cs109a_hw2_109_FINAL

September 26, 2018

1 CS109A Introduction to Data Science:

1.1 Homework 2: Linear and k-NN Regression

Harvard University Fall 2018 Instructors: Pavlos Protopapas, Kevin Rader

```
In [1]: #RUN THIS CELL
        import requests
        from IPython.core.display import HTML
        styles = requests.get("https://raw.githubusercontent.com/Harvard-IACS/2018-CS109A/maste
        HTML(styles)
```

```
Out[1]: <IPython.core.display.HTML object>
```

1.1.1 INSTRUCTIONS

- To submit your assignment follow the instructions given in canvas.
- Restart the kernel and run the whole notebook again before you submit.
- If you submit individually and you have worked with someone, please include the name of your [one] partner below.
- As much as possible, try and stick to the hints and functions we import at the top of the homework, as those are the ideas and tools the class supports and is aiming to teach. And if a problem specifies a particular library you're required to use that library, and possibly others from the import list.

Names of people you have worked with goes here:

```
In [2]: import numpy as np
        import pandas as pd
        import matplotlib
        import matplotlib.pyplot as plt
        from sklearn.metrics import r2_score
        from sklearn.neighbors import KNeighborsRegressor
        from sklearn.linear_model import LinearRegression
        from sklearn.model_selection import train_test_split
        import statsmodels.api as sm
        from statsmodels.api import OLS
        %matplotlib inline
```

```
# some more required libraries
#import seaborn as sns
#from sklearn.cross_validation import train_test_split
from matplotlib.lines import Line2D
```

Predicting Taxi Pickups in NYC

In this homework, we will explore k-nearest neighbor and linear regression methods for predicting a quantitative variable. Specifically, we will build regression models that can predict the number of taxi pickups in New York city at any given time of the day. These prediction models will be useful, for example, in monitoring traffic in the city.

The data set for this problem is given in the file `dataset_1.csv`. You will need to separate it into training and test sets. The first column contains the time of a day in minutes, and the second column contains the number of pickups observed at that time. The data set covers taxi pickups recorded in NYC during Jan 2015.

We will fit regression models that use the time of the day (in minutes) as a predictor and predict the average number of taxi pickups at that time. The models will be fitted to the training set and evaluated on the test set. The performance of the models will be evaluated using the R^2 metric.

Question 1 [25 pts]

1.1. Use pandas to load the dataset from the csv file `dataset_1.csv` into a pandas data frame. Use the `train_test_split` method from `sklearn` with a `random_state` of 42 and a `test_size` of 0.2 to split the dataset into training and test sets. Store your train set dataframe in the variable `train_data`. Store your test set dataframe in the variable `test_data`.

1.2. Generate a scatter plot of the training data points with clear labels on the x and y axes. The time of the day on the x-axis and the number of taxi pickups on the y-axis. Make sure to title your plot.

1.3. Does the pattern of taxi pickups make intuitive sense to you?

1.1.2 Answers

1.1 Use pandas to load the dataset from the csv file ...

```
In [3]: # read the file
        # your code here
        dftaxi = pd.read_csv('data/dataset_1.csv')

In [4]: #check your dataframe
        dftaxi.head()
```

Out[4]:

	TimeMin	PickupCount
0	860.0	33.0
1	17.0	75.0
2	486.0	13.0
3	300.0	5.0
4	385.0	10.0

```
In [5]: # split the data
        # your code here
```

```
        #set random_state to get the same split every time
        train_data, test_data = train_test_split(dftaxi, test_size=0.2, random_state=42)
```

```
In [6]: print(dftaxi.shape, train_data.shape, test_data.shape)
```

```
(1250, 2) (1000, 2) (250, 2)
```

```
In [7]: # Test size is indeed 20% of total
        # your code here
        test_data.shape[0]/dftaxi.shape[0]
```

```
Out[7]: 0.2
```

```
In [8]: print(test_data.head())
```

```
      TimeMin  PickupCount
680     436.0         24.0
1102    408.0          9.0
394    1189.0         51.0
930     139.0         66.0
497     756.0         15.0
```

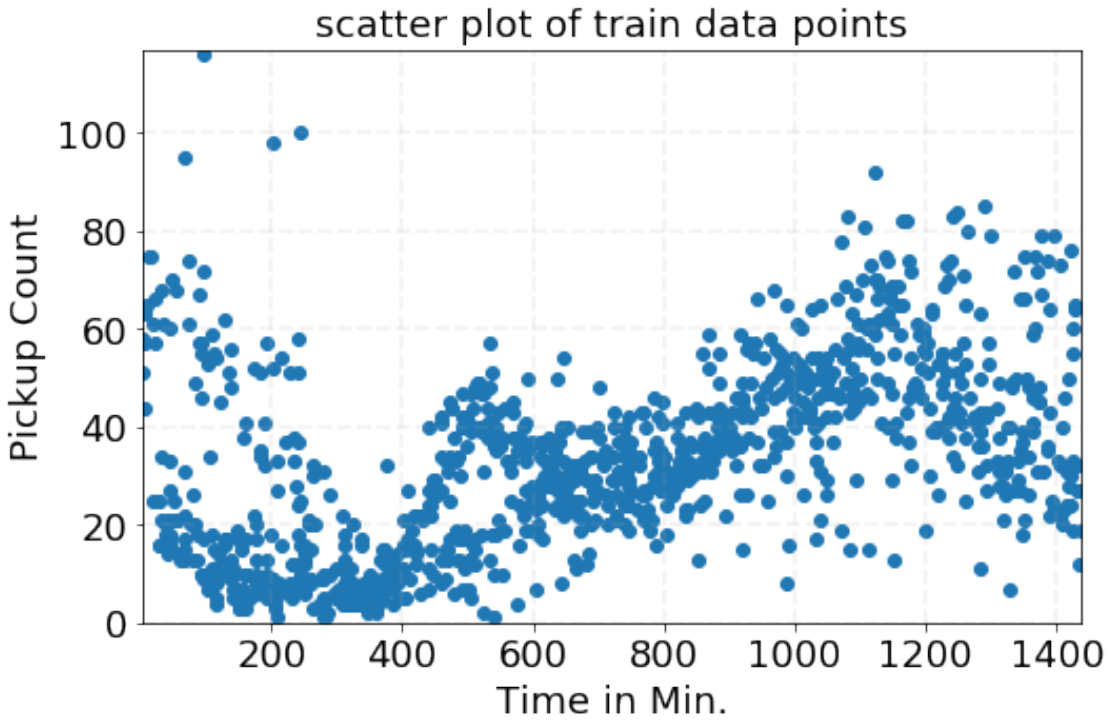
1.2 Generate a scatter plot of the training data points

```
In [33]: # your code here
         # function for scatter plot
         def gen_scatterplot(x, y, xlabel, ylabel, title):
             # font size
             f_size = 18

             # make the figure
             fig, ax = plt.subplots(1,1, figsize=(8,5)) # Create figure object

             # set axes limits to make the scale nice
             ax.set_xlim(np.min(x)-1, np.max(x) + 1)
             ax.set_ylim(np.min(y)-1, np.max(y) + 1)

             # adjust size of tickmarks in axes
             ax.tick_params(labelsize = f_size)

             # adjust size of axis label
             ax.set_xlabel(xlabel, fontsize = f_size)
             ax.set_ylabel(ylabel, fontsize = f_size)

             # set figure title label
             ax.set_title(title, fontsize = f_size)
```

```
    # set up grid
    ax.grid(True, lw=1.75, ls='--', alpha=0.15)

    # make actual plot
    ax.scatter(x, y)
    #ax.legend(loc='best', fontsize = f_size)

    return ax
```

```
In [34]: gen_scatterplot(train_data.TimeMin, train_data.PickupCount, 'Time in Min.', 'Pickup C
```

```
Out[34]: <matplotlib.axes._subplots.AxesSubplot at 0x1fc67c58e48>
```



1.3 Discuss your results. Does the pattern of taxi pickups make intuitive sense to you?
your answer here
Given the x-axis is time of the day in minutes, we notice:

(a) There are more pick ups in the early hours of the day and later hours of the day.

