S-109A Introduction to Data Science:

Homework 2: Linear and k-NN Regression

Harvard University
Summer 2018
Instructors: Pavlos Protopapas, Kevin Rader

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.

Names of people you have worked with goes here:

Nikita Roy

In [93]:
```
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
```

Main Theme: 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 [10 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 well-chosen labels on the x and y axes. The time of the day should be 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?

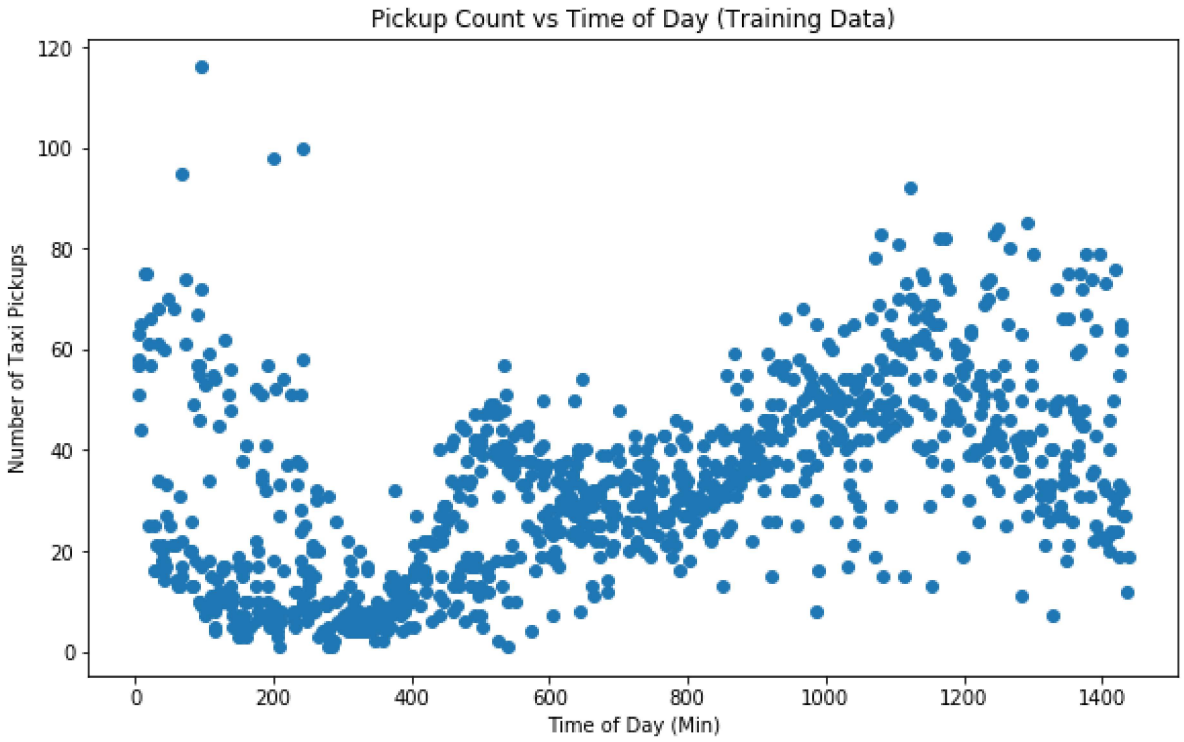
Answers

1.1

In [94]:
```
# Read in dataset_1.csv
dataset_1 = pd.read_csv("dataset_1.csv")


# Split dataset_1 into train_data, test_data
train_data, test_data = train_test_split(dataset_1, random_state = 42, test_si
ze = 0.2)

# Make variables for later use in the HW
train_pickup = train_data["PickupCount"].values
train_timemin = train_data["TimeMin"].values
test_pickup = test_data["PickupCount"].values
test_timemin = test_data["TimeMin"].values
```

1.2

In [95]:
```
# Build scatter plot of Training Data
fig, ax = plt.subplots(1,1, figsize=(10,6))

ax.scatter(x=train_data["TimeMin"].values, y=train_pickup)

# Create labels
ax.set_title("Pickup Count vs Time of Day (Training Data)")
ax.set_xlabel("Time of Day (Min)")
ax.set_ylabel("Number of Taxi Pickups")

# Display plot
plt.show()
```



1.3

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

Yes the plot generally makes sense because intuitively, we would expect the highest pickup count to be in the evening, which is the case. The seemingly rando outliers may be due to closing times of restaurants, bars, and clubs in the city, which would involve sudden spikes of passengers who would not be capable of driving.

Question 2 [20 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 predictor variable (aka, feature) 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 predicted `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. In each subplot, use two different colors and/or markers to distinguish k-NN 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. (use `suptitle`)

2.3. Report the R^2 score for the fitted models on both the training and test sets for each k.

Hints:

1. Reporting the R^2 values in tabular form is encouraged.
2. You should order your reported R^2 values by k.

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

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?

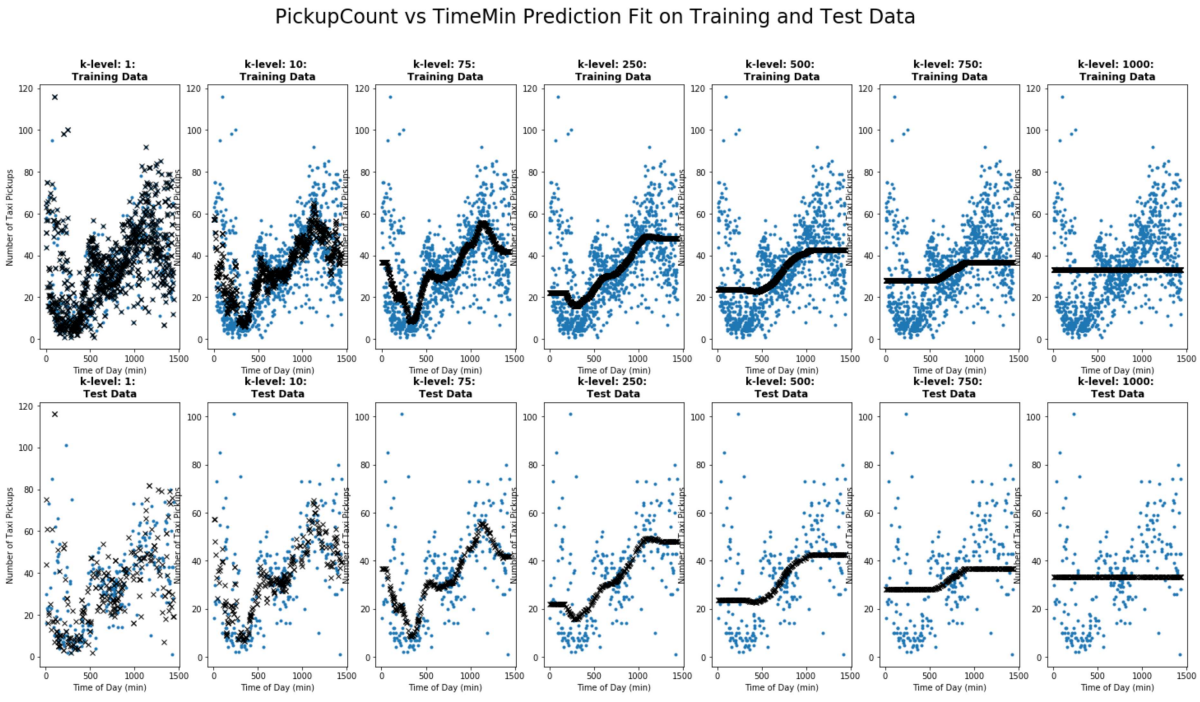
Answers

2.1

In [96]:
```
# Reshape train_data values so that they can be fit
train_timemin_reshape = train_data["TimeMin"].values.reshape(-1,1)

# Create a dictionary of KNeighborRegressor objects
KNNModels = {}
for k in [1, 10, 75, 250, 500, 750, 1000]:
    knnreg = KNeighborsRegressor(n_neighbors=k)
    knnreg.fit(train_timemin_reshape, train_pickup)
    KNNModels[k] = knnreg
```

