CS109A Introduction to Data Science:

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
HTML(styles)
```
Out[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
```

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?

Answers

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

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

Out[3]:
```
(1250, 2)
```

In [4]:
```
# split the data
# your code here
train_data, test_data = train_test_split(data, test_size=0.2, random_state=42)
```

In [5]:
```
# your code here
train_data.shape, test_data.shape
```

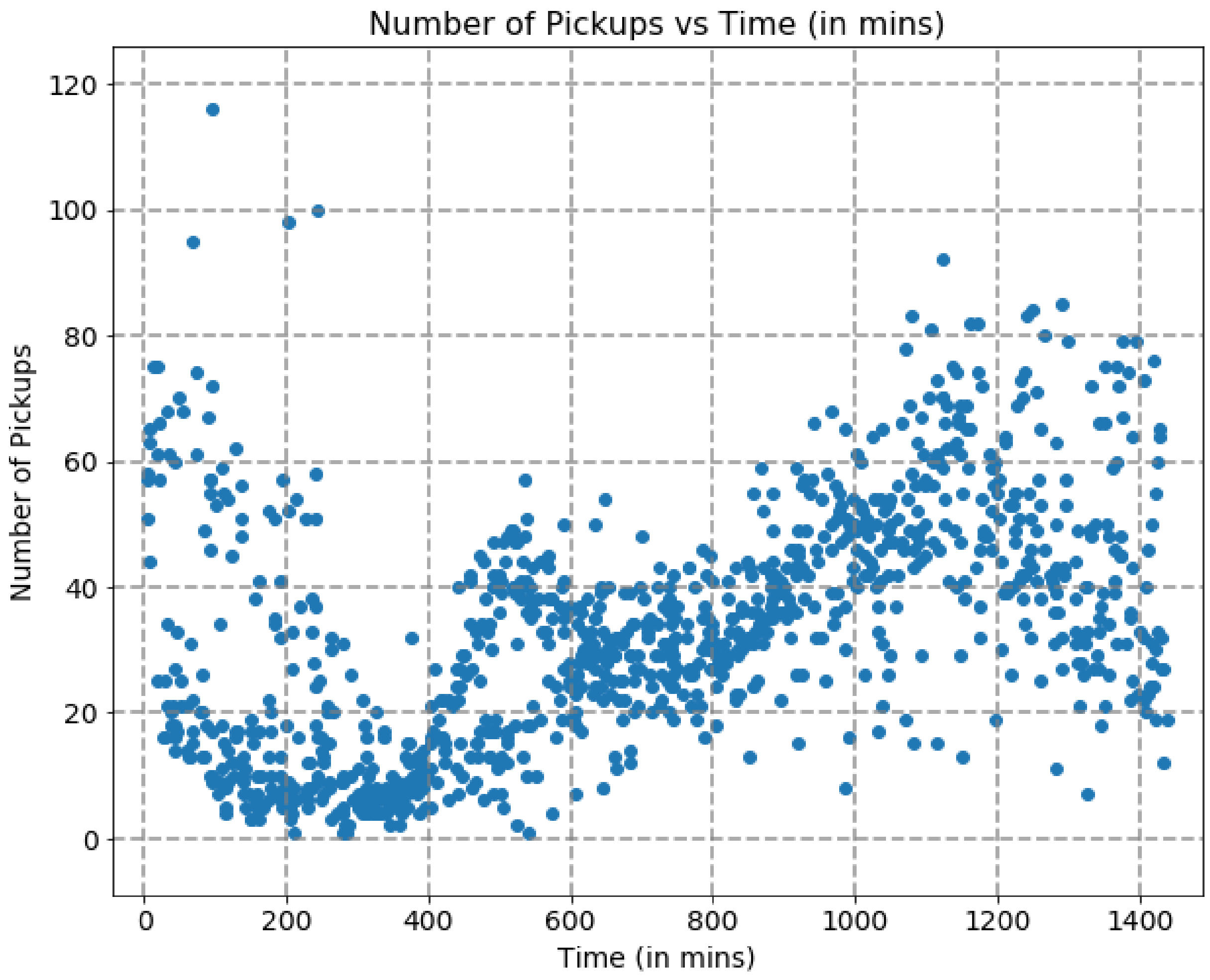
Out[5]:
```
((1000, 2), (250, 2))
```

In [6]:
```
# Test size is indeed 20% of total
# your code here
len(test_data)/len(data)
```

Out[6]:
```
0.2
```

1.2 Generate a scatter plot of the training data points

In [7]:
```
# Your code here
fig, ax = plt.subplots(1, 1, figsize = (10,8))
ax.scatter(train_data.TimeMin, train_data.PickupCount)
ax.set_xlim(min(train_data.TimeMin)-50, max(train_data.TimeMin)+50)
ax.set_ylim(min(train_data.PickupCount)-10, max(train_data.PickupCount)+10)
ax.set_xlabel('Time (in mins)', fontsize = 14)
ax.set_ylabel('Number of Pickups', fontsize = 14)
ax.set_title('Number of Pickups vs Time (in mins)', fontsize = 16)
ax.grid(True, lw = 1.75, ls = '--', color ='grey', alpha = 0.75)
ax.tick_params(labelsize = 14)
```



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

Answer: *Overall we see that as TimeMin increases, PickupCount also increases. More specifically within the range* $200 <= TimeMin <= 1200$. *For* $TimeMin < 200$ *and* $TimeMin > 1200$*, as TimeMin increases, PickupCount also decreases. We also see some outliers in the data, specifically for* $PickupCounts > 40$ *and* $TimeMin = 0$

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?

Answers

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

In [8]:
```
# Define Test and Train data
X_train = train_data[['TimeMin']]
y_train = train_data[['PickupCount']]
X_test = test_data[['TimeMin']]
y_test = test_data[['PickupCount']]
```

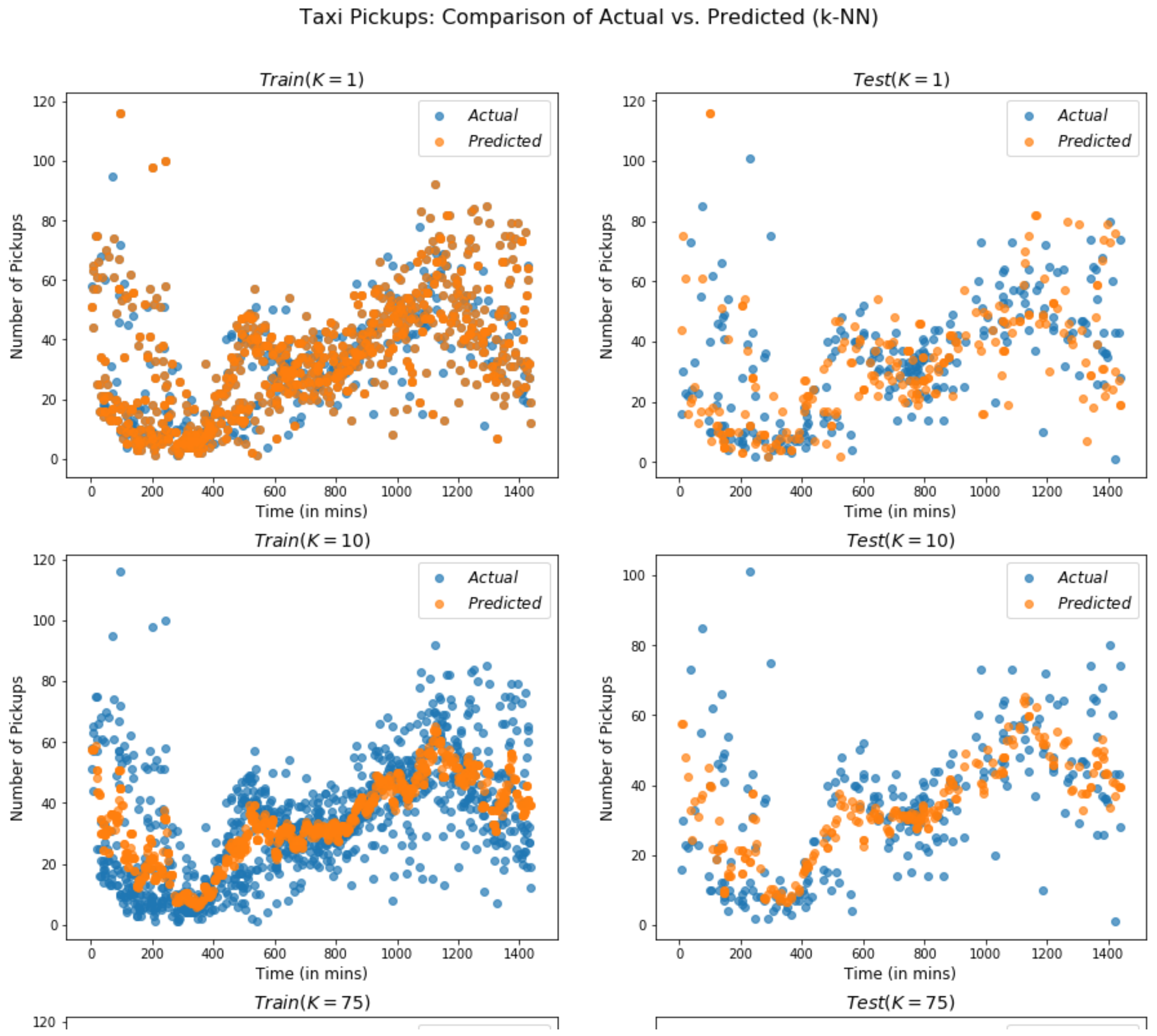
In [9]:
```
# your code here
KNNModels = {}
for k in (1, 10, 75, 250, 500, 750, 1000):
    knnreg = KNeighborsRegressor(n_neighbors=k)
    KNNModels[k] = knnreg.fit(X_train, y_train)

KNNModels.keys()
```