(b) In other words, number of taxi pick ups in NYC is more between 4pm and 5am

This could probably be the case as people prefer trains during the day and taxis for evening and late night commute.

(c) hourly number of taxi pick ups gradually increases after 6 AM till about 6 PM. It then starts decreasing till 5 AM.

Question 2 [25 pts]

In lecture we've seen k-Nearest Neighbors (k-NN) Regression, a non-parametric regression technique. In the following problems please use built in functionality from `sklearn` to run k-NN Regression.

2.1. Choose `TimeMin` as your feature variable and `PickupCount` as your response variable. Create a dictionary of `KNeighborsRegressor` objects and call it `KNNModels`. Let the key for your `KNNmodels` dictionary be the value of k and the value be the corresponding `KNeighborsRegressor` object. For $k \in \{1, 10, 75, 250, 500, 750, 1000\}$, fit k-NN regressor models on the training set (`train_data`).

2.2. For each k on the training set, overlay a scatter plot of the actual values of `PickupCount` vs. `TimeMin` with a scatter plot of **predictions** for `PickupCount` vs `TimeMin`. Do the same for the test set. You should have one figure with 2 x 7 total subplots; for each k the figure should have two subplots, one subplot for the training set and one for the test set.

Hints: 1. Each subplot should use different color and/or markers to distinguish k-NN regression prediction values from the actual data values. 2. Each subplot must have appropriate axis labels, title, and legend. 3. The overall figure should have a title.

2.3. Report the R^2 score for the fitted models on both the training and test sets for each k (reporting the values in tabular form is encouraged).

2.4. Plot, in a single figure, the R^2 values from the model on the training and test set as a function of k.

Hints: 1. Again, the figure must have axis labels and a legend. 2. Differentiate R^2 visualization on the training and test set by color and/or marker. 3. Make sure the k values are sorted before making your plot.

2.5. Discuss the results:

1. If n is the number of observations in the training set, what can you say about a k-NN regression model that uses $k = n$?

2. What does an R^2 score of 0 mean?

3. What would a negative R^2 score mean? Are any of the calculated R^2 you observe negative?
4. Do the training and test R^2 plots exhibit different trends? Describe.

5. How does the value of k affect the fitted model and in particular the training and test R^2 values?
6. What is the best value of k and what are the corresponding training/test set R^2 values?

1.1.3 Answers

2.1 Choose `TimeMin` as your feature variable and `PickupCount` as your response variable. Create a dictionary ...

```
In [11]: # inspect the data
         dftaxi.describe() # the TimeMin falls between 12 AM and 11:59 PM
         dftaxi.groupby(['TimeMin']).count().head() # there are duplicated entries with the sa
```

```
Out[11]:          PickupCount
         TimeMin
```

```
4.0                2
5.0                1
6.0                1
7.0                2
8.0                1
```

In [12]:
```
# your code here

#reshape x and y of the train and test data

def fun_reshape(train_data, test_data):

  x_train = np.array(train_data.TimeMin)
  x_train = x_train.reshape(-1, 1)

  y_train = np.array(train_data.PickupCount)
  y_train = y_train.reshape(-1, 1)

  x_test = np.array(test_data.TimeMin)
  x_test = x_test.reshape(-1, 1)

  y_test = np.array(test_data.PickupCount)
  y_test = y_test.reshape(-1, 1)

  return [x_train, y_train, x_test, y_test]
```

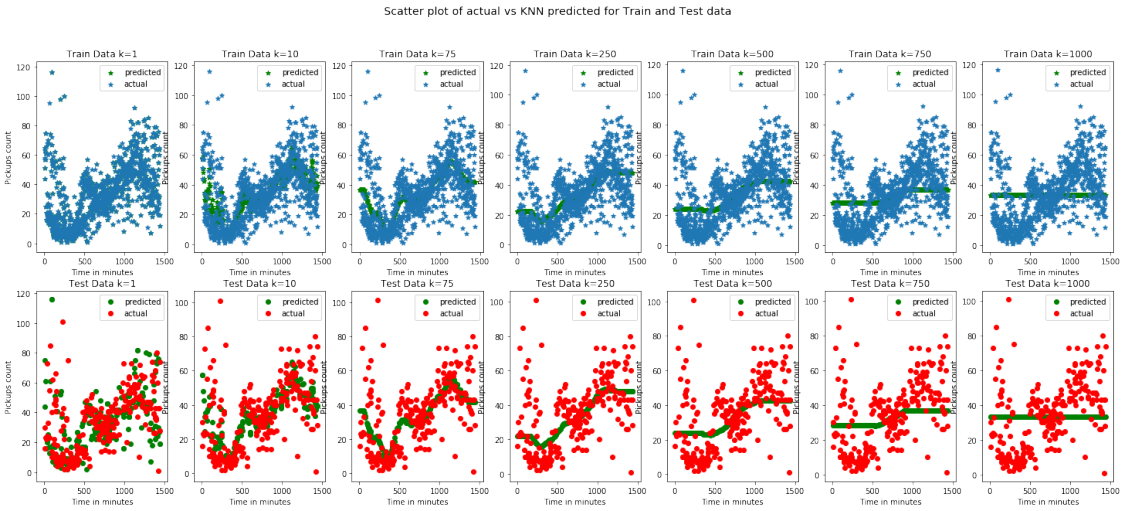
In [13]:
```
froutput = fun_reshape(train_data, test_data)
x_train = froutput[0]
y_train = froutput[1]

x_test = froutput[2]
y_test = froutput[3]
```

In [14]:
```
#your code here

# KNN Regression steps to predict Pickup Count values

KNNModels = {}   #empty dictionary of KNNModels
k_list = [1,10,75,250,500,750,1000]   #list of given k values to find the 'k' nearest
for k in k_list:
    knr = KNeighborsRegressor(n_neighbors=k)
    KNNModels[k] = knr.fit(x_train, y_train)
```

In [ ]:
```
#Check KNNModels dictionary
for key,values in KNNModels.items():
    print(key, values)
```