2.2

In [97]:
```
# Declare variables
test_timemin_reshape = test_data["TimeMin"].values.reshape(-1,1)

# Initialize subplots
fig, ax = plt.subplots(nrows=2, ncols=7, figsize = (25,13))

# Creates subplots for training and test data at each k-level
for index, key in enumerate(KNNModels):

    # Train, predicted values and scatter plot
    ax[0][index].plot(train_timemin_reshape, train_pickup, ".")
    ax[0][index].plot(train_timemin_reshape, KNNModels[key].predict(train_time
min_reshape), "kx")

    # Test, predicted values and scatter plot
    ax[1][index].plot(test_timemin_reshape, test_pickup, ".")
    ax[1][index].plot(test_timemin_reshape, KNNModels[key].predict(test_timemi
n_reshape), "kx")

    # Set labels
    ax[0][index].set_title("k-level: {0}: \nTraining Data".format(key), fontwe
ight = 'bold')
    ax[1][index].set_title("k-level: {0}: \nTest Data".format(key), fontweight
 = 'bold')
    ax[0][index].set_xlabel("Time of Day (min)")
    ax[0][index].set_ylabel("Number of Taxi Pickups")
    ax[1][index].set_xlabel("Time of Day (min)")
    ax[1][index].set_ylabel("Number of Taxi Pickups")

fig.suptitle("PickupCount vs TimeMin Prediction Fit on Training and Test Data"
, fontsize=24)
plt.show()
```



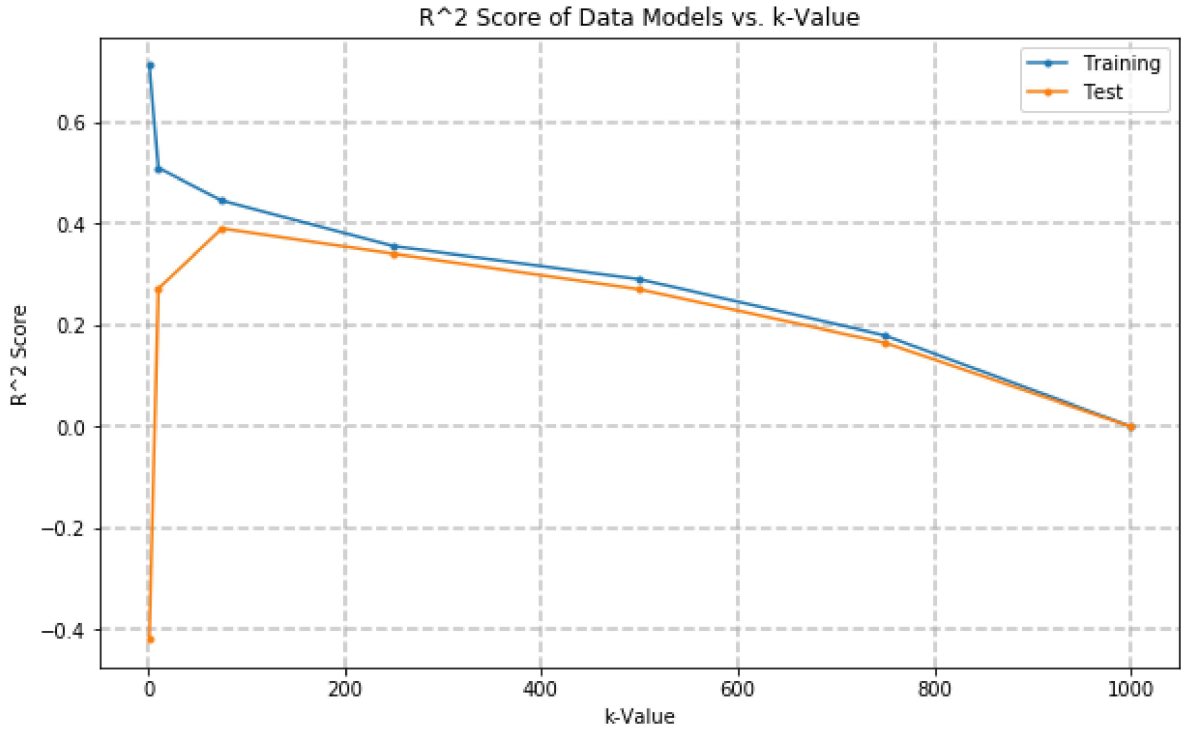
2.3

In [98]:
```
## Initialize DataFrame
df = pd.DataFrame(columns=("k-value", "Train R^2", "Test R^2"))

# Calcuates R^2 values for test and training sets
for index, key in enumerate(KNNModels):

    # Train
    r2_train = KNNModels[key].score(train_timemin_reshape, train_pickup)

    # Test
    r2_test = KNNModels[key].score(test_timemin_reshape, test_pickup)
    df.loc[index] = [key, r2_train, r2_test]

# Print dataframe
print(df)
```

```
   k-value  Train R^2  Test R^2
0      1.0   0.712336 -0.418932
1     10.0   0.509825  0.272068
2     75.0   0.445392  0.390310
3    250.0   0.355314  0.340341
4    500.0   0.290327  0.270321
5    750.0   0.179434  0.164909
6   1000.0   0.000000 -0.000384
```

2.4

In [99]:
```
# Create plot
fig, ax = plt.subplots(1,1, figsize=(10,6))

ax.plot(df["k-value"].values, df["Train R^2"].values, marker=".", label="Train
ing")
ax.plot(df["k-value"].values, df["Test R^2"].values, marker=".", label="Test")

# Set style, labels
ax.legend()
ax.set_title("R^2 Score of Data Models vs. k-Value")
ax.set_xlabel("k-Value")
ax.set_ylabel("R^2 Score")
ax.grid(True, lw=1.5, ls='--', alpha=0.75)
plt.show()
```



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 $k = n$?*

   A k-NN regression model that uses $k = n$ will return the mean of all n observations as the prediction value.

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

   An R^2 score of 0 means that the model is as accurate of a predictor as the mean value.

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

   A negative R^2 score means that our model is worse than the average. We observed this in the test set for k-values k=1 and k=1000.

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

   Yes. The training plot exhibits a downward trend with the highest R^2 score occurring at k=1. The test plot exhibits an inverted curve similar to an even function with the highest R^2 value being observed at k = 75. The R^2 value for the test plot was also negative for k=1, and the R^2 values approached 0 from k = 75 to k = 1000.