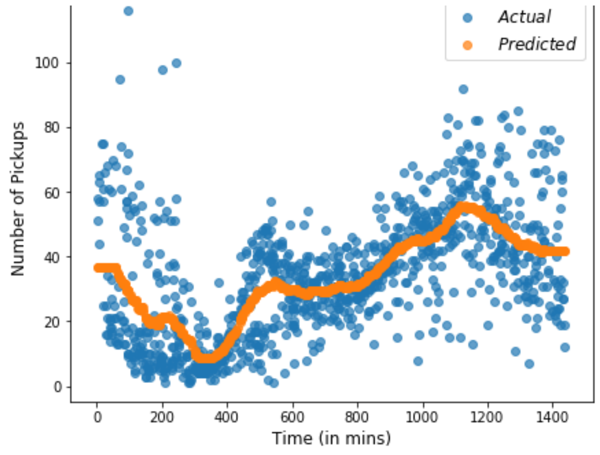
Out[9]:
```
dict_keys([1, 10, 75, 250, 500, 750, 1000])
```

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

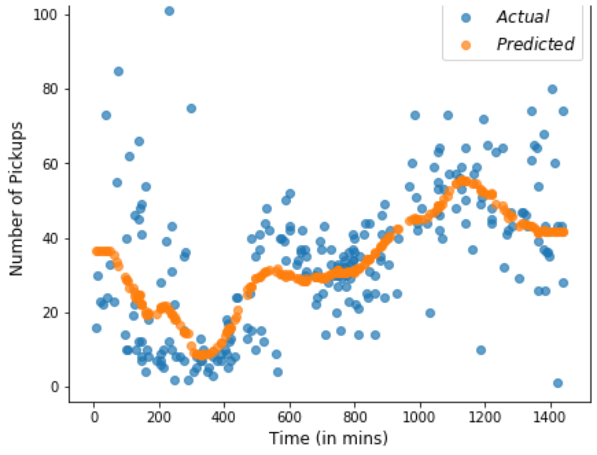
In [10]:
```python
# your code here

fig_knn, ax_knn = plt.subplots(7, 2, figsize = (15,45))

i=0
for k in KNNModels.keys():
    predict_test = KNNModels[k].predict(X_test)
    predict_train = KNNModels[k].predict(X_train)
    for j in range(2):
        X = X_train if j==0 else X_test
        y = y_train if j==0 else y_test
        predict = predict_train if j == 0 else predict_test
        label = 'Train' if j==0 else 'Test'
        ax_knn[i,j].set_xlabel('Time (in mins)', fontsize=12)
        ax_knn[i,j].set_ylabel('Number of Pickups', fontsize=12)
        ax_knn[i,j].scatter(X, y, label=r'$Actual$', alpha = 0.7)
        ax_knn[i,j].scatter(X, predict, label=r'$Predicted$', alpha = 0.7)
        ax_knn[i,j].set_title(r'$%s(K=%s)$'%(label,k), fontsize=14)
        ax_knn[i,j].legend(loc='best', fontsize=12)
    i=i+1
fig.tight_layout(h_pad=20)
fig_knn.suptitle('Taxi Pickups: Comparison of Actual vs. Predicted (k-NN)', y = 0
```

Out[10]:
```
Text(0.5,0.9,'Taxi Pickups: Comparison of Actual vs. Predicted (k-NN)')
```

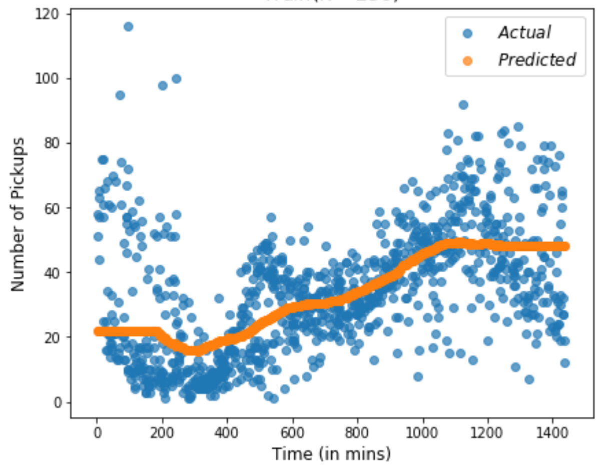




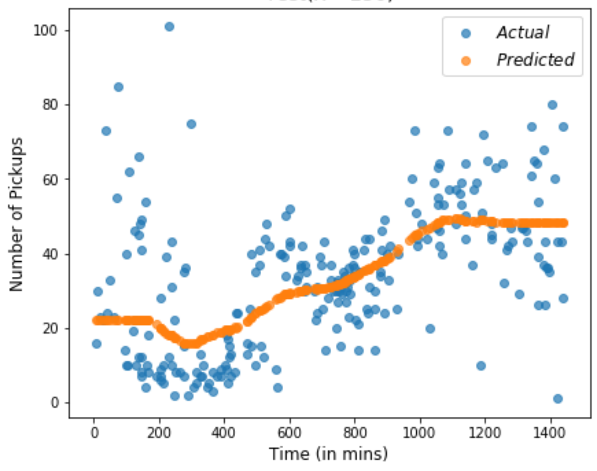
Train(*K* = 250)



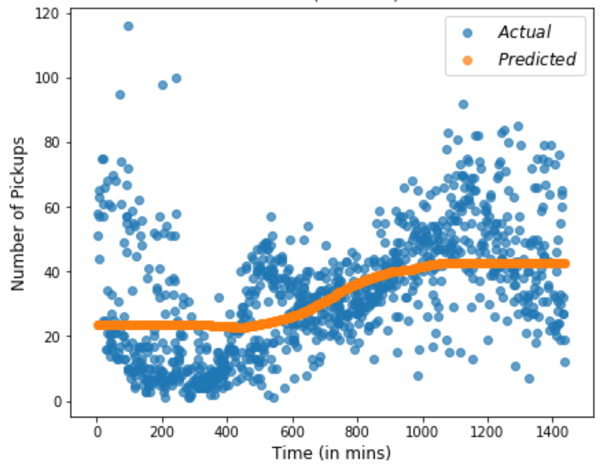
Test(*K* = 250)