2.2 For each k on the training set, overlay a scatter plot ...

In [16]:
```
# your code here

# Plot predictions vs actual
## your code here

f, axarr = plt.subplots(2, 7, figsize=(25,10))
f.suptitle('Scatter plot of actual vs KNN predicted for Train and Test data', fontsize
count = 0
r2_scores = []
for key, value in KNNModels.items():

    pred_key = KNNModels[key].predict(x_train)
    axarr[0,count].scatter(x_train, pred_key, color='g', label='predicted', marker =
    axarr[0,count].legend(loc="upper right")          #legend for each subplot
    axarr[0,count].set_xlabel('Time in minutes')   #x-axis label
    axarr[0,count].set_ylabel('Pickups count')     #y-axis label
    r2_score_train = r2_score(y_train, pred_key)              #r2 score calculation

    axarr[0,count].scatter(x_train, y_train, label='actual', marker = '*')
    axarr[0,count].legend(loc="upper right")
    axarr[0,count].set_title('Train Data k=' + str(key))
    axarr[0,count].set_xlabel('Time in minutes')
    axarr[0,count].set_ylabel('Pickups count')
    axarr[0,count].grid(False)          #hide grid lines

    pred_key = KNNModels[key].predict(x_test)
    axarr[1,count].scatter(x_test, pred_key, color='g', label='predicted')
    axarr[1,count].legend(loc="upper right")
    axarr[1,count].set_xlabel('Time in minutes')
    axarr[1,count].set_ylabel('Pickups count')
    r2_score_test = r2_score(y_test, pred_key)                  #r2 score calculation

    axarr[1,count].scatter(x_test, y_test, label='actual', color='r')
    axarr[1,count].legend(loc="upper right")
    axarr[1,count].set_title('Test Data k=' + str(key))
    axarr[1,count].set_xlabel('Time in minutes')
    axarr[1,count].set_ylabel('Pickups count')
    axarr[1,count].grid(False)

    count+=1    # counter that increments to move along the values of k
    r2_scores.append([key, r2_score_train, r2_score_test])   # append the key and R2
```



2.3 Report the R^2 score for the fitted models ...

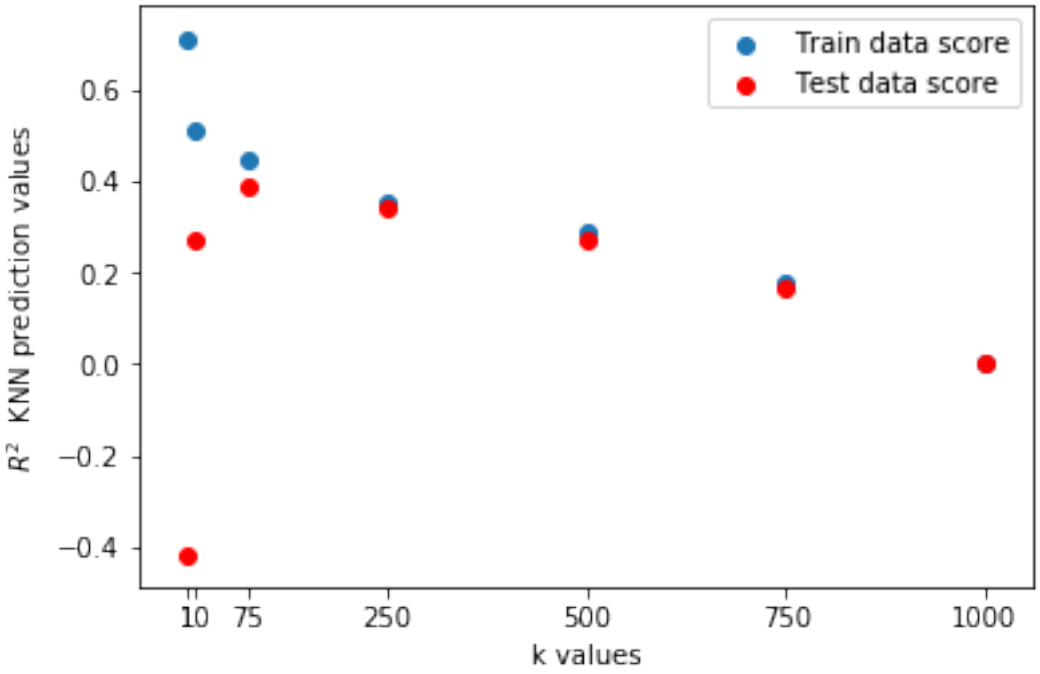
```
In [17]: # your code here
         r2df = pd.DataFrame(r2_scores, columns=["k", "train_data_score", "test_data_score"])
         r2df
```

Out[17]:

	k	train_data_score	test_data_score
0	1	0.712336	-0.418932
1	10	0.509825	0.272068
2	75	0.445392	0.390310
3	250	0.355314	0.340341
4	500	0.290327	0.270321
5	750	0.179434	0.164909
6	1000	0.000000	-0.000384

2.4 Plot, in a single figure, the R^2 values from the model on the training and test set as a function of k

```
In [18]: # your code here
         plt.scatter(r2df.k, r2df.train_data_score, label='Train data score')                #
         plt.scatter(r2df.k, r2df.test_data_score, color='r', label='Test data score')       #
         plt.xticks(r2df.k)
         plt.legend()
         plt.xlabel('k values')
         plt.ylabel(r'$R^2$  KNN prediction values')
         plt.grid(False)
```



2.5 Discuss the results

- your answer here*

1. If n is the number of observations in the training set, what can you say about a k-NN regression model that uses $k = n$?

If $k = n$ then the regression model will be the mean model which a poor model and doesn't predict correctly

2. What does an R^2 score of 0 mean?

R^2 score of 0 means that the regression model is equivalent to the mean model and is just a horizontal line parallel to x-axis

3. What would a negative R^2 score mean? Are any of the calculated R^2 you observe negative?

R^2 score is always positive on training data unless there is an error in the model. But we can have the R^2 score of test data as negative on a model built on train data. This happens during overfitting as it is the case for $k = 1, 1000$.

4. Do the training and test R^2 plots exhibit different trends? Describe.

Yes the training R^2 can be different from the test data R^2. This is seen for $k = 1, 10, 75$ where training data exhibits a better score than test.

5. How does the value of k affect the fitted model and in particular the training and test R^2 values?

When k has a low value between 1 and 10, R^2 score is better and is closer to 1. For large k values, R^2 score is closer to zero or the mean model which we would like to avoid.

6. What is the best value of k and what are the corresponding training/test set R^2 values?

k=1 is the best value of k as it fits the best model than others

Question 3 [25 pts]

We next consider simple linear regression, which we know from lecture is a parametric approach for regression that assumes that the response variable has a linear relationship with the predictor. Use the `statsmodels` module for Linear Regression. This module has built-in functions to summarize the results of regression and to compute confidence intervals for estimated regression parameters.

3.1. Again choose `TimeMin` as your predictor and `PickupCount` as your response variable. Create a `OLS` class instance and use it to fit a Linear Regression model on the training set (`train_data`). Store your fitted model in the variable `OLSModel`.