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

   When the value of k is low (k=1), the complexity of the fitted model is very high, thus giving it a strong fit with the training data, but having high variance and a much worse fit for the test data. As the k-value increases and complexity decreases towards k=75, the fitted model fits the test data much better and the R^2 value for the test data increases. As the value of k increases towards 1000, the complexity of the model lowers and the R^2 value for the training data decreases towards 0. From k= 75 onward the same pattern applies to the R^2 values for the test data, and the R^2 approaches 0 due to k=1000 being the equivalent of using the mean response value as a model.

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

   The best value of k is k = 75, and the corresponding test set R^2 value is 0.390310 while the corresponding training set R^2 value is 0.445392.

Question 3 [20 pts]

We next consider simple linear regression for the same train-test data sets, 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 variable 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. (use `suptitle`)

3.3. Report the R^2 score for the fitted model on both the training and test sets. You may notice something peculiar about how they compare.

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 a 99% confidence interval for the slope and intercept to be tighter or looser 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.

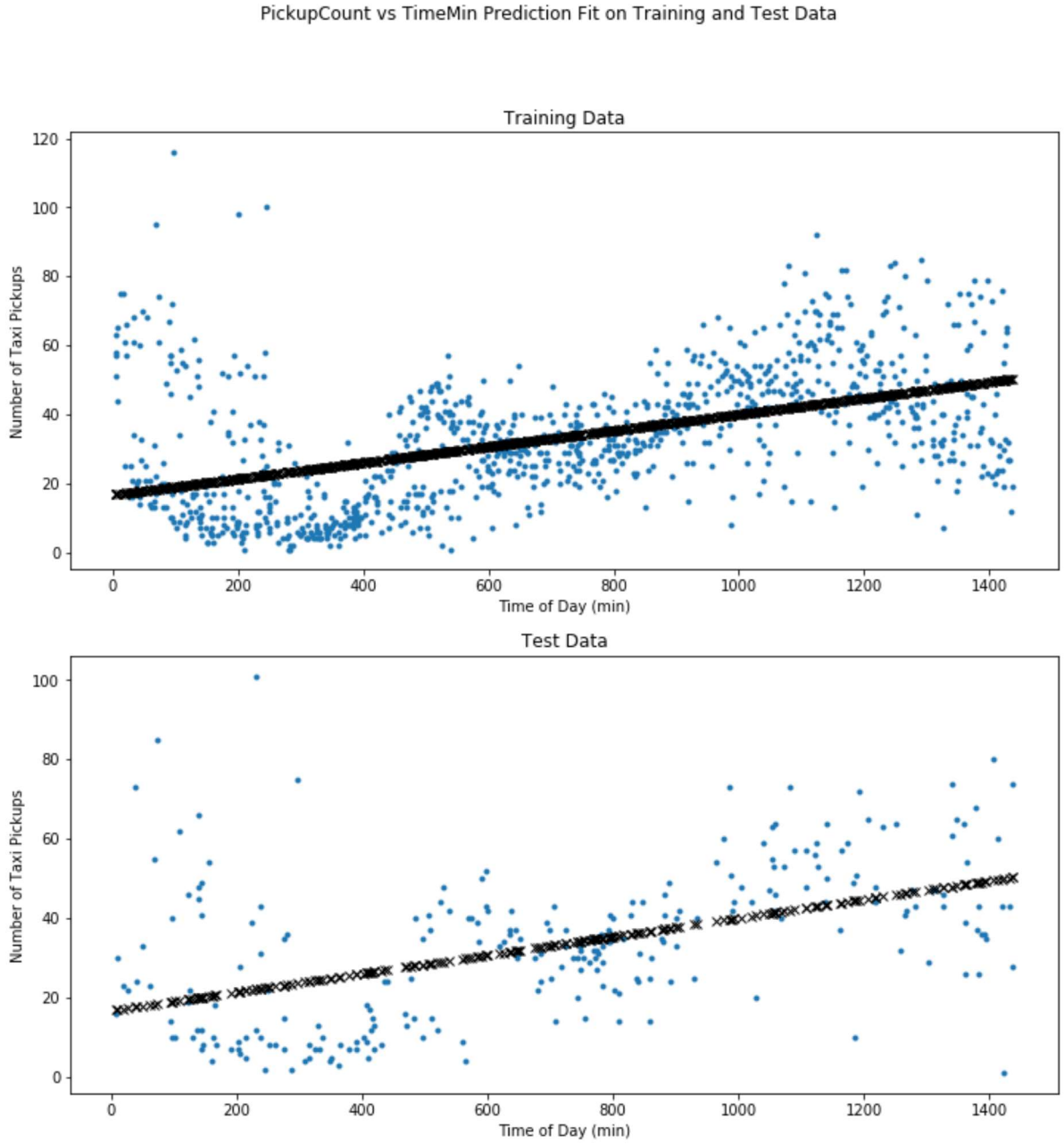
Answers

3.1

In [100]:
```
# Declare Variables
ols_train_timemin = sm.add_constant(train_data["TimeMin"].values)
ols_test_timemin = sm.add_constant(test_data["TimeMin"].values)

# Create class instance and model
OLSStats = sm.OLS(train_pickup, ols_train_timemin)
OLSModel = OLSStats.fit()
```

3.2

In [101]:
```
# Initialize subplots
fig, ax = plt.subplots(nrows=2, ncols=1, figsize = (12,12))

# Train, predicted values and scatter plot
ax[0].plot(train_timemin_reshape, train_pickup, ".")
ax[0].plot(train_timemin_reshape, OLSModel.predict(ols_train_timemin), "kx")

# Test, predicted values and scatter plot
ax[1].plot(test_timemin_reshape, test_pickup, ".")
ax[1].plot(test_timemin_reshape, OLSModel.predict(ols_test_timemin), "kx")

# Add labels
ax[0].set_title("Training Data")
ax[0].set_xlabel("Time of Day (min)")
ax[0].set_ylabel("Number of Taxi Pickups")
ax[1].set_title("Test Data")
ax[1].set_xlabel("Time of Day (min)")
ax[1].set_ylabel("Number of Taxi Pickups")

fig.suptitle("PickupCount vs TimeMin Prediction Fit on Training and Test Data"
)
plt.show()
```



3.3

In [102]:
```
# Store predicted values
train_pickup_predict = OLSModel.predict(ols_train_timemin)
test_pickup_predict = OLSModel.predict(ols_test_timemin)

# Store R^2 calculated values
train_r2 = r2_score(train_pickup, train_pickup_predict)
test_r2 = r2_score(test_pickup, test_pickup_predict)

print("\033[1mR^2\nTraining:\033[0m {0}\n\033[1mTest:\033[0m {1}".format(train
_r2, test_r2))
```

R^2
Training: 0.24302603531893352
Test: 0.240661535615741

3.4

In [103]:
```
# Print coefficients
print("\033[1mValues\033[0m")
print("\033[1mIntercept:\033[0m {0}\n\033[1mSlope:\033[0m {1}".format(OLSModel
.params[0], OLSModel.params[1]))
```