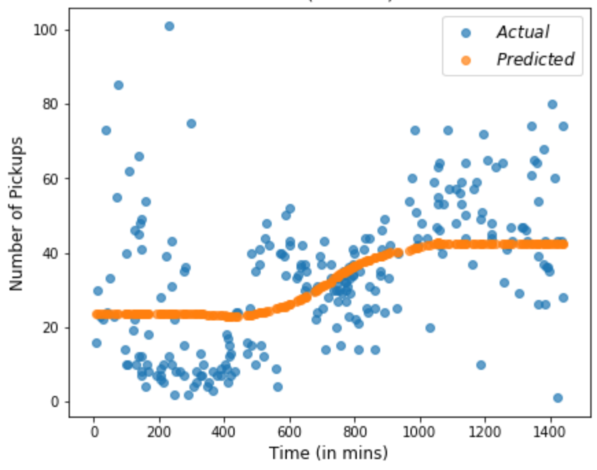
Train(*K* = 500)



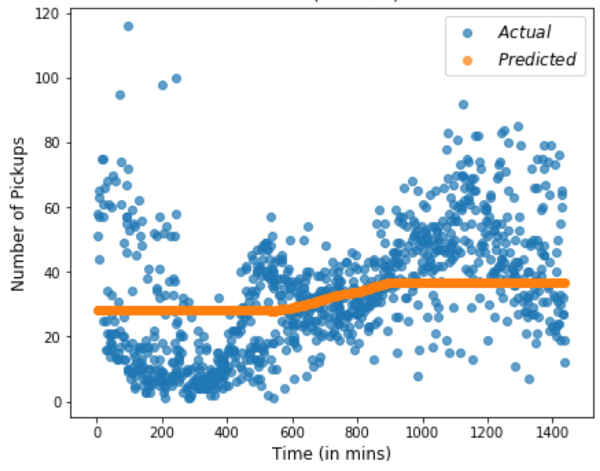
Test(*K* = 500)



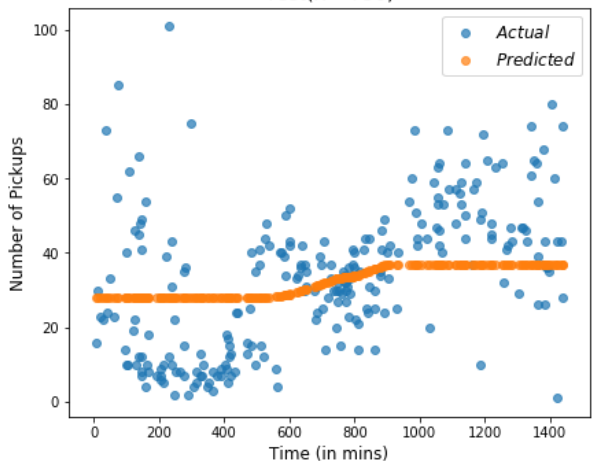
Train(*K* = 750)



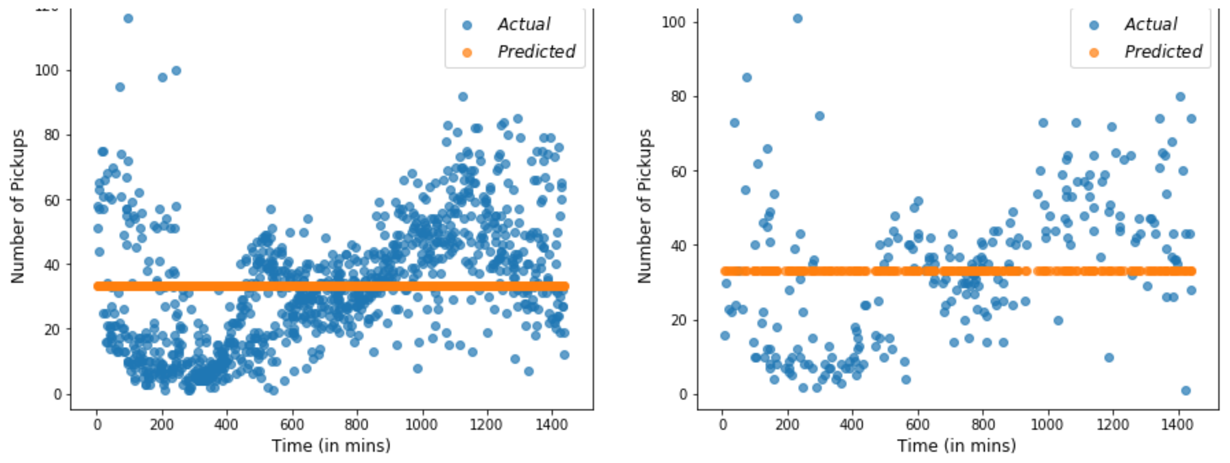
Test(*K* = 750)



Train(*K* = 1000)



Test(*K* = 1000)



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

In [11]:
```
# Get scores
score_test = []
score_train = []
columns = []
i=0
for k in KNNModels.keys():
    score_test.append(KNNModels[k].score(X_test, y_test))
    score_train.append(KNNModels[k].score(X_train, y_train))
    columns.append(str(k))

row_labels = ['k','Test', 'Train']
score = np.array((columns,score_test,score_train))

# Create dataframe
scores_tbl = pd.DataFrame(data=score, index=row_labels).T
scores_tbl['Test'] = scores_tbl.Test.astype(float)
scores_tbl['Train'] = scores_tbl.Train.astype(float)
scores_tbl
```

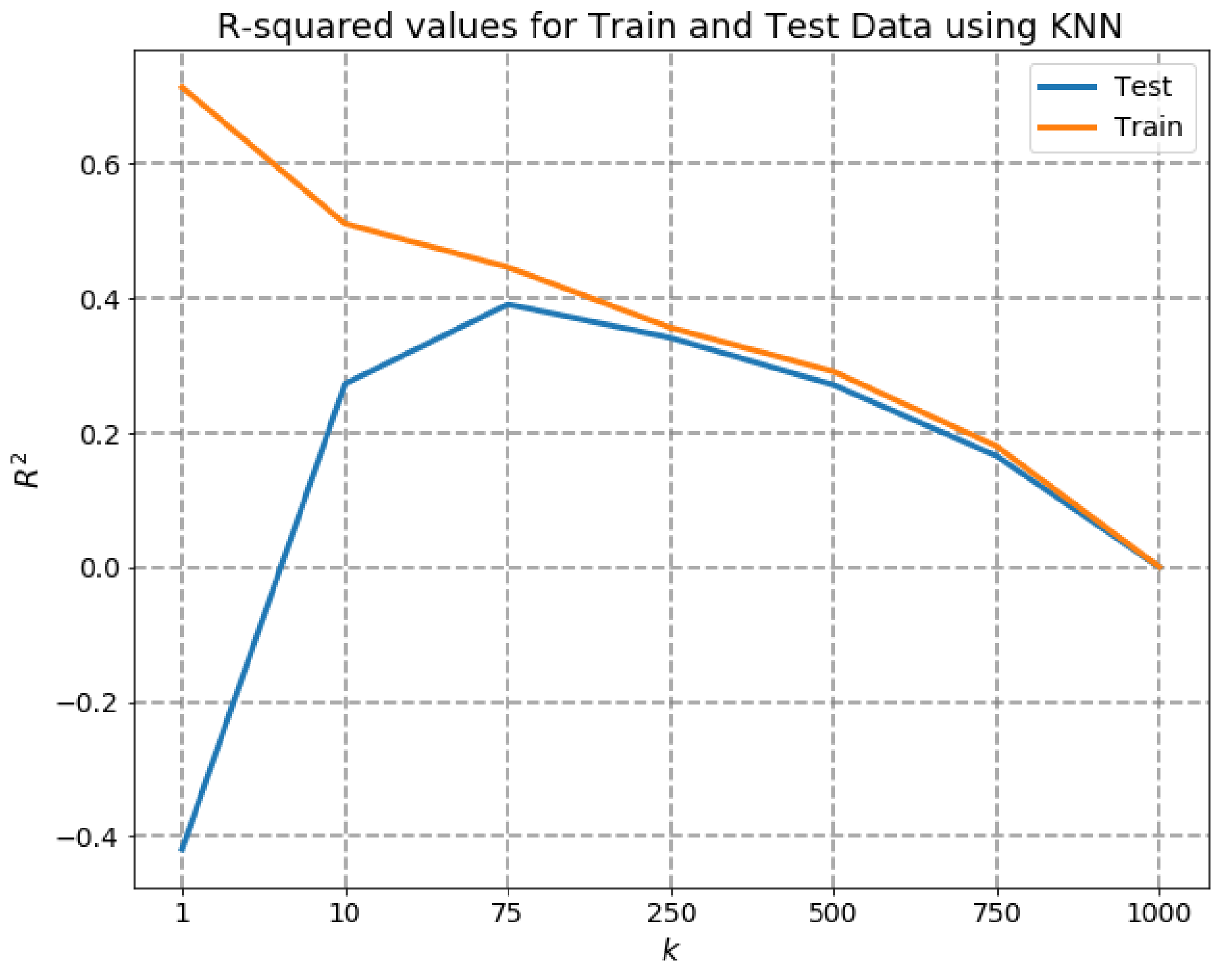
Out[11]:

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

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 [12]:
```python
# your code here
scores_tbl.round({'Test':2, 'Train':2})
fig_r2, ax_r2 = plt.subplots(1, 1, figsize = (10,8))
ax_r2.plot(scores_tbl.k, scores_tbl.Test, label='Test', lw=3)
ax_r2.plot(scores_tbl.k, scores_tbl.Train, label='Train', lw=3)
ax_r2.set_xlabel(r'$k$', fontsize = 16)
ax_r2.set_ylabel(r'$R^2$', fontsize = 16)
ax_r2.set_title('R-squared values for Train and Test Data using KNN', fontsize =
ax_r2.grid(True, lw = 1.75, ls = '--', color ='grey', alpha = 0.75)
ax_r2.tick_params(labelsize = 14)
ax_r2.legend(loc='best', fontsize=14)
```

Out[12]:
```
<matplotlib.legend.Legend at 0x3ceb45ba58>
```



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?

Answers:

1. Using $k = n$ is same as averaging all the values in the training set and using that single value as the prediction (aka. the average model).

2. R^2 score of 0 means that the prediction error of the regression model is same as that of the average model. However, this does not necessarily mean that the model is using $k = n$, where n is the number of observations in the training set.

3. A negative R^2 means that the prediction error is higher than the average model. We do get $-ve$ R^2 for the test set at $k = 1$ due to overfitting as we are trying fit our model exactly to the training set.

4. For $k <= 75$, both the plot exhibit opposite trends with the increasing value of k. For $k > 75$, however, both model follow the almost the same decreasing trend for the value of R^2.

5. As we go from $k = 1$ to $k = 1000$ the scores for both train and test set seem to converge to $R^2 = 0$. For test set, we see that we get much lower values of R^2 at $k <= 10$ scores due to the overfitting that happens at lower values of k. For the train set, the R^2 scores keep dropping until the model converges to the average model at $k = 1000$.

6. The best of k is at $k = 75$. Where $R^2 = 0.39$ for test set and $R^2 = 0.445$ for train set.

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?

Answers

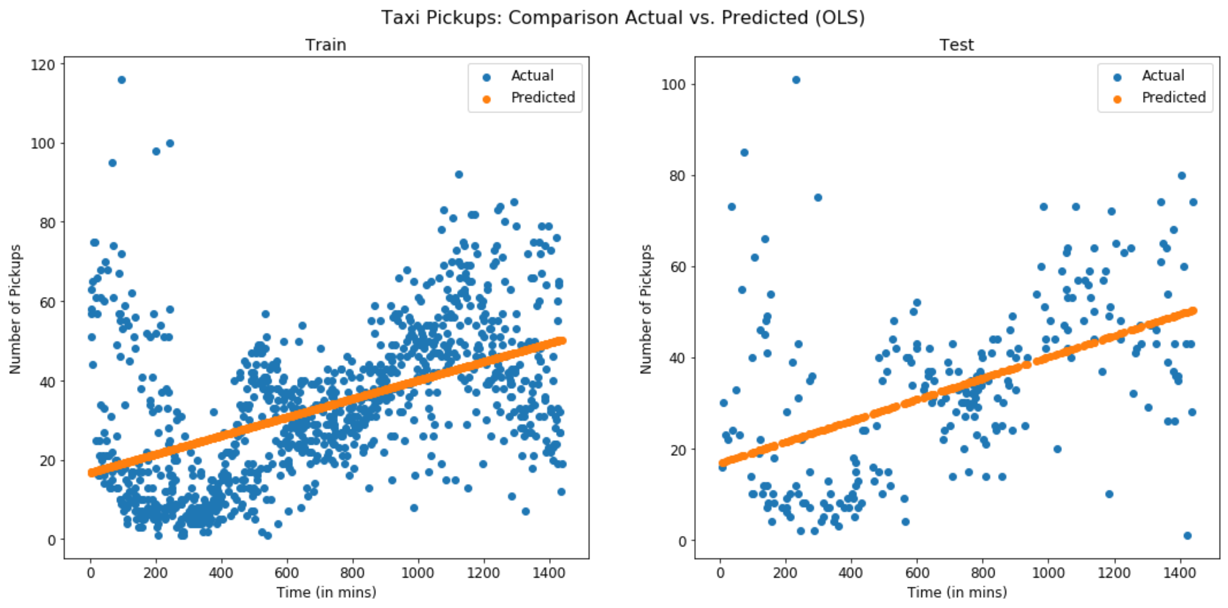
3.1 Again choose `TimeMin` as your predictor and `PickupCount` as your response variable. Create a `OLS` class instance ...

In [13]:
```
# your code here
OLS = sm.OLS(y_train, sm.add_constant(X_train))
OLSModel = OLS.fit()
```

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

In [14]:
```python
# your code here
predict_test = OLSModel.predict(sm.add_constant(X_test))
predict_train = OLSModel.predict(sm.add_constant(X_train))

fig_sm, ax_sm = plt.subplots(1, 2, figsize=(18,8))

# Subplots for training data
ax_sm[0].scatter(X_train, y_train, label='Actual')
ax_sm[0].scatter(X_train, predict_train, label='Predicted')
ax_sm[0].set_xlabel('Time (in mins)', fontsize=12)
ax_sm[0].set_ylabel('Number of Pickups', fontsize=12)
ax_sm[0].tick_params(labelsize=12)
ax_sm[0].set_title('Train', fontsize=14)
ax_sm[0].legend(loc='best', fontsize=12)

# Subplots for test data
ax_sm[1].scatter(X_test, y_test, label='Actual')
ax_sm[1].scatter(X_test, predict_test, label='Predicted')
ax_sm[1].set_xlabel('Time (in mins)', fontsize=12)
ax_sm[1].set_ylabel('Number of Pickups', fontsize=12)
ax_sm[1].tick_params(labelsize=12)
ax_sm[1].set_title('Test', fontsize=14)
ax_sm[1].legend(loc='best', fontsize=12)

fig_sm.suptitle('Taxi Pickups: Comparison Actual vs. Predicted (OLS)', y=0.95, fo
```

Out[14]:
```
Text(0.5,0.95,'Taxi Pickups: Comparison Actual vs. Predicted (OLS)')
```



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

In [15]:
```
# your code here
# Get scores
X_train_ca = sm.add_constant(X_train)
X_test_ca = sm.add_constant(X_test)

score_ols_train = r2_score(y_train, OLSModel.predict(X_train_ca))
score_ols_test = r2_score(y_test, OLSModel.predict(X_test_ca))

row = ['OLS $R^2$ Score - Test', 'OLS $R^2$ Score - Train']
score = np.array([score_ols_test, score_ols_train])

# Create dataframe
scores_ols = pd.DataFrame(data=score, index=row, columns = ['Values'])
scores_ols
```

Out[15]:

	Values
OLS $R^2$ Score - Test	0.240662
OLS $R^2$ Score - Train	0.243026

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

In [16]:
```
## show summary
# your code here
import warnings
warnings.filterwarnings('ignore')
print(OLSModel.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:            PickupCount   R-squared:                       0.243
Model:                            OLS   Adj. R-squared:                  0.242
Method:                 Least Squares   F-statistic:                     320.4
Date:                Wed, 26 Sep 2018   Prob (F-statistic):           2.34e-62
Time:                        22:17:23   Log-Likelihood:                -4232.9
No. Observations:                1000   AIC:                             8470.
Df Residuals:                     998   BIC:                             8480.
Df Model:                           1
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const         16.7506      1.058     15.838      0.000      14.675      18.826
TimeMin        0.0233      0.001     17.900      0.000       0.021       0.026
==============================================================================
Omnibus:                      203.688   Durbin-Watson:                   1.910
Prob(Omnibus):                  0.000   Jarque-Bera (JB):              462.910
Skew:                           1.111   Prob(JB):                    3.02e-101
Kurtosis:                       5.485   Cond. No.                     1.63e+03
==============================================================================

Warnings:
[1] Standard Errors assume that the covariance matrix of the errors is correctl
y specified.
[2] The condition number is large, 1.63e+03. This might indicate that there are
strong multicollinearity or other numerical problems.
```

In [17]:
```
# your code here
slope = OLSModel.params['TimeMin']
intercept = OLSModel.params['const']
pd.DataFrame(data=np.array([slope,intercept]), index=['Slope', 'Intercept'], colu
```

Out[17]:

	Values
Slope	0.023335
Intercept	16.750601

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

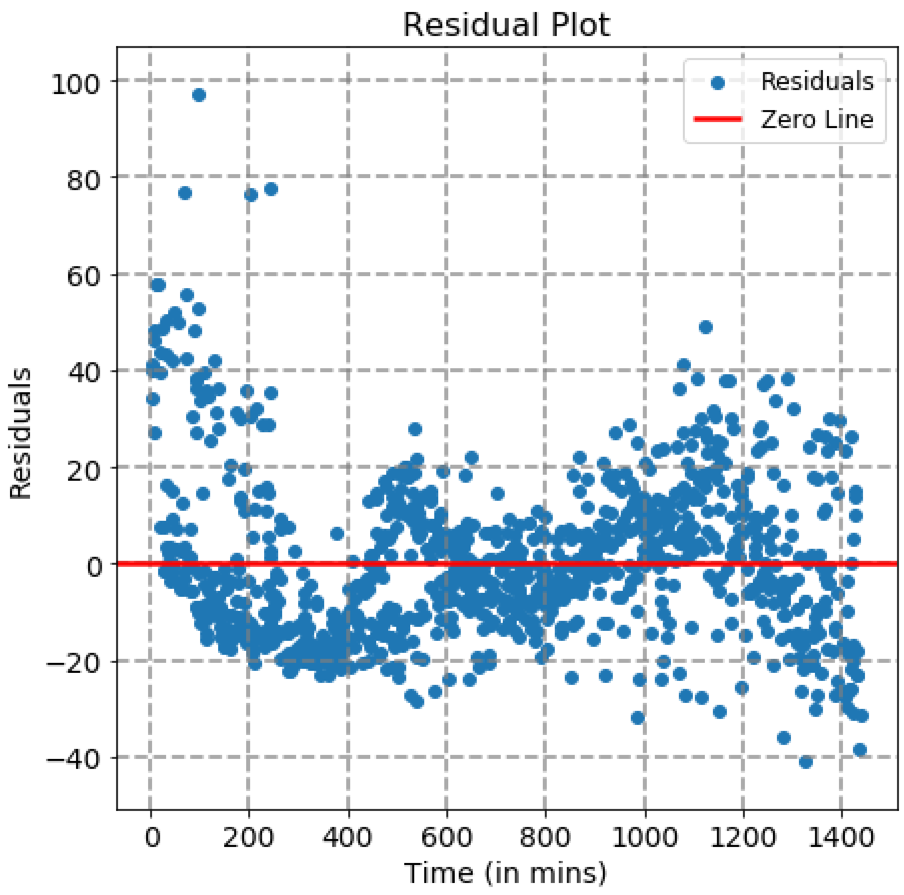
In [18]:
```
# your code here
CI = OLSModel.conf_int(alpha=0.05)
CI
intervals = CI.rename(index=str, columns={0:"2.5%",1:"97.5%"})
display(intervals)
```

	2.5%	97.5%
const	14.675141	18.826062
TimeMin	0.020777	0.025893

3.6 Create a scatter plot of the residuals

In [19]:
```
# your code here
residuals_train = OLSModel.resid

fig_rs, ax_rs = plt.subplots(1, 1, figsize = (7,7))
ax_rs.scatter(X_train, residuals_train, label='Residuals')
ax_rs.hlines(y=0, xmin=np.min(train_data.TimeMin)-75, xmax=np.max(train_data.Time
ax_rs.set_xlabel('Time (in mins)', fontsize=14)
ax_rs.set_ylabel('Residuals', fontsize=14)
ax_rs.set_xlim(np.min(train_data.TimeMin)-75, np.max(train_data.TimeMin)+75)
ax_rs.set_ylim(np.min(residuals_train)-10, np.max(residuals_train)+10)
ax_rs.set_title('Residual Plot', fontsize=16)
ax_rs.grid(True, lw=1.75, ls='--', color='grey', alpha=0.75)
ax_rs.tick_params(labelsize=14)
ax_rs.legend(loc='best', fontsize=12)
```

Out[19]:
```
<matplotlib.legend.Legend at 0x3ced616ef0>
```



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?

Answers:

1. The best R^2 value obtained by k-NN regression is at $k = 75$: 0.39 for test and 0.44 for train. These scores are higher compared to the R^2 values obtained by linear regression (0.243 for test and 0.24 for train).

2. The $+ve$ sign of the slope indicates a positive correlation between our X and y variables in the given data i.e. as TimeMin increases, PickupCount increases.

3. The 95% confidence intervals are fairly narrow compared to the mean values of the model parameters. Hence, we can consider that the estimates of the model parameters to be reliable.

4. 99% confidence intervals will be wider compared to the 95%. The reason being that 99% confidence interval includes 99% of the slope/intercept values that we get by bootstrapping, so we can say that there is 99% chance that the slope or the intercept would be within that interval.

5. We can clearly see that the data is not linear based on the residual plot. However, the linear model does give us reasonably good prediction for $200 <= TimeMin <= 1200$, we can see in the above plot that the residuals are closer to zero in this range. Hence, the assumption of linearity is still valid for this range but not for the entire data set.

6. Considering the scatter plots, we should model for $200 <= TimeMin <= 1200$ with the linear model. This tells us our data is not linear.

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?

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 [20]:
```
# read the data
# your code here
train = pd.read_csv('data/outliers_train.csv')
test = pd.read_csv('data/outliers_test.csv')
train.head()
```

Out[20]:

	X	Y
0	-0.773019	-219.103753
1	-0.394034	-334.859357
2	0.630360	-16.232549
3	-0.350418	-179.034618
4	-1.491328	-109.710316