3.2. Re-create your plot from 2.2 using the predictions from `OLSModel` on the training and test set. You should have one figure with two subplots, one subplot for the training set and one for the test set.

Hints: 1. Each subplot should use different color and/or markers to distinguish Linear Regression prediction values from that of the actual data values. 2. Each subplot must have appropriate axis labels, title, and legend. 3. The overall figure should have a title.

3.3. Report the R^2 score for the fitted model on both the training and test sets.

3.4. Report the slope and intercept values for the fitted linear model.

3.5. Report the 95% confidence interval for the slope and intercept.

3.6. Create a scatter plot of the residuals ($e = y - \hat{y}$) of the linear regression model on the training set as a function of the predictor variable (i.e. `TimeMin`). Place on your plot a horizontal line denoting the constant zero residual.

3.7. Discuss the results:

1. How does the test R^2 score compare with the best test R^2 value obtained with k-NN regression?
2. What does the sign of the slope of the fitted linear model convey about the data?

3. Based on the 95% confidence interval, do you consider the estimates of the model parameters to be reliable?

4. Do you expect 99% confidence intervals for the slope and intercept to be tighter or wider than the 95% confidence intervals? Briefly explain your answer.

5. Based on the residuals plot that you made, discuss whether or not the assumption of linearity is valid for this data.
6. Based on the data structure, what restriction on the model would you put at the endpoints (at $x \approx 0$ and $x \approx 1440$)? What does this say about the linearity assumption?

1.1.4 Answers

3.1 Again choose `TimeMin` as your predictor and `PickupCount` as your response variable. Create a `OLS` class instance ...

In [19]:
```
# your code here

# Transform the predictor pandas series into a np array
# Invoke function on converting dataframe to numpy array and reshape --- written in 2

froutput = fun_reshape(train_data, test_data)
x_train = froutput[0]
y_train = froutput[1]

x_test = froutput[2]
y_test = froutput[3]
```

In [20]:
```
# your code here

# Regression OLS function
def OLS_reg(x_train, y_train):

  # create the X matrix by appending a column of ones to x_train
  X = sm.add_constant(x_train)

  # build the OLS model (ordinary least squares) from the training data
  osl = sm.OLS(y_train, X)

  # do the fit and save regression info (parameters, etc) in results_sm
  # save the fitted model in OLSModel variable
  OLSModel = osl.fit()
  return OLSModel
```

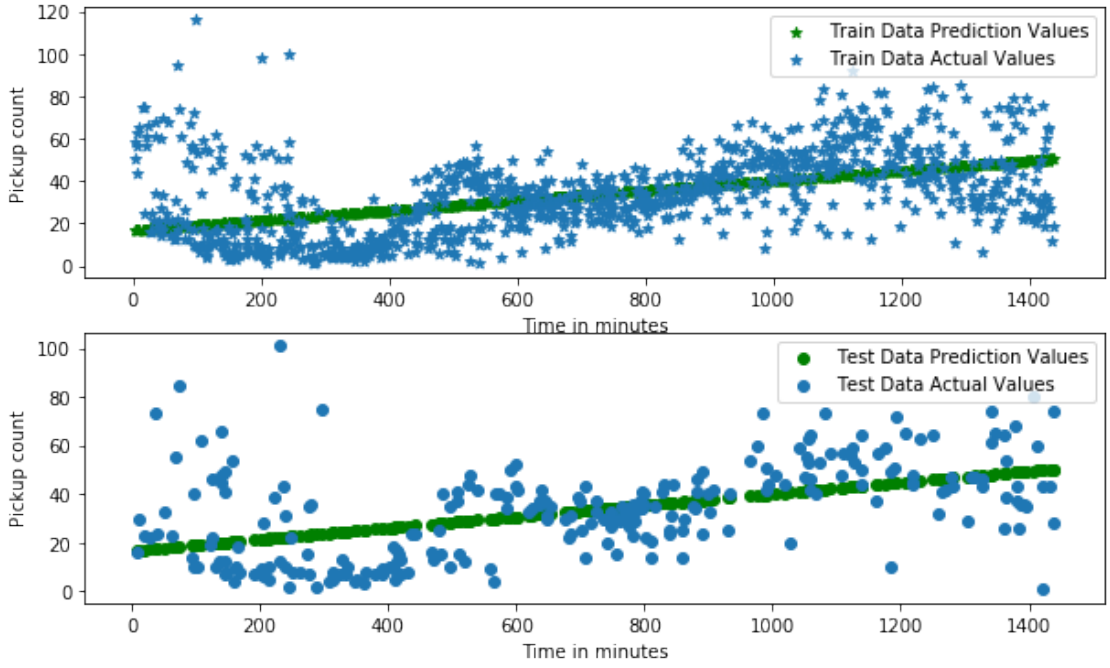
In [21]:
```
# your code here
OLSModel = OLS_reg(x_train, y_train)
```

3.2 Re-create your plot from 2.2 using the predictions from `OLSModel` on the training and test set ...

In [23]:
```
# your code here

r2_scores_osl = []
fig_scat, ax_scat = plt.subplots(2,1, figsize=(10,6))
fig_scat.suptitle('Scatter plot of train and test data with actual and OLS model predi

X = sm.add_constant(x_train)
y_train_predicted = OLSModel.predict(X)
ax_scat[0].scatter(x_train, y_train_predicted, color = 'g', marker = '*', label='Trai
ax_scat[0].set_xlabel('Time in minutes')
ax_scat[0].set_ylabel('Pickup count')
ax_scat[0].legend(loc="upper right")
r2_score_osl_train = r2_score(y_train, y_train_predicted)

ax_scat[0].scatter(x_train, y_train, marker = '*', label='Train Data Actual Values')
```

```
ax_scat[0].set_xlabel('Time in minutes')
ax_scat[0].set_ylabel('Pickup count')
ax_scat[0].legend(loc="upper right")
ax_scat[0].grid(False)


# Plot best-fit line for test
X = sm.add_constant(x_test)
y_test_predicted = OLSModel.predict(X)
ax_scat[1].scatter(x_test, y_test_predicted, color = 'g', label='Test Data Prediction
ax_scat[1].set_xlabel('Time in minutes')
ax_scat[1].set_ylabel('Pickup count')
ax_scat[1].legend(loc="upper right")
r2_score_osl_test = r2_score(y_test, y_test_predicted)

ax_scat[1].scatter(x_test, y_test, label='Test Data Actual Values')
ax_scat[1].set_xlabel('Time in minutes')
ax_scat[1].set_ylabel('Pickup count')
ax_scat[1].legend(loc="upper right")
ax_scat[1].grid(False)

r2_scores_osl.append([r2_score_osl_train, r2_score_osl_test])
```

Scatter plot of train and test data with actual and OLS model predicted values



3.3 Report the R^2 score for the fitted model on both the training and test sets.

```
In [24]: # your code here

         r2df = pd.DataFrame(r2_scores_osl, columns=["r2_score_osl_train", "r2_score_osl_test"]
         r2df
```