Values
Intercept: 16.750601427446817
Slope: 0.023335175692397344

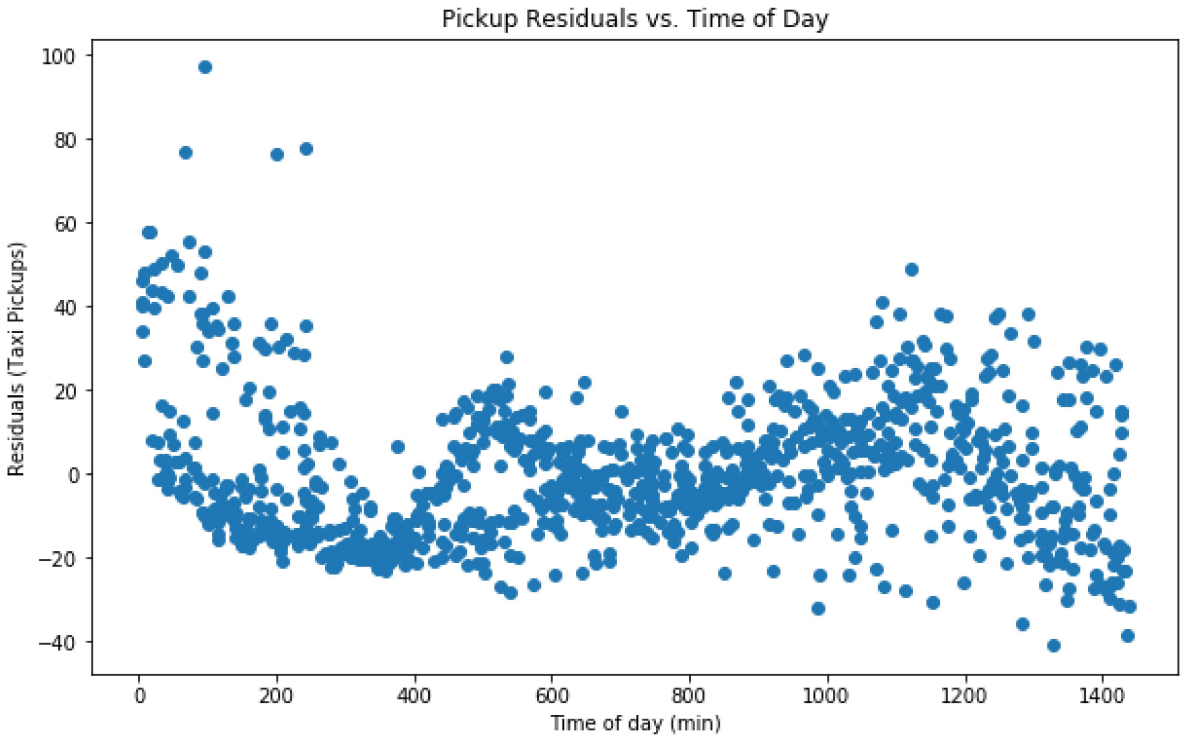
3.5

In [104]:
```
# Print confidence interval
conf_ints = OLSModel.conf_int()
print("\033[1m95% Confidence Intervals\033[0m")
print("\033[1mIntercept:\033[0m ({0}, {1})".format(conf_ints[0][0], conf_ints[
0][1]))
print("\033[1mSlope:\033[0m ({0}, {1})".format(conf_ints[1][0], conf_ints[1][1
]))
```

95% Confidence Intervals
Intercept: (14.67514134465737, 18.826061510236265)
Slope: (0.02077697281825772, 0.025893378566536968)

3.6

In [105]:
```
# Initialize subplot
fig, ax = plt.subplots(1,1, figsize=(10,6))

# Insert data, set labels
ax.scatter(train_timemin, OLSModel.resid)
ax.set_xlabel("Time of day (min)")
ax.set_ylabel("Residuals (Taxi Pickups)")
ax.set_title("Pickup Residuals vs. Time of Day")
plt.show()
```



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?*

   The test R^2 score from linear regression is worse than the best test R^2 value obtained with k-NN regression.

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

   The sign of the slope of the fitted linear model is positive, suggesting a positive relationship between the time of day and number of taxi pickups.

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

   Yes. Neither 95% confidence interval contains 0 so both parameters are significant. The proportion of the parameter estimates do not change by more than 30% over the entire confidence interval, so the current parameters are fairly reliable.

1. *Do you expect a 99% confidence interval for the slope and intercept to be tighter or looser than the 95% confidence intervals? Briefly explain your answer.*

   I expect a 99% confidence interval for the slope and intercept to be looser than the 95% confidence interval as to increase confidence in knowing what range of values the coefficients are you must decrease your precision of what the value may be.

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

   The residuals show an oscillating pattern, not random scatter. Therefore, the assumption of linearity is not valid for this data, as the error is not random for the predictions.

Question 4 [20 pts]: Roll Up Your Sleeves Show Some Class

We've seen Simple Linear Regression in action and we hope that you're convinced it works. In lecture we've thought about the mathematical basis for Simple Linear Regression. There's no reason that we can't take advantage of our knowledge to create our own implementation of Simple Linear Regression. We'll provide a bit of a boost by giving you some basic infrastructure to use. In the last problem, you should have heavily taken advantage of the `statsmodels` module. In this problem we're going to build our own machinery for creating Linear Regression models and in doing so we'll follow the `statsmodels` API pretty closely. Because we're following the `statmodels` API, we'll need to use python classes to create our implementation. If you're not familiar with python classes don't be alarmed. Just implement the requested functions/methods in the CS109OLS class that we've given you below and everything should just work. If you have any questions, ask the teaching staff.

4.1. Implement the `fit` and `predict` methods in the CS109OLS class we've given you below as well as the `CS109r2score` function that we've provided outside the class.

Hints:

1. `fit` should take the provided numpy arrays `endog` and `exog` and use the normal equations to calculate the optimal linear regression coefficients. Store those coefficients in self.params
2. In `fit` you'll need to calculate an inverse. Use np.linalg.pinv
3. `predict` should use the numpy array stored in self.exog and calculate an np.array of predicted values.
4. `CS109r2score` should take the true values of the response variable `y_true` and the predicted values of the response variable `y_pred` and calculate and return the R^2 score.
5. To replicate the statsmodel API your code should be able to be called as follows:

```
mymodel = CS109OLS(y_data, augmented_x_data)
mymodel.fit()
predictions = mymodel.predict()
R2score = CS109r2score(true_values, predictions)
```

4.2. As in 3.1 create a `CS109OLS` class instance and fit a Linear Regression model on the training set (`train_data`). Store your model in the variable `CS109OLSModel`. Remember that as with sm.OLS your class should assume you want to fit an intercept as part of your linear model (so you may need to add a constant column to your predictors).

4.3 As in 3.2 Overlay a scatter plot of the actual values of `PickupCount` vs. `TimeMin` on the training set with a scatter plot of `PickupCount` vs predictions of `TimeMin` from your `CS109OLSModel` Linear Regression model on the training set. Do the same for the test set. You should have one figure with two subplots, one subplot for the training set and one for the test set. How does your figure compare to that in 3.2?

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. (use `suptitle`)

4.4. As in 3.3, report the R^2 score for the fitted model on both the training and test sets using your `CS109OLSModel`. Make sure to use the `CS109r2score` that you created. How do the results compare to the the scores in 3.3?

4.5. as in 3.4, report the slope and intercept values for the fitted linear model your `CS109OLSModel`. How do the results compare to the the values in 3.4?

