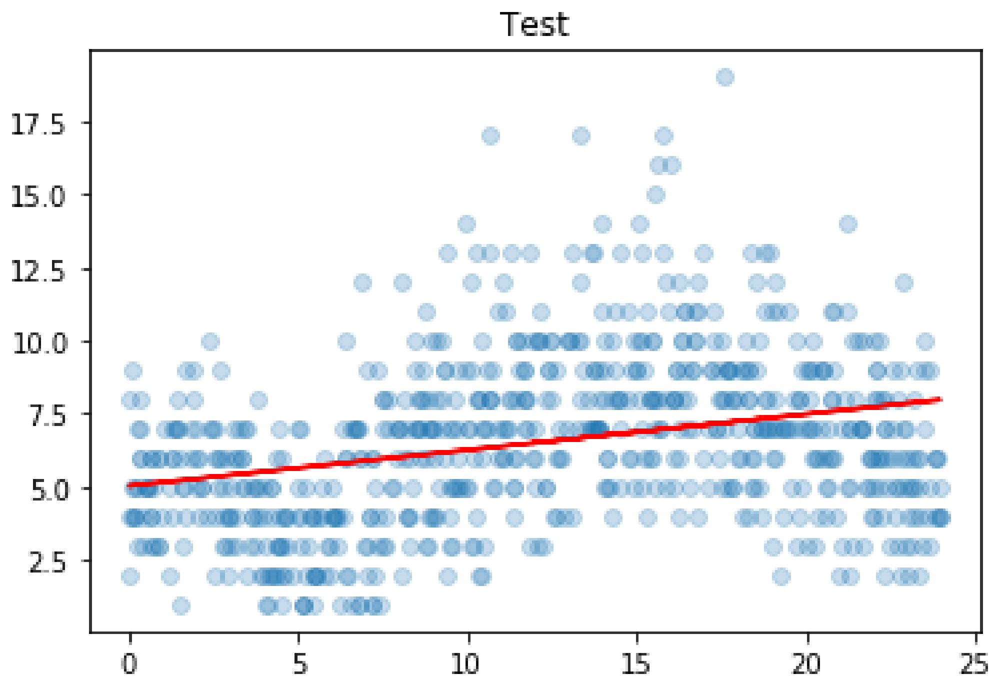
In [37]:
```
# Moving on to a different method

dispatched = hours['dispatched']
hour = hours[['hour']]

hour_train, hour_test, dispatched_train, dispatched_test = train_test_split(hour, dispa

model = LinearRegression()
model.fit(hour_train, dispatched_train)

print("Train .5: ", mean_squared_error(dispatched_train, np.dot(hour_train, model.coef_
print("Test .5:  ", mean_squared_error(dispatched_test, np.dot(hour_test, model.coef_)
model.fit(hour_train, dispatched_train)
model.coef_, model.intercept_
```

```
Train .5:  8.774710294324322
Test .5:   7.747893969743095
```

Out[37]: (array([0.11879698]), 5.066547860346615)

In [38]:
```
plt.scatter(hour_train,dispatched_train, alpha=.25)
plt.plot(hour_train, np.dot(hour_train, model.coef_) + model.intercept_, c="red")
plt.title("Train")
plt.show()


model.fit(hour_test, dispatched_test)
model.coef_, model.intercept_
plt.scatter(hour_test,dispatched_test, alpha=.25)
plt.plot(hour_test, np.dot(hour_test, model.coef_) + model.intercept_, c="red")
plt.title("Test")
plt.show()
```






```
In [39]: # Conclusion: I like the polynomial ridge regression the best out of all model I've use
```