Out[24]:

	r2_score_osl_train	r2_score_osl_test
0	0.243026	0.240662

3.4 Report the slope and intercept values for the fitted linear model.

```
In [25]: # your code here

         # pull the beta parameters out from results_sm
         beta0_sm = OLSModel.params[0]
         beta1_sm = OLSModel.params[1]

         print("The regression coefficients from the statsmodels package are: beta_0 = {0:8.6f}
```

```
The regression coefficients from the statsmodels package are: beta_0 = 16.750601 and beta_1 = (
```

3.5 Report the 95% confidence interval for the slope and intercept.

```
In [26]: # your code here
         print("95% Confidence interval for slope is:", OLSModel.conf_int(0.05)[0], "\n95% Con
```

```
95% Confidence interval for slope is: [14.67514134 18.82606151]
95% Confidence interval for intercept is: [0.02077697 0.02589338]
```

3.6 Create a scatter plot of the residuals

```
In [27]: # your code here
         fig, ax = plt.subplots(1, 1, figsize=(8,8))
         y_bar = np.mean(y_train)
         e = y_train - y_bar
         ax.scatter(x_train, e)
         ax.axhline(0, 0, 1, color='r')
         ax.set_ylabel('Residuals in Pickup count')
         ax.set_xlabel('Time in minutes')
         ax.set_title('Scatter plot of the residuals on training set', fontsize = 'x-large')
```

```
Out[27]: Text(0.5,1,'Scatter plot of the residuals on training set')
```

3.7 Discuss the results:

your answer here 1. How does the test R^2 score compare with the best test R^2 value obtained with k-NN regression?

On the test data set, we ontained $R^2 = 0.24066$ using the linear model which is less than $R^2 = 0.39$ obtained using the best k-NN model ($k = 75$).

2. What does the sign of the slope of the fitted linear model convey about the data?

Slope has a positive sign which conveys positive correlation between the feature and response variable. That is if the feature variable increases by one unit then the response variable also increases by the function f(x)+epsilon

3. Based on the 95% confidence interval, do you consider the estimates of the model parameters to be reliable?

Yes, I consider the model parameters to be reliable for 2 reasons: (a) The confidence interval doesn't contain zero, so we can be confident that there will be no change in direction of the beta parameters (b) The confidence interval contains our beta parameters

4. Do you expect 99% confidence intervals for the slope and intercept to be tighter or wider than the 95% confidence intervals? Briefly explain your answer.

99% confidence intervals for the slope and intercept will be wider than 95% confidence intervals because 95% confidence intervals is at 2 standard deviations away from the mean while 99% confidence intervals is at 3 standard deviations away from the mean.

5. Based on the residuals plot that you made, discuss whether or not the assumption of linearity is valid for this data.

The residuals plot shows a wave, something like a quadratic line. Although some residuals are positive, some negative, for each x the residuals are somehow not evenly distributed arround zero, especially at the endpoints. Therefore the assumption of linearity for this data seems to be undermined.

6. Based on the data structure, what restriction on the model would you put at the endpoints (at $x \approx 0$ and $x \approx 1440$)? What does this say about the linearity assumption?

If we remove data points at the endpoints for the training data, then the model would produce a data structure of residuals that does not have a quadratic shape any more. The model would become $\hat{f}(x)$ for $x >\approx 0$ and $x <\approx 1440$. The linearity assumption based on the scatter plot of data together with the best fit line is not a certainty until it is confirmed by an unstructured residuals distribution.

Outliers

You may recall from lectures that OLS Linear Regression can be susceptible to outliers in the data. We're going to look at a dataset that includes some outliers and get a sense for how that affects modeling data with Linear Regression. **Note, this is an open-ended question, there is not one correct solution (or one correct definition of an outlier).**

Question 4 [25 pts]

4.1. We've provided you with two files `outliers_train.txt` and `outliers_test.txt` corresponding to training set and test set data. What does a visual inspection of training set tell you about the existence of outliers in the data?

4.2. Choose `X` as your feature variable and `Y` as your response variable. Use `statsmodel` to create a Linear Regression model on the training set data. Store your model in the variable `OutlierOLSModel`.

4.3. You're given the knowledge ahead of time that there are 3 outliers in the training set data. The test set data doesn't have any outliers. You want to remove the 3 outliers in order to get the optimal intercept and slope. In the case that you're sure of the existence and number (3) of outliers ahead of time, one potential brute force method to outlier detection might be to find the best Linear Regression model on all possible subsets of the training set data with 3 points removed. Using this method, how many times will you have to calculate the Linear Regression coefficients on the training data?

4.4 In CS109 we're strong believers that creating heuristic models is a great way to build intuition. In that spirit, construct an approximate algorithm to find the 3 outlier candidates in the training data by taking advantage of the Linear Regression residuals. Place your algorithm in the

function `find_outliers_simple`. It should take the parameters `dataset_x` and `dataset_y` representing your features and response variable values (make sure your response variable is stored as a numpy column vector). The return value should be a list `outlier_indices` representing the indices of the 3 outliers in the original datasets you passed in. Remove the outliers that your algorithm identified, use `statsmodels` to create a Linear Regression model on the remaining training set data, and store your model in the variable `OutlierFreeSimpleModel`.

4.5 Create a figure with two subplots: the first is a scatterplot where the color of the points denotes the outliers from the non-outliers in the training set, and include two regression lines on this scatterplot: one fitted with the outliers included and one fitted with the outlier removed (all on the training set). The second plot should include a scatterplot of points from the test set with the same two regression lines fitted on the training set: with and without outliers. Visually which model fits the test set data more closely?

4.6. Calculate the R^2 score for the `OutlierOLSModel` and the `OutlierFreeSimpleModel` on the test set data. Which model produces a better R^2 score?

4.7. One potential problem with the brute force outlier detection approach in 4.3 and the heuristic algorithm you constructed 4.4 is that they assume prior knowledge of the number of outliers. In general you can't expect to know ahead of time the number of outliers in your dataset. Alter the algorithm you constructed in 4.4 to create a more general heuristic (i.e. one which doesn't presuppose the number of outliers) for finding outliers in your dataset. Store your algorithm in the function `find_outliers_general`. It should take the parameters `dataset_x` and `dataset_y` representing your features and response variable values (make sure your response variable is stored as a numpy column vector). It can take additional parameters as long as they have default values set. The return value should be the list `outlier_indices` representing the indices of the outliers in the original datasets you passed in (in the order of 'severity'). Remove the outliers that your algorithm identified, use `statsmodels` to create a Linear Regression model on the remaining training set data, and store your model in the variable `OutlierFreeGeneralModel`.

Hints: 1. How many outliers should you try to identify in each step? 2. If you plotted an R^2 score for each step the algorithm, what might that plot tell you about stopping conditions?

4.8. Run your algorithm in 4.7 on the training set data.
1. What outliers does it identify? 2. How do those outliers compare to the outliers you found in 4.4? 3. How does the general outlier-free Linear Regression model you created in 4.7 perform compared to the simple one in 4.4?

1.1.5 Answers

4.1 We've provided you with two files `outliers_train.txt` and `outliers_test.txt` corresponding to training set and test set data. What does a visual inspection of training set tell you about the existence of outliers in the data?