Answers

4.1

In [106]:
```
class CS109OLS(object):

    def __init__(self, endog = [], exog = []):
        """ Initializes CS109OLS object instance.

          Args:
            self: The instance object
            endog: The numpy array of y-values
            exog: The numpy array of x-values
        """
        ## Make sure you initialize self.params
        self.params = []

        ## store exog and endog in instance variables
        self.endog = np.array(endog)
        self.exog = np.array(exog)


    def fit(self):
        """ Uses normal equations to calculate optimal linear regression coeff
icients. Stores the results in self.params.

          Args:
            self: the instance object.
        """

        # Uses normal equation with self.exog and self.endog to calculate line
ar regression coefficients
        # Stores the result in self.params
        self.params = np.linalg.inv(self.exog.T.dot(self.exog)).dot(self.exog.
T).dot(self.endog)
        return self

    def predict(self, exog=None):
        """ Returns a numpy array of predicted y-values utilizing matrix dot p
roducts

          Args:
            self: The instance object.
            exog: The numpy array of x-values, assigns value of None if no arg
ument provided.

          Returns: A numpy array of predicted y-values corresponding to the x-
values
        """
        # Check if the linear regression coefficients have been calculated
        if not np.array(self.params).size:
            raise(Exception("fit() has not been called on OLS Model!"))

        # Check if exog is none and initialize it to self.exog if it is
        if exog  is None:
            exog = self.exog


        # Calculates predictions by taking the dot product of exog and self.pa
rams. Returns them as a numpy array
```

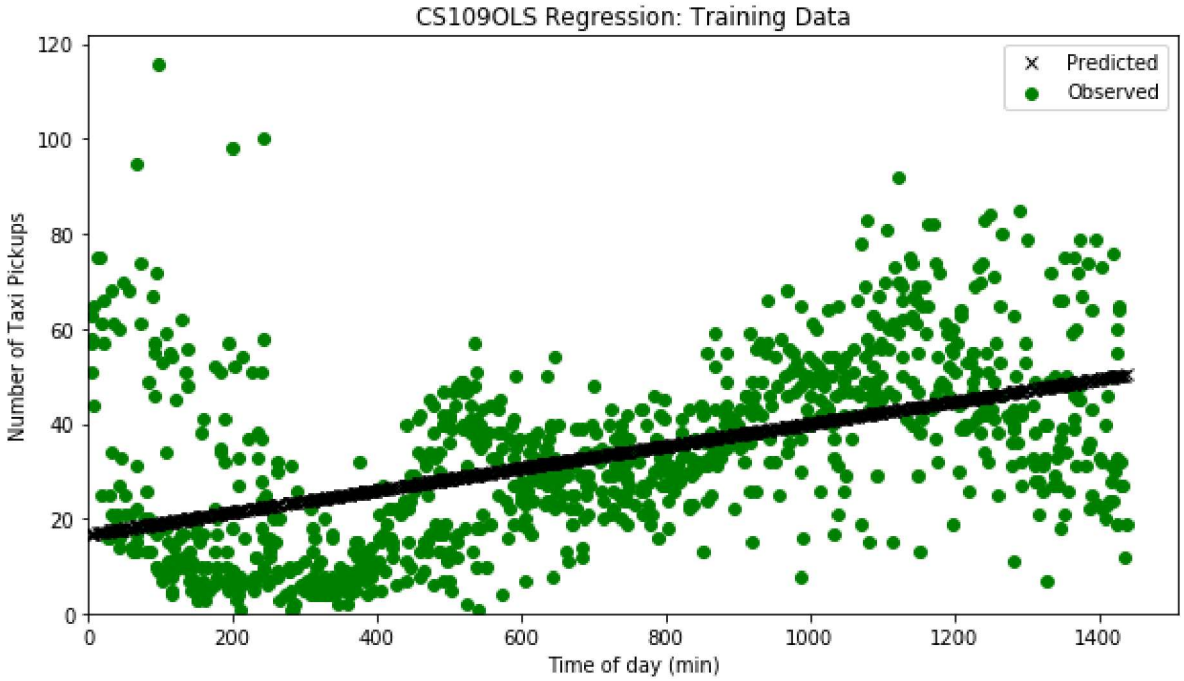
```
        return exog.dot(self.params)



def CS109r2score(y_true, y_pred):
    """ Calculates an R^2 value for data.

      Args:
          y_true: A numpy array of the observed y-values.
          y_pred: A numpy array of the corresponding predicted y-values.

      Returns:
          Returns the R^2 value as a float.
    """

    # y_true should be your actual y data (endogenous data)
    # y_pred should be the corresponding predictions from your model

    # calculate the r^2 score and return it
    return 1  - np.sum((y_pred - y_true) ** 2) / np.sum((np.mean(y_true) - y_t
rue) ** 2)
```

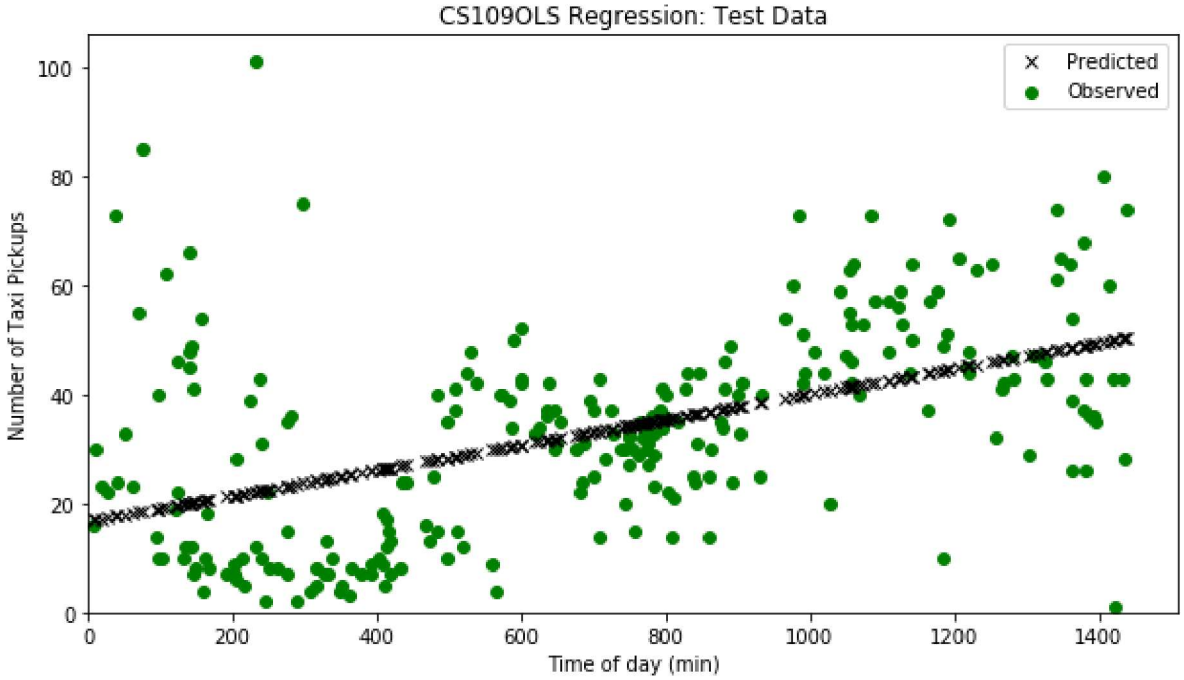
4.2

In [107]:
```
# Create an instance of the CS109OLS class and create the model with fit()
CS109OLSModel = CS109OLS(train_pickup, ols_train_timemin)
CS109OLSModel.fit();
```