In [21]:
```
# your code here
train.shape, test.shape
```

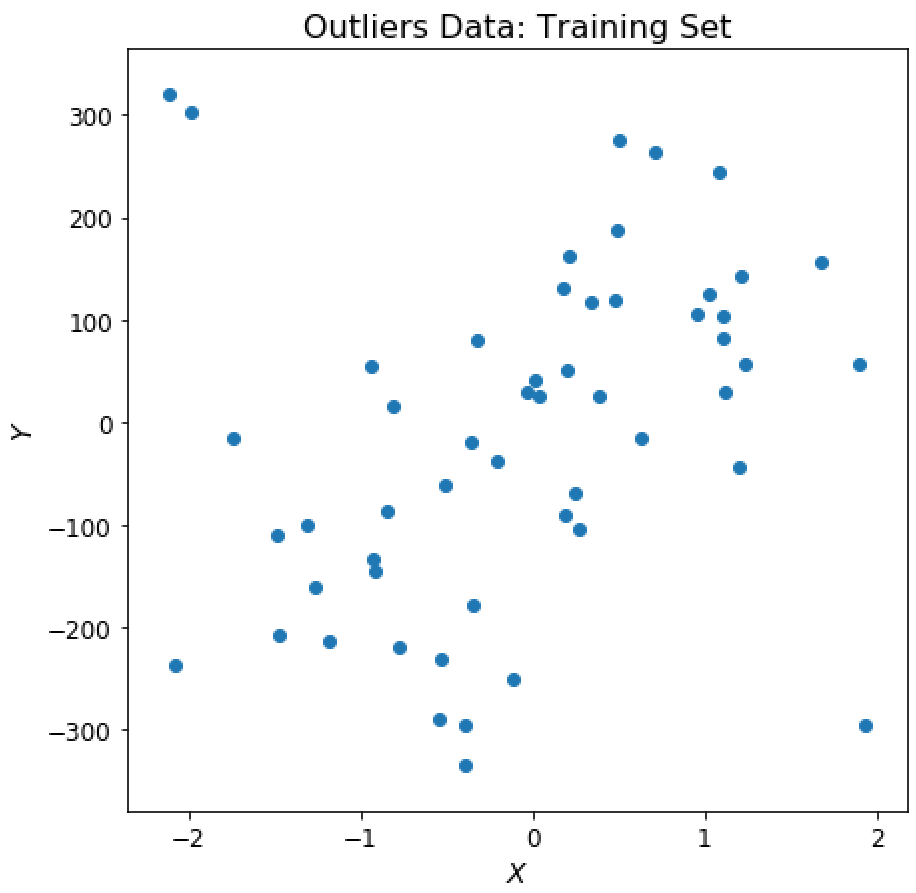
Out[21]:
```
((53, 2), (50, 2))
```

In [22]:
```
# your code here
test.head()
```

Out[22]:

	X	Y
0	-0.573524	-91.080764
1	-0.793911	-19.982576
2	0.788391	118.593685
3	0.489036	64.973804
4	1.530648	178.281580

In [23]:
```python
# scatter plot
# your code here
fig_ol, ax_ol = plt.subplots(1, 1, figsize=(7,7))
ax_ol.scatter(train.X, train.Y)
ax_ol.set_xlabel(r'$X$', fontsize = 14)
ax_ol.set_ylabel(r'$Y$', fontsize = 14)
ax_ol.set_xlim(np.min(train.X)-0.25, np.max(train.X)+0.25)
ax_ol.set_ylim(np.min(train.Y)-45, np.max(train.Y)+45)
ax_ol.set_title('Outliers Data: Training Set', fontsize=16)
ax_ol.tick_params(labelsize=12)
```



Answer:

Based on the above scatter plot, we can see that there are certainly some outliers in the data. Two of them we can see at upper left hand corner and one at lower right corner. We can also see that the points are sparse.

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

In [24]:
```
# your code here
OutlierOLS = sm.OLS(train.Y, sm.add_constant(train.X))
OutlierOLSModel = OutlierOLS.fit()
print(OutlierOLSModel.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                      Y   R-squared:                       0.084
Model:                            OLS   Adj. R-squared:                  0.066
Method:                 Least Squares   F-statistic:                     4.689
Date:                Wed, 26 Sep 2018   Prob (F-statistic):             0.0351
Time:                        22:17:39   Log-Likelihood:                -343.59
No. Observations:                  53   AIC:                             691.2
Df Residuals:                      51   BIC:                             695.1
Df Model:                           1
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const         -9.5063     22.192     -0.428      0.670     -54.059      35.046
X             47.3554     21.869      2.165      0.035       3.452      91.259
==============================================================================
Omnibus:                        2.102   Durbin-Watson:                   1.758
Prob(Omnibus):                  0.350   Jarque-Bera (JB):                1.251
Skew:                           0.215   Prob(JB):                        0.535
Kurtosis:                       3.617   Cond. No.                         1.06
==============================================================================

Warnings:
[1] Standard Errors assume that the covariance matrix of the errors is correctl
y specified.
```

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?

Answer: *For any given n observations, if we are to find best Linear Regression model on all possible subsets with k points removed, the number of ways to do this (n choose k) is $n!/k!(n-k)!$. So, for our training set ($n = 53$ and $k = 3$),*

Total Number of Linear Regression Models $= 53!/(3!50!) = 23426$

4.4 CS109 hack ...

In [25]:
```
# your code here
def find_outliers_simple(dataset_x, dataset_y):
    # your code here
    OLSOutlier = sm.OLS(dataset_y, sm.add_constant(dataset_x))
    OutlierOLSModel = OLSOutlier.fit()
    rs2 = OutlierOLSModel.resid**2 # taking care of negative residuals
    ol_ind = list(rs2.argsort()[-3:,])
    return(ol_ind)

outlier_indices = find_outliers_simple(train.X, train.Y)
outlier_indices
```

Out[25]:
```
[52, 51, 50]
```

In [26]:
```
# get outliers
# your code here
simple_outliers_y = train.Y[outlier_indices]
```

In [27]:
```
simple_outliers_y
```

Out[27]:
```
52   -297.0
51    303.0
50    320.0
Name: Y, dtype: float64
```

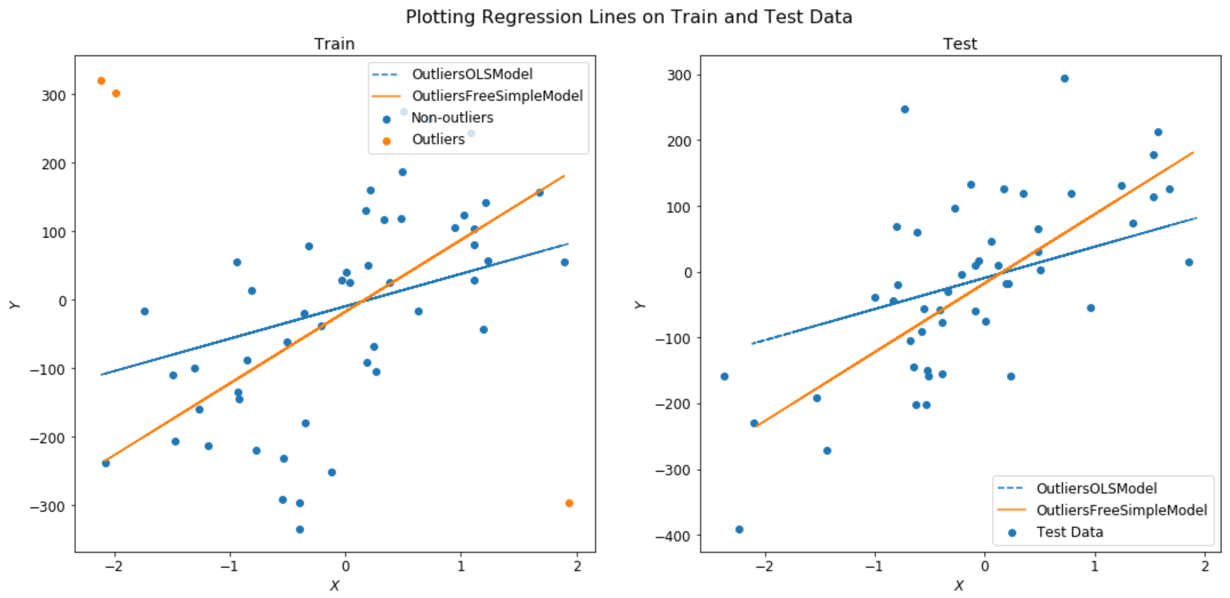
In [28]:
```
# calculate outlier model
# your code here
olfree_train = train.drop(outlier_indices)

OLS_free = sm.OLS(olfree_train.Y, sm.add_constant(olfree_train.X))
OutlierFreeSimpleModel = OLS_free.fit()
print(OutlierFreeSimpleModel.summary())
```

```
                            OLS Regression Results                            
==============================================================================
Dep. Variable:                      Y   R-squared:                       0.404
Model:                            OLS   Adj. R-squared:                  0.391
Method:                 Least Squares   F-statistic:                     32.50
Date:                Wed, 26 Sep 2018   Prob (F-statistic):           7.16e-07
Time:                        22:17:45   Log-Likelihood:                -309.21
No. Observations:                  50   AIC:                             622.4
Df Residuals:                      48   BIC:                             626.2
Df Model:                           1                                         
Covariance Type:            nonrobust                                         
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const        -17.4796     16.944     -1.032      0.307     -51.547      16.588
X            104.8467     18.392      5.701      0.000      67.867     141.827
==============================================================================
Omnibus:                        0.600   Durbin-Watson:                   1.683
Prob(Omnibus):                  0.741   Jarque-Bera (JB):                0.673
Skew:                          -0.238   Prob(JB):                        0.714
Kurtosis:                       2.689   Cond. No.                         1.09
==============================================================================

Warnings:
[1] Standard Errors assume that the covariance matrix of the errors is correctl
y specified.
```