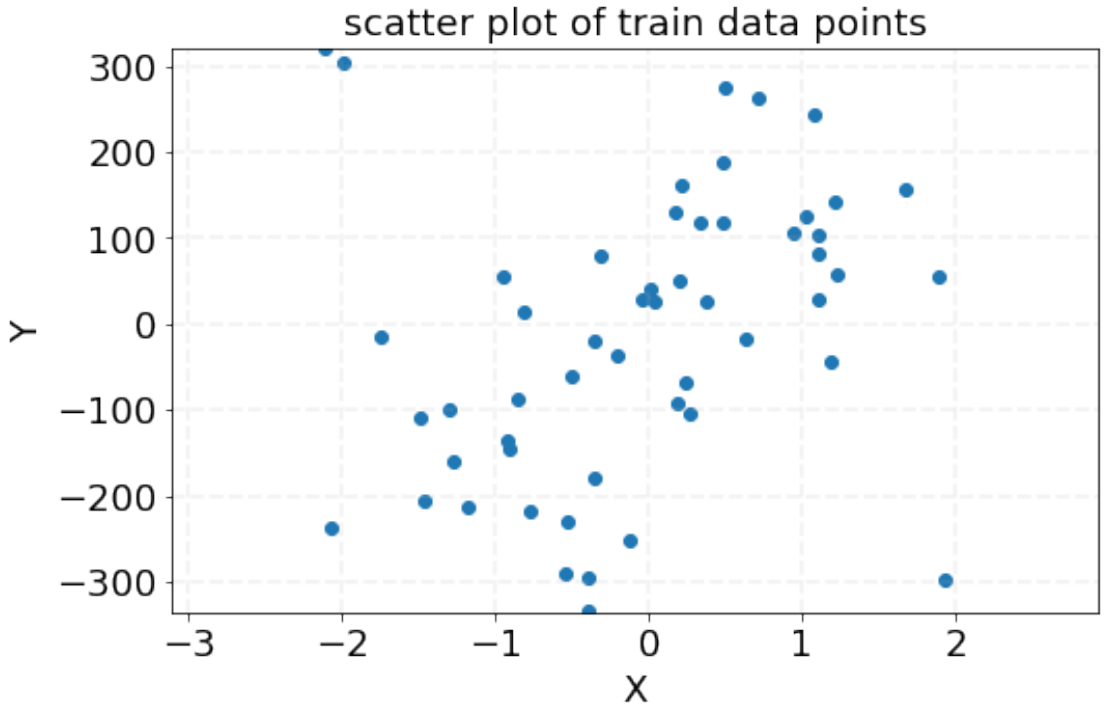
```
In [28]: # read the data
         # your code here

         dfouttrain = pd.read_csv('data/outliers_train.csv')
         dfouttest = pd.read_csv('data/outliers_test.csv')
```

```
In [29]: # your code here
         # obtaining x and y from the dataframe
         train_x = np.array(dfouttrain['X'])
         train_y = np.array(dfouttrain['Y'])
```

In [30]:
```
# your code here
test_x = np.array(dfouttest['X'])
test_y = np.array(dfouttest['Y'])
```

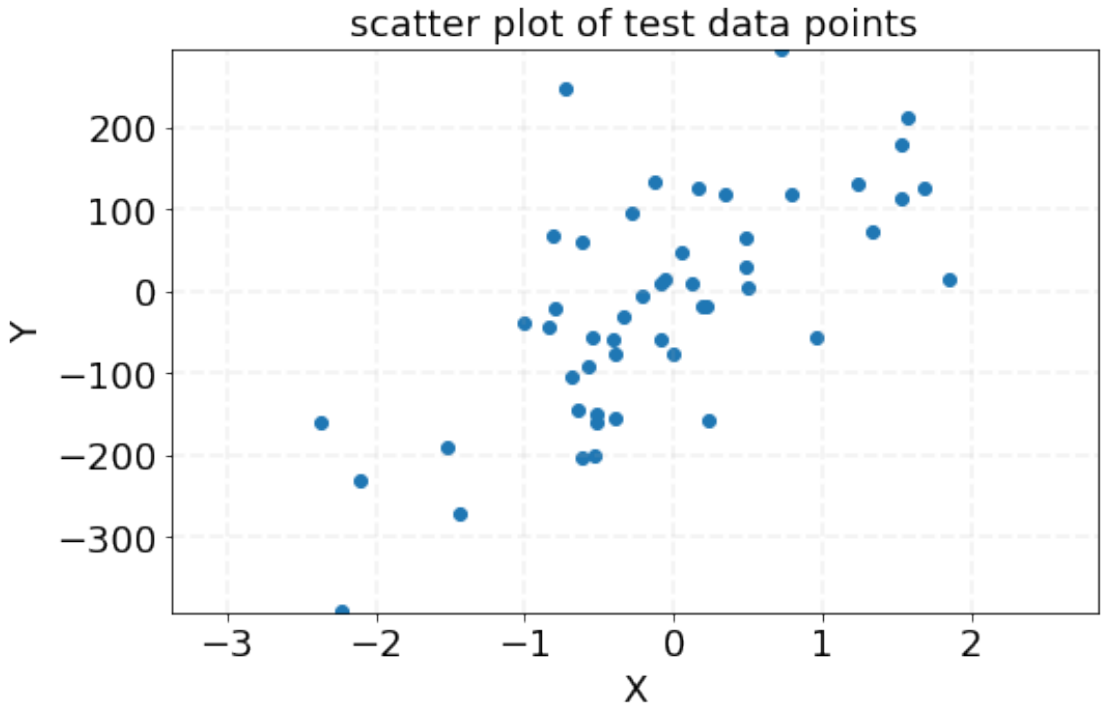
In [35]:
```
# your code here

gen_scatterplot(train_x, train_y, 'X', 'Y', 'scatter plot of train data points')
```

Out[35]: <matplotlib.axes._subplots.AxesSubplot at 0x1fc6762d400>



In [36]:
```
# your code here
gen_scatterplot(test_x, test_y, 'X', 'Y', 'scatter plot of test data points')
```

Out[36]: <matplotlib.axes._subplots.AxesSubplot at 0x1fc69189cf8>



your answer here

Visual inspection tells us there are 2 outliers around x=-2 on the train data set and 1 outlier around x=2

We can remove these outliers by taking out the top 3 maximum distance points from the best fit line.

4.2 Choose `X` as your feature variable and `Y` as your response variable. Use `statsmodel` to create ...

```
In [37]: import warnings
         warnings.filterwarnings('ignore')

In [38]: OutlierOLSModel = OLS_reg(train_x, train_y)    # invoke function OLS_reg to run statsm

In [ ]: #check your outlierOLSmodel variable
        print(OutlierOLSModel.summary())
```

4.3 One potential brute force method to outlier detection might be to find the best Linear Regression model on all possible subsets of the training set data with 3 points removed. Using this method, how many times will you have to calculate the Linear Regression coefficients on the training data?

your answer here

Using brute force method, we can remove the outliers in 53C3 combinations that is by calculating the regression coeffecients 27,720 times.