4.3

In [108]:
```
# Declare prediction variables
cs109_train_predict = CS109OLSModel.predict()
cs109_test_predict = CS109OLSModel.predict(ols_test_timemin)

# Initialize subplot
fig, ax = plt.subplots(2,1, figsize=(10,12))

# Create scatter plot of train observed and predicted pickup values
ax[0].scatter(train_timemin, train_pickup, c="g", label="Observed")
ax[0].plot(train_timemin,cs109_train_predict, "kx", label="Predicted")

# Set origin to intersection of axes
ax[0].set_ylim(ymin=0)
ax[0].set_xlim(xmin=0)

#Set labels
ax[0].set_xlabel("Time of day (min)")
ax[0].set_ylabel("Number of Taxi Pickups")
ax[0].set_title("CS109OLS Regression: Training Data")

# Create plot of observed and predicted values
ax[1].scatter(test_timemin, test_pickup, c="g", label="Observed")
ax[1].plot(test_timemin, cs109_test_predict, "kx", label="Predicted")

# Set origin to intersection of axes
ax[1].set_ylim(ymin=0)
ax[1].set_xlim(xmin=0)

#Set labels
ax[1].set_xlabel("Time of day (min)")
ax[1].set_ylabel("Number of Taxi Pickups")
ax[1].set_title("CS109OLS Regression: Test Data")

# Set legends, show plots
ax[0].legend()
ax[1].legend()
plt.show()
```


CS109OLS Regression: Training Data
Predicted
Observed
Number of Taxi Pickups
Time of day (min)
0
20
40
60
80
100
120
0
200
400
600
800
1000
1200
1400


CS109OLS Regression: Test Data
Predicted
Observed
Number of Taxi Pickups
Time of day (min)
0
20
40
60
80
100
0
200
400
600
800
1000
1200
1400

4.4

In [109]:
```
# Store predicted values
cs_train_pickup_predict = CS109OLSModel.predict(ols_train_timemin)
cs_test_pickup_predict = CS109OLSModel.predict(ols_test_timemin)

# Store R^2 calculated values
cs_train_r2 = CS109r2score(train_pickup, train_pickup_predict)
cs_test_r2 = CS109r2score(test_pickup, test_pickup_predict)

# Print results
print("\033[1mR^2\nTraining:\033[0m {0}\n\033[1mTest:\033[0m {1}".format(cs_tr
ain_r2, cs_test_r2))
print("\nThese values are exactly the same as those calculated in 3.3")
```

R^2
Training: 0.24302603531893352
Test: 0.240661535615741

These values are exactly the same as those calculated in 3.3

4.5

In [110]:
```
# Print coefficients
print("\033[1mValues\033[0m")
print("\033[1mIntercept:\033[0m {0}\n\033[1mSlope:\033[0m {1}".format(CS109OLS
Model.params[0], CS109OLSModel.params[1]))
print("\nThe intercept and slope values of the CS109OLSModel are the exact sam
e as the values reported in 3.4")
```

Values
Intercept: 16.750601427446803
Slope: 0.02333517569239736

The intercept and slope values of the CS109OLSModel are the exact same as the
 values reported in 3.4

Question 5

.

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.

5.1. We've provided you with two files `outliers_train.csv` and `outliers_test.csv` 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?

5.2. 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`.

5.3. 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 ahead of time 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?

5.4 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 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`.

Hint:

```
1. What measure might you use to compare the performance of different Linear Regres
sion models?
```

5.5 Create a figure with two subplots. In one subplot include a visualization of the Linear Regression line from the full training set overlayed on the test set data in `outliers_test`. In the other subplot include a visualization of the Linear Regression line from the training set data with outliers removed overlayed on the test set data in `outliers_test`. Visually which model fits the test set data more closely?

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

5.7. One potential problem with the brute force outlier detection approach in 5.3 and the heuristic algorithm constructed in 5.4 is that they assume prior knowledge of the number of outliers. In general we can't expect to know ahead of time the number of outliers in our dataset. Alter the algorithm you constructed in 5.4 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 that your algorithm found them). 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? (i.e. is there any reason not to try to identify one outlier at a time)
2. If you plotted an R^2 score for each step the algorithm, what might that plot tell you about stopping conditions?
3. As mentioned earlier we don't know ahead of time how many outliers to expect in the dataset or know mathematically how we'd define a point as an outlier. For this general algorithm, whatever measure you use to determine a point's impact on the Linear Regression model (e.g. difference in R^2, size of the residual or maybe some other measure) you may want to determine a tolerance level for that measure at every step below which your algorithm stops looking for outliers.
4. You may also consider the maximum possible number of outliers it's reasonable for a dataset of size n to have and use that as a cap for the total number of outliers identified (i.e. would it reasonable to expect all but one point in the dataset to be an outlier?)

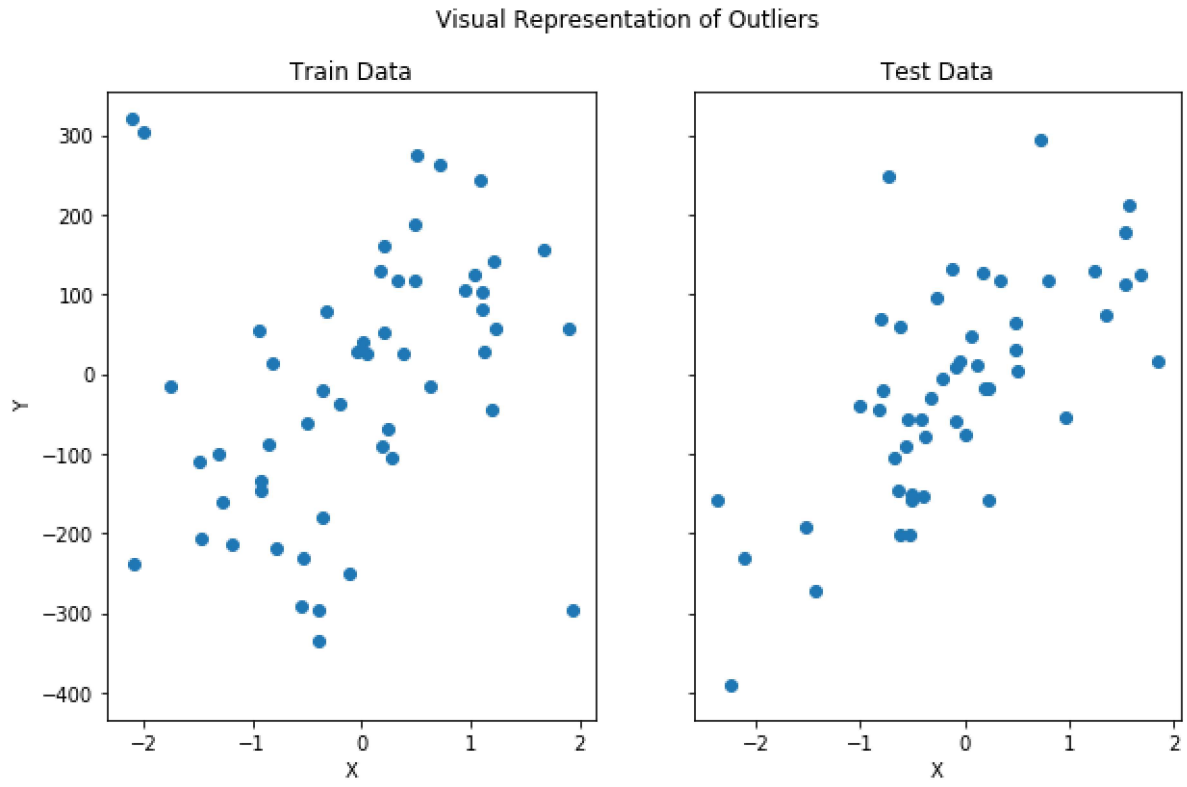
5.8. Run your algorithm in 5.7 on the training set data.