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

In [29]:
```
# plot
# your code here

Beta0_ol = OutlierOLSModel.params['const']
Beta1_ol = OutlierOLSModel.params['X']
fit_ol = Beta0_ol + Beta1_ol*train.X

Beta0_olfree = OutlierFreeSimpleModel.params['const']
Beta1_olfree = OutlierFreeSimpleModel.params['X']
fit_olfree = Beta0_olfree + Beta1_olfree*olfree_train.X

fig_ol1, ax_ol1 = plt.subplots(1, 2, figsize=(18,8))

# Subplots for training data
ax_ol1[0].scatter(olfree_train.X, olfree_train.Y, label='Non-outliers')
ax_ol1[0].scatter(train.X[outlier_indices], train.Y[outlier_indices], label='Outl
ax_ol1[0].plot(train.X, fit_ol, ls = '--', label = 'OutliersOLSModel')
ax_ol1[0].plot(olfree_train.X, fit_olfree, ls = '-', label = 'OutliersFreeSimpleM
ax_ol1[0].set_xlabel(r'$X$', fontsize=12)
ax_ol1[0].set_ylabel(r'$Y$', fontsize=12)
ax_ol1[0].tick_params(labelsize=12)
ax_ol1[0].set_title('Train', fontsize=14)
ax_ol1[0].legend(loc='upper right', fontsize=12)

# Subplots for test data
ax_ol1[1].scatter(test.X, test.Y, label='Test Data')
ax_ol1[1].plot(train.X, fit_ol, ls = '--', label = 'OutliersOLSModel')
ax_ol1[1].plot(olfree_train.X, fit_olfree, ls = '-', label = 'OutliersFreeSimpleM
ax_ol1[1].set_xlabel(r'$X$', fontsize=12)
ax_ol1[1].set_ylabel(r'$Y$', fontsize=12)
ax_ol1[1].tick_params(labelsize=12)
ax_ol1[1].set_title('Test', fontsize=14)
ax_ol1[1].legend(loc='best', fontsize=12)

fig_ol1.suptitle('Plotting Regression Lines on Train and Test Data', y=0.95, font
```

Out[29]:
```
Text(0.5,0.95,'Plotting Regression Lines on Train and Test Data')
```



Answer: *Looking at the Test plot we see that the 'OutliersFreeOLSModel' fits the test set better, as we can apporximately see that the distances of the data points from its regression line are smaller compared to the 'OutliersOLSModel'.*

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 [30]:
```
# your code here
Xtest_ca = sm.add_constant(test.X)

score_ol = r2_score(test.Y, OutlierOLSModel.predict(Xtest_ca))
score_olfree = r2_score(test.Y, OutlierFreeSimpleModel.predict(Xtest_ca))

print(r'The R^2 score for OutliersFreeOLSModel is %s'%score_olfree)
print(r'The R^2 score for OutliersOLSModel is %s'%score_ol)
```

```
The R^2 score for OutliersFreeOLSModel is 0.4529566870167582
The R^2 score for OutliersOLSModel is 0.34085656043405654
```

Answer: *We can see that the OutliersFreeOLSModel does better compared to the OutliersOLSModel*

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.

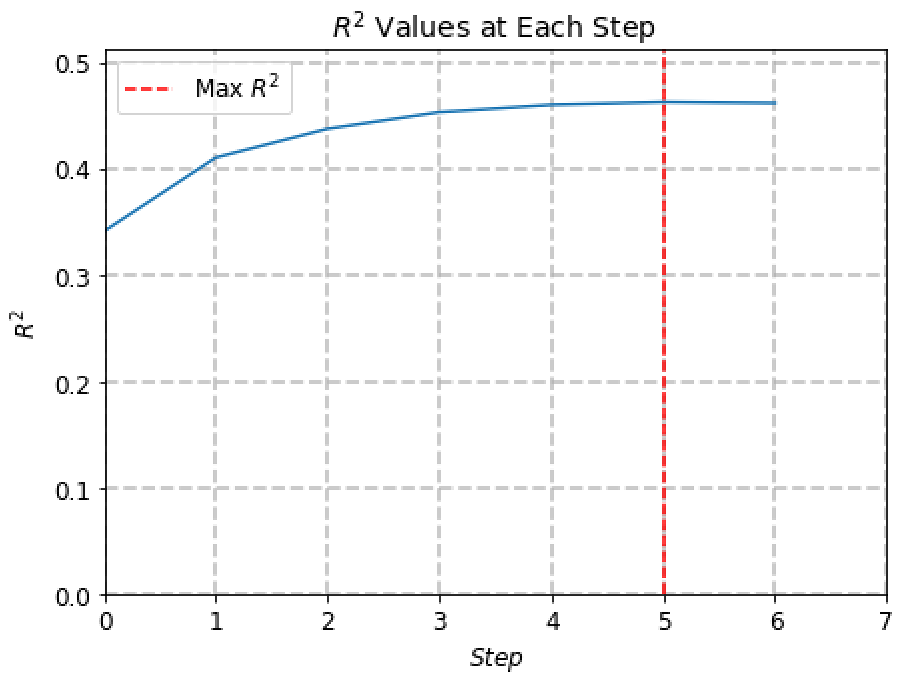
In [31]:
```
# your code here
r2_gen = []
def find_outliers_general(dataset_x, dataset_y, test_x = test.X, test_y = test.Y):
    # your code here
    prev_r2 = 0
    curr_r2 = 0
    x = sm.add_constant(dataset_x)
    y = dataset_y
    x_test = sm.add_constant(test_x)
    y_test = test_y
    outlier_indices = []
    i = 0
    OLSOutlier = sm.OLS(y, x)
    OutlierOLSModel = OLSOutlier.fit()

    while (True):
        i += 1
        rs2 = OutlierOLSModel.resid**2 # taking care of negative residuals
        ol_ind = rs2.idxmax()
        prev_r2 = r2_score(y_test, OutlierOLSModel.predict(x_test))
        # Outlier free r2
        olfree_x = x.drop(index = ol_ind, axis = 0)
        olfree_y = y.drop(index = ol_ind, axis = 0)
        # Model without outliers
        OLSOutlierFree = sm.OLS(olfree_y, olfree_x)
        OutlierFreeOLSModel = OLSOutlierFree.fit()
        curr_r2 = r2_score(y_test, OutlierFreeOLSModel.predict(x_test))

        if i>1:
            r2_gen.append(curr_r2)
        else:
            r2_gen.append(prev_r2)
            r2_gen.append(curr_r2)

        if curr_r2 > prev_r2:
            outlier_indices.append(ol_ind)
        else:
            break

        OutlierOLSModel = OutlierFreeOLSModel
        x = olfree_x
        y = olfree_y

    return(outlier_indices)
```

In [32]:
```
# Outliers using general function
outlier_ind = find_outliers_general(train.X, train.Y)

# Building model
olfreegen_train = train.drop(outlier_ind)
OLS_free_gen = sm.OLS(olfreegen_train.Y, sm.add_constant(olfreegen_train.X))
OutlierFreeGeneralModel = OLS_free_gen.fit()
print(OutlierFreeGeneralModel.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                      Y   R-squared:                       0.434
Model:                            OLS   Adj. R-squared:                  0.422
Method:                 Least Squares   F-statistic:                     35.33
Date:                Wed, 26 Sep 2018   Prob (F-statistic):           3.51e-07
Time:                        22:17:50   Log-Likelihood:                -292.41
No. Observations:                  48   AIC:                             588.8
Df Residuals:                      46   BIC:                             592.6
Df Model:                           1
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const         -6.8068     15.779     -0.431      0.668     -38.567      24.954
X            100.1170     16.843      5.944      0.000      66.214     134.020
==============================================================================
Omnibus:                        0.125   Durbin-Watson:                   1.916
Prob(Omnibus):                  0.939   Jarque-Bera (JB):                0.330
Skew:                          -0.041   Prob(JB):                        0.848
Kurtosis:                       2.602   Cond. No.                         1.07
==============================================================================

Warnings:
[1] Standard Errors assume that the covariance matrix of the errors is correctl
y specified.
```

In [33]:
```
fig_r2, ax_r2 = plt.subplots(1, 1, figsize=(7,5))
ax_r2.plot(r2_gen)
ax_r2.vlines(x=len(r2_gen[:-2]), ymin = 0, ymax = np.max(r2_gen)+0.05, linestyles
ax_r2.set_xlabel(r'$Step$', fontsize = 12)
ax_r2.set_ylabel(r'$R^2$', fontsize = 12)
ax_r2.set_xlim(0, len(r2_gen))
ax_r2.set_ylim(0, np.max(r2_gen)+0.05)
ax_r2.grid(alpha = 0.75, lw = 1.75, ls = '--')
ax_r2.tick_params(labelsize = 12)
ax_r2.set_title('$R^2$ Values at Each Step', fontsize = 14)
ax_r2.legend(loc = 'best', fontsize = 12)
```

Out[33]:
```
<matplotlib.legend.Legend at 0x3cede03438>
```



Description of the General Algorithm: *Since we dont have a stopping point already given to us, the goal is to find the optimum number of outliers to get the highest $R^2$ score on the test set. At each step, we try to identify one point with the highest squared residual value and compare the $R^2$ scores on the test set before and after taking out that point. We then fit a model on the new data each time to measure the improvement in terms of the most recent model and not just the original OutlierOLSModel.*

Also note the plot of $R^2$ values at each step. We stop at step 5 since after that we start getting same $R^2$ scores.

4.8 Run your algorithm in 4.7 on the training set data

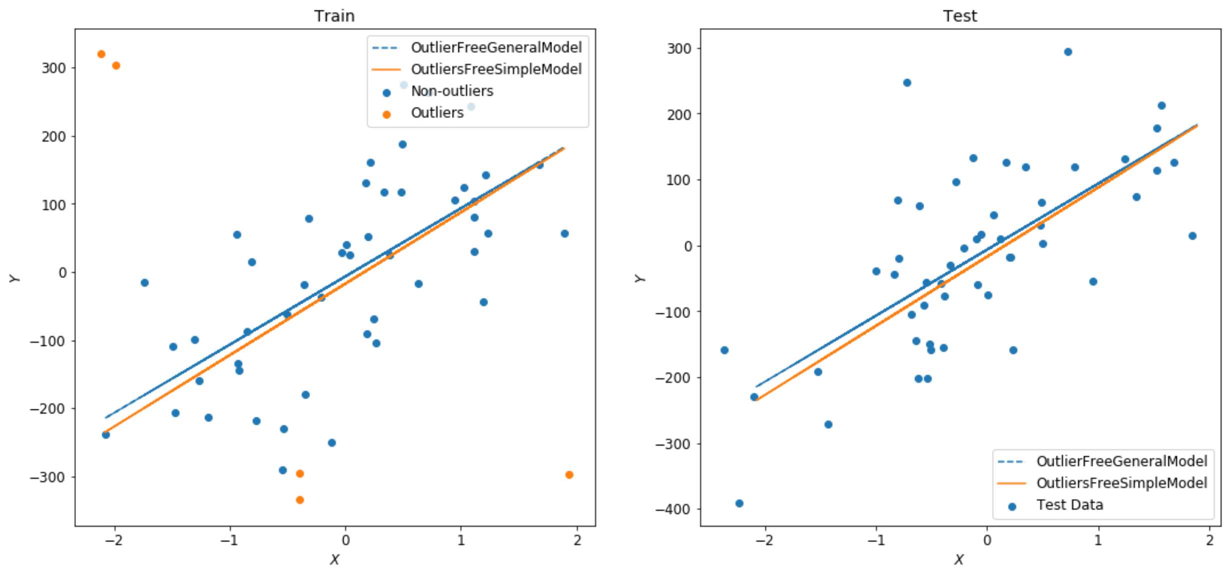
In [34]:
```
# your code here
outlier_ind = find_outliers_general(train.X, train.Y)
train.iloc[outlier_ind,:]
```

Out[34]:

	X	Y
50	-2.110000	320.000000
51	-1.991000	303.000000
52	1.931000	-297.000000
1	-0.394034	-334.859357
14	-0.391668	-295.878637

In [35]:
```
# your code here
Beta0_olfree = OutlierFreeSimpleModel.params['const']
Beta1_olfree = OutlierFreeSimpleModel.params['X']
fit_olfree = Beta0_olfree + Beta1_olfree*olfree_train.X

Beta0_olfree_gen = OutlierFreeGeneralModel.params['const']
Beta1_olfree_gen = OutlierFreeGeneralModel.params['X']
fit_olfree_gen = Beta0_olfree_gen + Beta1_olfree_gen*olfreegen_train.X

fig_ol2, ax_ol2 = plt.subplots(1, 2, figsize=(18,8))

# Subplots for training data
ax_ol2[0].scatter(olfreegen_train.X, olfreegen_train.Y, label='Non-outliers')
ax_ol2[0].scatter(train.X[outlier_ind], train.Y[outlier_ind], label='Outliers')
ax_ol2[0].plot(olfreegen_train.X, fit_olfree_gen, ls = '--', label = 'OutlierFree
ax_ol2[0].plot(olfree_train.X, fit_olfree, ls = '-', label = 'OutliersFreeSimpleM
ax_ol2[0].set_xlabel(r'$X$', fontsize=12)
ax_ol2[0].set_ylabel(r'$Y$', fontsize=12)
ax_ol2[0].tick_params(labelsize=12)
ax_ol2[0].set_title('Train', fontsize=14)
ax_ol2[0].legend(loc='upper right', fontsize=12)

# Subplots for test data
ax_ol2[1].scatter(test.X, test.Y, label='Test Data')
ax_ol2[1].plot(olfreegen_train.X, fit_olfree_gen, ls = '--', label = 'OutlierFree
ax_ol2[1].plot(olfree_train.X, fit_olfree, ls = '-', label = 'OutliersFreeSimpleM
ax_ol2[1].set_xlabel(r'$X$', fontsize=12)
ax_ol2[1].set_ylabel(r'$Y$', fontsize=12)
ax_ol2[1].tick_params(labelsize=12)
ax_ol2[1].set_title('Test', fontsize=14)
ax_ol2[1].legend(loc='best', fontsize=12)

fig_ol1.suptitle('Plotting Regression Lines on Train and Test Data', y=0.95, font
```

Out[35]:
```
Text(0.5,0.95,'Plotting Regression Lines on Train and Test Data')
```



In [36]:
```
# your code here
Xtest_gen = sm.add_constant(test.X)

score_olfree = r2_score(test.Y, OutlierFreeSimpleModel.predict(Xtest_gen))
score_olfree_gen = r2_score(test.Y, OutlierFreeGeneralModel.predict(Xtest_gen))

print(r'The R^2 score for OutlierFreeSimpleModel is: %s'%score_olfree)
print(r'The R^2 score for OutlierFreeGeneralModel is: %s'%score_olfree_gen)
```

```
The R^2 score for OutlierFreeSimpleModel is: 0.4529566870167582
The R^2 score for OutlierFreeGeneralModel is: 0.46253823357744095
```

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?

Answers:

1. The general algorithm identifies 5 outliers as shown in the 'Train' plot above, including the 3 that we identified in 4.4. The specific outliers are listed in the output of the first snippet of code in 4.8.
2. We see that the top 3 outliers are the same as those identified in 4.4 and then the algorithm continues to identify more to improve the R^2 score.
3. The general method gives us a better R^2 (0.434) as compared to the OutlierFreeSimpleModel (0.404). The above plots and its R^2 scores on the test data, both suggest that this model performs better than the Simple model.

In [ ]:
```
```