56C3 = 56! / (3! * 53!) = 27,720

There are 53 data points in total. For each subset of 3 data points chosen without replacement, we will need to evaluate the linear model. The number of times we need to do this equals the

number of subsets of 3 elements we can build out of 53 elements. From algebreic combinatoric, this is the binomial coefficient $\binom{53}{3}$, what gives 23426

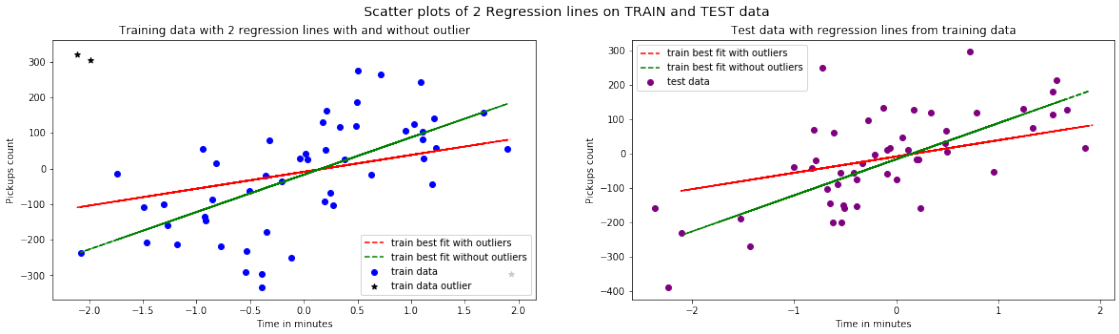
4.4 CS109 hack ...

```
In [41]: # get outliers
         # your code here

         def find_outliers_simple(dataset_x, dataset_y):
             # your code here

             #reshape dataset
             dataset_x = dataset_x.reshape(-1, 1)
             dataset_y = dataset_y.reshape(-1, 1)

             #add constant to save a place for beta0 in regression equation
             X = sm.add_constant(dataset_x)

             #OutlierOLSModel variable is set to Statsmodel OLS regression output
             OutlierOLSModel = OLS_reg(dataset_x, dataset_y)

             #computing y predicted values
             dataset_y_predicted = OutlierOLSModel.predict(X)
             dataset_y_predicted = dataset_y_predicted.reshape(-1, 1)    #reshape dataset_y_pr

             residuals =  (dataset_y_predicted - dataset_y)**2    #compute residuals as a squa

             residuals = np.reshape(residuals, (1, np.product(residuals.shape)))[0]   #reshape
             return residuals.argsort()[-3:]   #return a list of top 3 residuals or top 3 dist

In [42]: # your code here
         # check output of the outliers function
         outliers_3 = find_outliers_simple(train_x, train_y)
         print("3 outliers where found at indices:")
         outliers_3

3 outliers where found at indices:


Out[42]: array([52, 51, 50], dtype=int64)
```

4.5 Create a figure with two subplots: the first is a scatterplot ...

```
In [43]: f, axarr = plt.subplots(1, 2, figsize=(20,5))
         f.suptitle('Scatter plots of 2 Regression lines on TRAIN and TEST data', fontsize='x-l

         train_x_woo = np.delete(train_x, outliers_3)   # x-array after deleting outliers
         train_y_woo = np.delete(train_y, outliers_3)   # y-array after deleting outliers

         ### SCATTER PLOT 1 ON TRAIN DATA ###
```

```
# create a scatter plot of the training data
axarr[0].set_title('Training data with 2 regression lines with and without outlier')
axarr[0].scatter(train_x_woo, train_y_woo, color='b', label='train data')    #plot da
axarr[0].set_xlabel('Time in minutes')
axarr[0].set_ylabel('Pickups count')

# overlay a scatter plot of the outliers
axarr[0].scatter(train_x[outliers_3], train_y[outliers_3], color='black', label='trai

# create a regression line on training data with outliers included
OutlierOLSModel = OLS_reg(train_x, train_y)
X = sm.add_constant(train_x)
best_fit_line = OutlierOLSModel.predict(X)
best_fit_line = best_fit_line.reshape(best_fit_line.shape[0],1)
axarr[0].plot(train_x, best_fit_line, ls='--', color ='r', label='train best fit with

# create a regression line on training data with outliers removed
OutlierFreeSimpleModel = OLS_reg(train_x_woo, train_y_woo)
X = sm.add_constant(train_x_woo)
best_fit_woo = OutlierFreeSimpleModel.predict(X)
best_fit_woo = best_fit_woo.reshape(best_fit_woo.shape[0],1)
axarr[0].plot(train_x_woo, best_fit_woo, ls='--', color='g', label='train best fit wi
axarr[0].legend(loc="best")
axarr[0].grid(False)

### SCATTER PLOT 2 ON TEST DATA ###

# create a scatter plot of the test data
axarr[1].set_title('Test data with regression lines from training data')
axarr[1].set_xlabel('Time in minutes')
axarr[1].set_ylabel('Pickups count')
axarr[1].scatter(test_x, test_y, color='purple', label='test data')
# create a regression line on training data with outliers included
axarr[1].plot(train_x, best_fit_line, ls='--', color ='r', label='train best fit with
# create a regression line on training data with outliers removed
axarr[1].plot(train_x_woo, best_fit_woo, ls='--', color='g', label='train best fit wi
axarr[1].legend(loc="best")
axarr[1].grid(False)
```



your answer here

By visual inspection of the 2 above subplots, best fit line from outlier free model is a good fit for the test data

4.6 Calculate the R^2 score for the `OutlierOLSModel` and the `OutlierFreeSimpleModel` on the test set data. Which model produces a better R^2 score?

```
In [44]: X = sm.add_constant(test_x)    # add constant to store a place for beta0

         # test score with outlier
         test_y_predicted = OutlierOLSModel.predict(X)
         r2_score_test = r2_score(test_y, test_y_predicted)

         test_y_predicted_woo = OutlierFreeSimpleModel.predict(X)
         r2_score_test_woo = r2_score(test_y, test_y_predicted_woo)

         print("The score for the model trained without outliers is {0:8.6f}, which is better t

The score for the model trained without outliers is 0.452957, which is better than the score fo
```