1. What outliers does it identify?
2. How do those outliers compare to the outliers you found in 5.4?
3. How does the general outlier-free Linear Regression model you created in 5.7 perform compared to the simple one in 5.4?

Answers

5.1

In [111]:
```
# read in data
train = pd.read_csv("outliers_train.csv")
test = pd.read_csv("outliers_test.csv")

# initialize subplots
fig, ax = plt.subplots(1,2, figsize = (10,6), sharey=True)
ax[0].scatter(train["X"], train["Y"])
ax[1].scatter(test["X"], test["Y"])

# Labels
ax[0].set_xlabel("X")
ax[0].set_ylabel("Y")
ax[0].set_title("Train Data")
ax[1].set_xlabel("X")
ax[1].set_title("Test Data")
fig.suptitle("Visual Representation of Outliers")
plt.show()
```



What does a visual inspection of training set tell you about the existence of outliers in the data?

It appears there are two potential outliers that is significantly higher than the rest of training data and one potential outlier that is significantly lower than the rest of the training data. The three potential outliers do not appear to be present in the test data.

5.2

In [112]:
```
# Create the OutlierOLSModel
x_outlier = sm.add_constant(train['X'])
OutlierOLS= sm.OLS(train['Y'], x_outlier)
OutlierOLSModel = OutlierOLS.fit()
```

5.3

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 ahead of time 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?

Work: ***53 C 50:***
*$53!/(50!* 3!) = 23426$ *

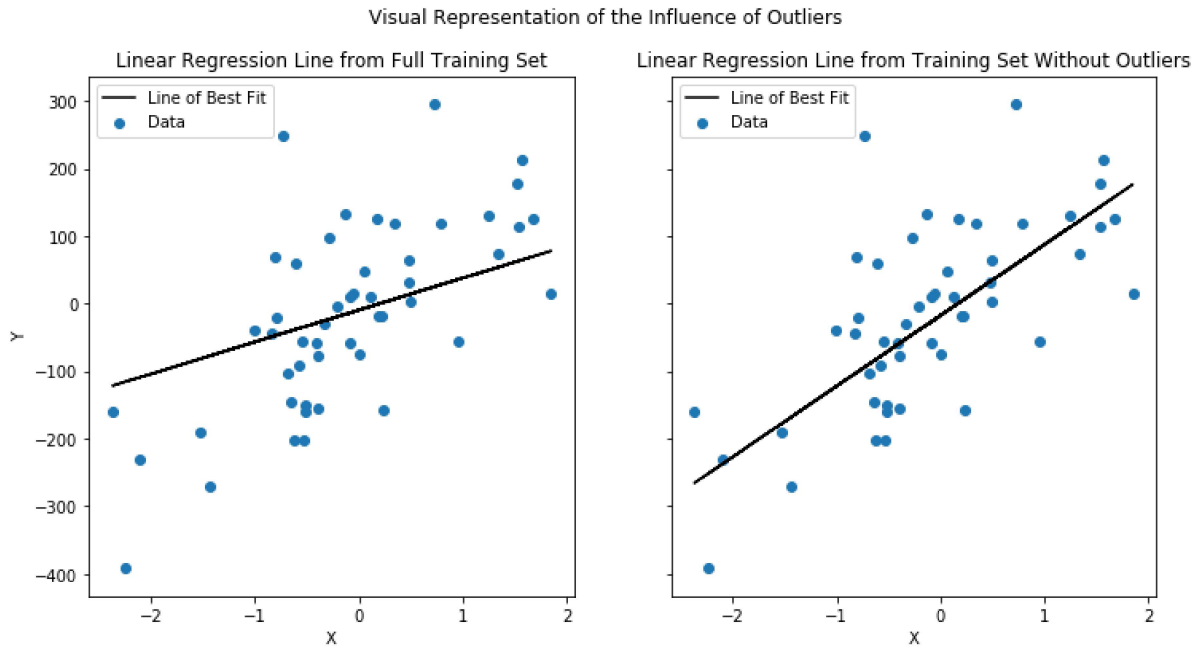
Using this brute force method, you would have to calculate Linear Regression coefficient 23426 times

5.4

In [113]:
```
def find_outliers_simple(dataset_x, dataset_y):
    """ Finds the three outlier candidates in training data using the linear r
egression residuals

      Args:
        dataset_x: a numpy array of features variable values
        dataset_y: a numpy array of response variable values

      Returns:
        outlier_indices: A numpy list representing the indices of the outliers
 in the original data set.
    """
    # Initialize dataframe
    outdf = pd.DataFrame(columns=("X", "Y", "resid"))
    outdf["X"] = dataset_x
    outdf["Y"] = dataset_y

    # Create model/get y prediction values
    x_outliers = sm.add_constant(dataset_x)
    OutOLSModel = sm.OLS(dataset_y, dataset_x)
    results_outliers = OutOLSModel.fit()
    y_predict = results_outliers.predict()

    # Calculate residuals
    outdf["resid"]  = np.abs(dataset_y - y_predict)

    # Return outlier indices
    outlier_indices = outdf.sort_values("resid", ascending=False).iloc[:3].ind
ex.values
    return outlier_indices
```

5.5

In [114]:
```
# get outliers
outliers = find_outliers_simple(train["X"], train["Y"])

# Create dataframe with outliers removed
new_train = train.drop(outliers)

# Create outlier free regression model
no_outlier_x = sm.add_constant(new_train["X"].values)
OutlierFreeSimple = sm.OLS(new_train["Y"].values, no_outlier_x)
OutlierFreeSimpleModel = OutlierFreeSimple.fit()

# Declare variables
test_outlier_x = sm.add_constant(test["X"].values)
no_outlier_predictions = OutlierFreeSimpleModel.predict(test_outlier_x)
outlier_predictions =  OutlierOLSModel.predict(test_outlier_x)

# Initialize subplots
fig, ax = plt.subplots(1,2, figsize = (12,6), sharey=True)

# Create subplot of data with outliers
ax[0].scatter(test["X"], test["Y"], label="Data")
ax[0].plot(test["X"], outlier_predictions, "k-", label="Line of Best Fit")

# Create subplot with outliers removed
ax[1].scatter(test["X"], test["Y"], label="Data")
ax[1].plot(test["X"].values, no_outlier_predictions, "k-", label="Line of Best
 Fit")

# Labels
ax[0].set_xlabel("X")
ax[0].set_ylabel("Y")
ax[0].set_title("Linear Regression Line from Full Training Set")
ax[1].set_xlabel("X")
ax[1].set_title("Linear Regression Line from Training Set Without Outliers")
fig.suptitle("Visual Representation of the Influence of Outliers")
ax[0].legend()
ax[1].legend()
plt.show()
```