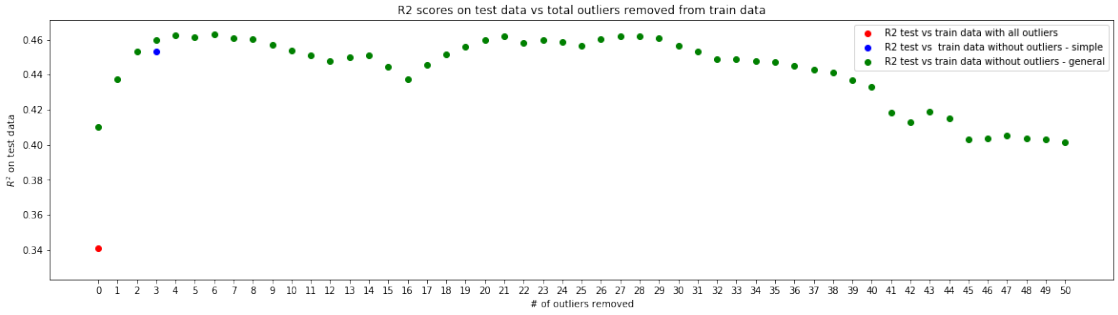
4.7 One potential problem with the brute force outlier detection approach in 4.3 and the heuristic algorithm you constructed 4.4 is that they assume prior knowledge of the number of outliers.

```
In [45]: # your code here
         # general heuristic algorithm for outliers identification
         # full_run = True means that all data points are returned as outliers sorted by sever
         #                 This is used for identifying the stopping condition
         def find_outliers_general(dataset_x, dataset_y, full_run = False):
             # your code here
             outliers = []   #empty list --- used for storing list of outliers
             LENGTH = len(dataset_x)
             r2_previous = 0
             r2_current = 0.001
             i = 0
             # loop until there is no improvement of r2 score by removing candidate outliers
```

```
    while (full_run or r2_previous < r2_current) and (i < LENGTH - 2):
      # create a linear regression model on the current data
      dataset_x = dataset_x.reshape(-1, 1)
      dataset_y = dataset_y.reshape(-1, 1)
      X = sm.add_constant(dataset_x)
      OutlierOLSModel = OLS_reg(dataset_x, dataset_y)  #invoke function to compute OL

      # computer Y predicted values and calculate the residuals
      dataset_y_pred = OutlierOLSModel.predict(X)
      dataset_y_pred = dataset_y_pred.reshape(-1, 1)
      residuals =  (dataset_y_pred - dataset_y)**2    #compute residuals for each dat

      # identify the data point with highest residual as outlier
      residuals = np.reshape(residuals, (1, np.product(residuals.shape)))[0]   #resha
      outlier = residuals.argsort()[-1]   #return maximum distant point as residual

      outliers.append(outlier+i)   #append every outlier to the list
      r2_previous = r2_current
      r2_current = r2_score(dataset_y, dataset_y_pred)

      # remove the outlier from the data
      dataset_x = np.delete(dataset_x, outlier)  # delete outliers and re-run the alg
      dataset_y = np.delete(dataset_y, outlier)
      i = i + 1           #increment while loop condition for next iteration

    return outliers
```

In [46]:
```
# Function for plotting outliers
def outliers_plot(outliers):
  # Calculate R2 score for each step of the algorithm on test data
  R_2_test_woo_general_all = []
  train_x_gwoo = np.copy(train_x)
  train_y_gwoo = np.copy(train_y)
  i = 0
  # for each set of outliers, estimate the R2 score of a linear regression model on t
  for outlier in outliers:
    # remove the outlier from training data
    train_x_gwoo = np.delete(train_x_gwoo, outlier - i)
    train_y_gwoo = np.delete(train_y_gwoo, outlier - i)
    # create a linear regression model on the remaining training data

    # OutlierFreeGeneralModel

    OutlierFreeGeneralModel = OLS_reg(train_x_gwoo, train_y_gwoo)   #invoke OLS regre
    # calculate the R2 score of the model on test data
    test_y_predicted_woo_general = OutlierFreeGeneralModel.predict(sm.add_constant(te
    test_y_predicted_woo_general = test_y_predicted_woo_general.reshape(-1, 1)
    R_2_test_woo_general = r2_score(test_y, test_y_predicted_woo_general)
```

```
        # store the R2 score
        R_2_test_woo_general_all.append(R_2_test_woo_general)
        i = i + 1

    # create a plot of r2 scores
    r2df = pd.DataFrame(R_2_test_woo_general_all, columns=["r2"])

    #scatter plot of R2 score on test data
    f, ax = plt.subplots(1, 1, figsize=(20,5))
    ax.scatter(0, r2_score_test, color = 'r', label = "R2 test vs train data with all ou
    ax.scatter(3, r2_score_test_woo, color = 'b', label = "R2 test vs  train data withou
    ax.scatter(range(r2df.r2.count()), r2df.r2, color = "g", label = "R2 test vs train d
    ax.set_xticks(range(r2df.r2.count()))   #values of r2 score
    ax.legend()
    ax.set_xlabel("# of outliers removed")
    ax.set_ylabel("$R^2$ on test data")
    ax.set_title("R2 scores on test data vs total outliers removed from train data")
    ax.grid(False)


outliers_general_full = find_outliers_general(train_x, train_y, True)
outliers_plot(outliers_general_full)
```



1. How many outliers should you try to identify in each step?

The plot suggest to identify one outlier at each step.

2. If you plotted an R^2 score for each step the algorithm, what might that plot tell you about stopping conditions?

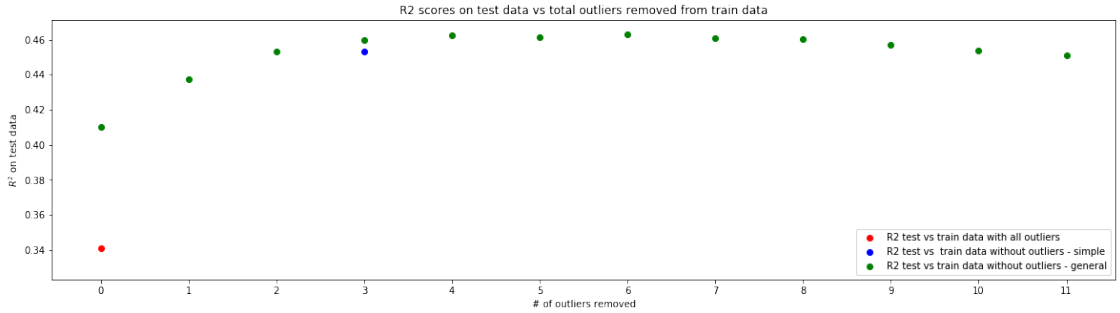
The plot suggests to stop when R2 improvment by removing outliers is not significant.

```
In [47]: # Run algorithm on the training set data
         outliers_general = find_outliers_general(train_x, train_y)
         # Create regression model on remaining training set data and store the model as Outli
         train_x_gwoo = np.delete(train_x, outliers_general)
         train_y_gwoo = np.delete(train_y, outliers_general)
         OutlierFreeGeneralModel = OLS_reg(train_x_gwoo, train_y_gwoo)   #invoke OLS regressio
```

4.8 Run your algorithm in 4.7 on the training set data

```
In [48]: # your code here

         # Plot outliers

         outliers_plot(outliers_general)

         # find and print all outliers in training set data
         print("The following outliers where found: {}".format(outliers_general))
```

```
The following outliers where found: [50, 51, 52, 4, 17, 31, 10, 38, 29, 26, 15, 44]
```



your answer here
Question 4.8:

2. How do those outliers compare to the outliers you found in 4.4?

We seem to detect more outliers which are subtle in nature when we ran the outliers_general algorithm. Whereas when we tried to run a scatter plot and find the 3 outliers it was only a small number of outliers being detected.

3. How does the general outlier-free Linear Regression model you created in 4.7 perform compared to the simple one in 4.4?

OutlierFreeSimple model does not detect all the outliers present in the training data while the General model is able to detect all outliers. Therefore once all outliers detected by general model is removed we can have the best R2 score for the training set data with the best prediction model