5.6

In [115]:
```
# Store R^2
no_out_r2 = r2_score(test["Y"], no_outlier_predictions)
out_r2 = r2_score(test["Y"], outlier_predictions)

# Print results
print("\033[1mR^2\nModel With No Outliers:\033[0m {0}\n\033[1mModel With Outli
ers:\033[0m {1}".format(no_out_r2, out_r2))
print("\nThe model with no outliers produces a better R^2")
```

```
R^2
Model With No Outliers: 0.4529566870167582
Model With Outliers: 0.34085656043405654

The model with no outliers produces a better R^2
```

5.7

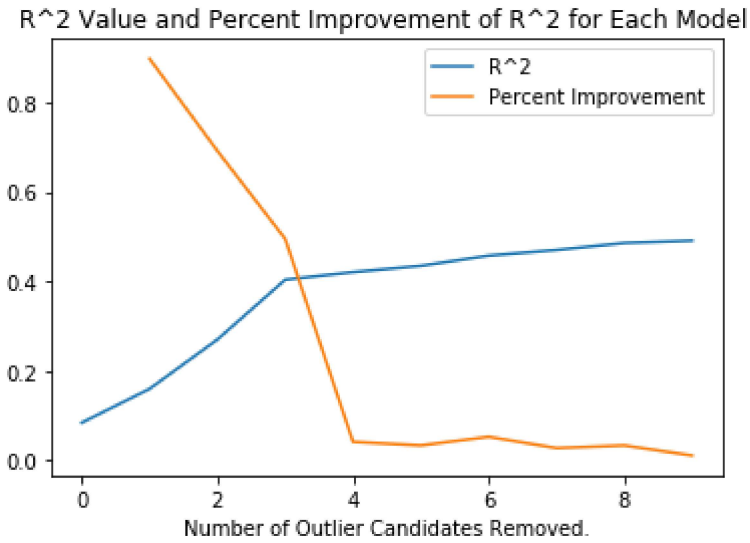
In [116]:
```
# Import summary_table
from statsmodels.stats.outliers_influence import summary_table

def find_outliers_general(dataset_x, dataset_y, r_2_threshold = 0.02, test_alg
orithm=False):
    """ Finds outlier candidates in the data

      Args:
        dataset_x: a numpy array of features variable values
        dataset_y: a numpy array of response variable values

      Returns:
        outlier_indices: A numpy list representing the indices of the outliers
 in the original data set.
    """

    # For testing
    if(test_algorithm):
        # Initialize plot
        fig, ax = plt.subplots(1,1, figsize=(6,4))
        x_es = []
        rsquares = []

    # Create DataFrame
    df = pd.DataFrame(columns=["X", "Y", "index_orig"])
    df["X"] = dataset_x
    df["Y"] = dataset_y
    df["index_orig"] = df.index.values

    # Create an OLS Model and extract 95% confidence interval bound data for e
ach data point
    OLSMod = sm.OLS(df["Y"], sm.add_constant(df["X"]))
    OLSModel = OLSMod.fit()
    _, data, _ = summary_table(OLSModel, alpha=0.05)

    # For testing
    if(test_algorithm):
        x_es.append(0)
        rsquares.append(OLSModel.rsquared)


    # Initialize the r2 value
    r2_old = OLSModel.rsquared

    # initialize array of indices and variables used in while loop condition
    outlier_ind = []
    row = 0
    bound = len(df)

    while row < bound:
        # Store confidence interval bounds for each predicted point
        predict_ci_low, predict_ci_upp = data[row,6:8].T

        # Determine if outlier criteria is met
        if (df.loc[row]["Y"] > predict_ci_upp) or (df.loc[row]["Y"] < predict_
ci_low):
```

```
            # Store the index of the outlier candidate
            outlier_candidate = int(df.loc[row]["index_orig"])

            # Remove outlier observation from dataframe and reset index
            df = df.drop(row, axis="rows").reset_index(drop=True)

            # Create a new OLS model for the data without the outlier
            OLSMod = sm.OLS(df["Y"], sm.add_constant(df["X"]))
            OLSModel = OLSMod.fit()

            # Store the new model's R^2 value
            r2_new = OLSModel.rsquared

            # Decide if the improvment in model fit is worth dropping outlier
 candidate (meets the threshold)
            if(r2_new - r2_old < r_2_threshold):
                break

            # Add outlier to list of outliers
            outlier_ind.append(outlier_candidate)

            # Create a new summary_table based on the new model and also updat
e r2_old
            _, data, _ = summary_table(OLSModel, alpha=0.05)
            r2_old =r2_new

            # Reset while loop iterator and change condition
            row = 0
            bound = len(df)

            # For testing
            if(test_algorithm):
                x_es.append(len(outlier_ind))
                rsquares.append(r2_new)

        # Increment iterator
        row += 1

    # Cast the list of indices to a numpy array and sort them
    outlier_indices = np.sort(np.array(outlier_ind))

    # For testing
    if(test_algorithm):

        # Plots the R^2 Values and percent improvement for each outlier remove
d
        percent_increase = []
        for i in range(len(rsquares) - 1):
            percent_increase.append((rsquares[i+1] - rsquares[i])/rsquares[i])
        ax.plot(x_es, rsquares, label="R^2")
        ax.plot(x_es[1:],percent_increase, label="Percent Improvement")
        ax.legend()
        ax.set_title("R^2 Value and Percent Improvement of R^2 for Each Model
 ")
        ax.set_xlabel("Number of Outlier Candidates Removed.")
```

```
    # Return the indices
    return outlier_indices

# Call find_outliers_general
outlier_indices = find_outliers_general(train["X"], train["Y"])

# Create the new model
train_no_outliers = train.drop(outlier_indices)
OutlierFreeGeneralMod =  sm.OLS(train_no_outliers["Y"], train_no_outliers["X"
])
OutlierFreeGeneralModel = OutlierFreeGeneralMod.fit()
```

In [117]:
```
# For testing
# Print R^2 values
print(outlier_indices)
print(r2_score(test["Y"], OutlierFreeGeneralModel.predict(test["X"])))

# Print Test
test = find_outliers_general(train["X"], train["Y"], 0.005, True)
```

```
[50 51 52]
0.4579491642913984
```



5.8

1. *What outliers does it identify?*

   It identifies rows $[50, 51, 52]$ as outliers in the dataset

1. *How do those outliers compare to the outliers you found in 5.4?*

   All outliers in 5.4 were found by our algorithm. No additional outlier candidates were considered outliers in our algorithm since their removal from the model did not incur a significant increase in R^2.

1. *How does the general outlier-free Linear Regression model you created in 5.7 perform compared to the simple one in 5.4?*

   The general outlier-free Linear Regression model performs the exact same as the simple one in 5.4. The same data is used in the same model fitting algorithm. The R^2 values are also the same.

In [118]:
```
from IPython.core.display import HTML
def css_styling(): styles = open("cs109.css", "r").read(); return HTML(styles)
css_styling()
```

Out[118]: