CS 109A/STAT 121A/AC 209A/CSCI E-109A: Homework 2

Linear and k-NN Regression

Harvard University
Fall 2017
Instructors: Pavlos Protopapas, Kevin Rader, Rahul Dave, Margo Levine

---

INSTRUCTIONS

- To submit your assignment follow the instructions given in canvas.
- Restart the kernel and run the whole notebook again before you submit.
- Do not include your name(s) in the notebook even if you are submitting as a group.
- If you submit individually and you have worked with someone, please include the name of your [one] partner below.

Import libraries

In [1]:
```
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
import statsmodels.api as sm
from statsmodels.api import OLS
from sklearn.preprocessing import PolynomialFeatures
%matplotlib inline
import seaborn as sns
```

```
D:\ProgramData\Anaconda3\envs\py36\lib\site-packages\statsmodels\compat\pandas.py:56: FutureWarning: The pandas.co
re.datetools module is deprecated and will be removed in a future version. Please use the pandas.tseries module in
stead.
  from pandas.core import datetools
```

Predicting Taxi Pickups in NYC

In this homework, we will explore k-nearest neighbor, linear and polynomial 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 files `dataset_1_train.txt` and `dataset_1_test.txt` as separate training and test sets. The first column in each file 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 during different days in Jan 2015 (randomly sampled across days and time of that day).

We will fit regression models that use the time of the day (in minutes) as a predictor and predict the average number of taxi pick ups 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.

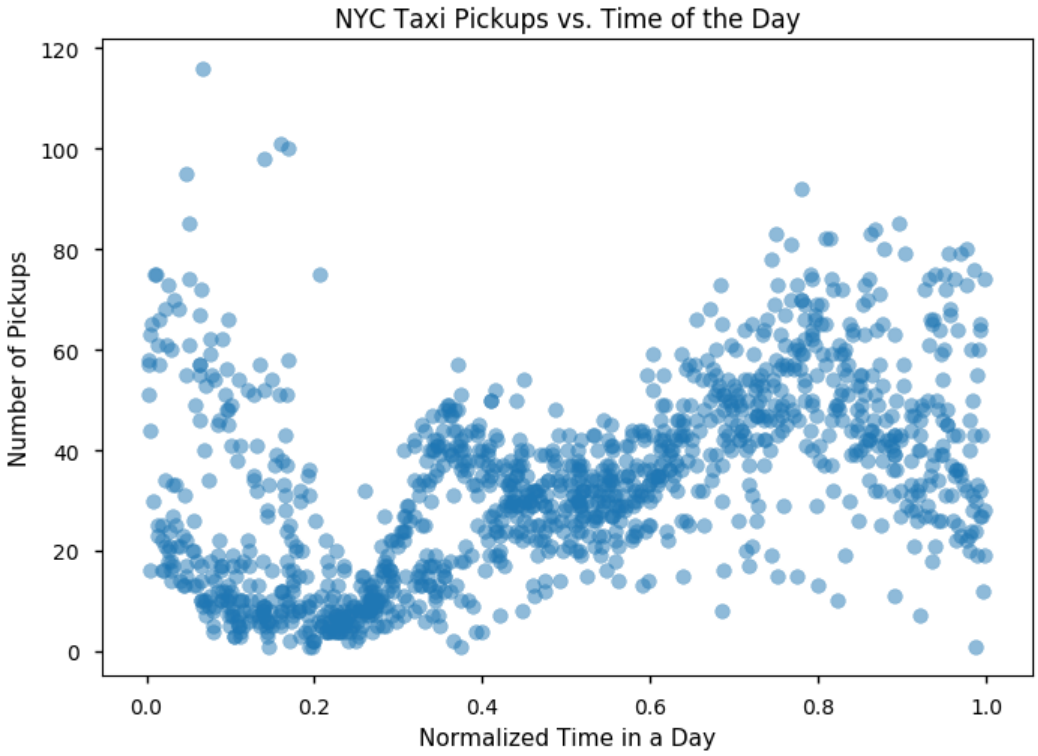
Data Normalization: As a first step, we suggest that you normalize the `TimeMin` predictor to a value between 0 and 1. This can be done by dividing the time column in the training and test sets by 1440 (i.e. the maximum value the predictor can take). This normalization step would be particularly helpful while fitting polynomial regression models on this data.

Part (0): EDA

Generate a scatter plot of the training data points, with the time of the day on the X-axis and the number of taxi pickups on the Y-axis. Does the pattern of taxi pickups make intuitive sense to you?

In [2]:
```
nyc_pickup_train = pd.read_csv('data/dataset_1_train.txt')
nyc_pickup_test = pd.read_csv('data/dataset_1_test.txt')

nyc_pickup_train['TimeMin'] = nyc_pickup_train['TimeMin'].apply(lambda t: t/1440.0)
nyc_pickup_test['TimeMin'] = nyc_pickup_test['TimeMin'].apply(lambda t: t/1440.0)

nyc_pickup_train = nyc_pickup_train.sort_values(by = 'TimeMin')
nyc_pickup_test = nyc_pickup_test.sort_values(by = 'TimeMin')

time_train = nyc_pickup_train['TimeMin'].values
pickup_train = nyc_pickup_train['PickupCount'].values
time_test = nyc_pickup_test['TimeMin'].values
pickup_test = nyc_pickup_test['PickupCount'].values

time_train = time_train.reshape(len(time_train), 1)
pickup_train = pickup_train.reshape(len(pickup_train), 1)
time_test = time_test.reshape(len(time_test), 1)
pickup_test = pickup_test.reshape(len(pickup_test), 1)

time_all = np.concatenate((time_train, time_test), axis = 0)
pickup_all = np.concatenate((pickup_train, pickup_test), axis = 0)
```

In [3]:
```
sns.reset_defaults()
sns.set_context('notebook')
plt.scatter(time_all, pickup_all, alpha = 0.5)
plt.xlabel('Normalized Time in a Day')
plt.ylabel('Number of Pickups')
plt.title('NYC Taxi Pickups vs. Time of the Day')
plt.show()
```



Part (a): k-Nearest Neighbors

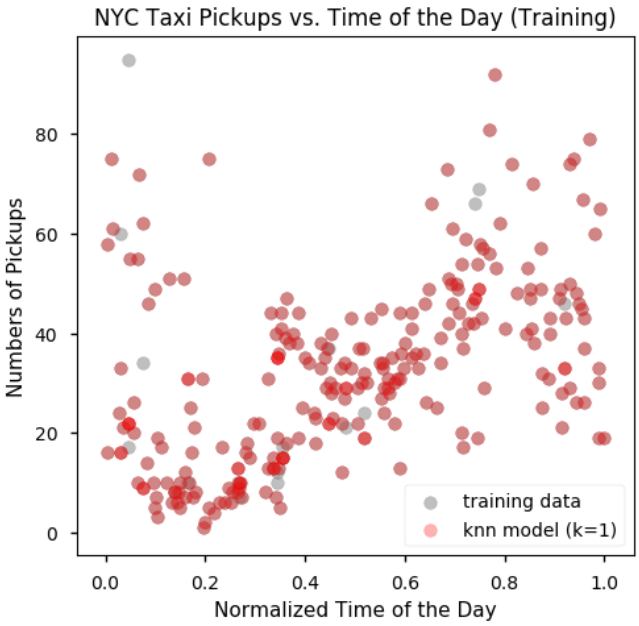
We begin with k-Nearest Neighbors (k-NN), a non-parametric regression technique. You may use `sklearn`'s built-in functions to run k-NN regression. Create a `KNeighborsRegressor` object, use the `fit` method in the object to fit a k-NN regressor model, use the `predict` method to make predictions from the model, and the `score` method to evaluate the R^2 score of the model on a data set.

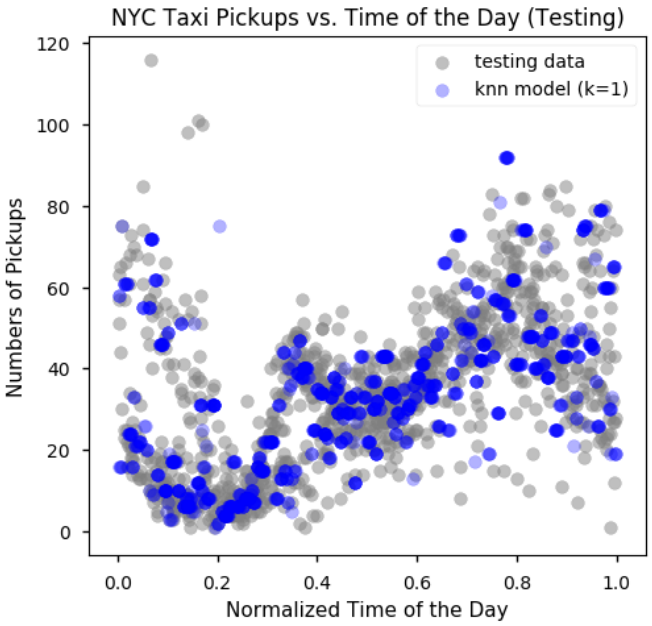
- Fit k-NN regression models:
  - Fit a k-NN regression model to the training set for different values of k (e.g. you may try out values 1, 2, 10, 25, 50, 100 and 200).
  - If you are using `sklearn`'s built-in functions for k-NN regression, explain what happens when you invoke the `fit` function.
  - If n is the number of observations in the training set, what can you say about a k-NN regression model that uses $k = n$?
- Visualize the fitted models:
  - Generate a scatter plot of the training data points, and in the same figure, also generate line plots of the predicted values $\hat{y}$ from each fitted model as a function of the predictor variable x. (*Hint:* you will want to sort the x values before plotting.)
  - How does the value of k effect the fitted model?
- Evaluate the fitted models:
  - Compute the R^2 score for the fitted models on both the training and test sets. Are some of the calculated R^2 values negative? If so, what does this indicate? What does a R^2 score of 0 mean?
  - Make plots of the training and test R^2 values as a function of k. Do the training and test R^2 plots exhibit different trends? Explain how the value of k influences the training and test R^2 values.

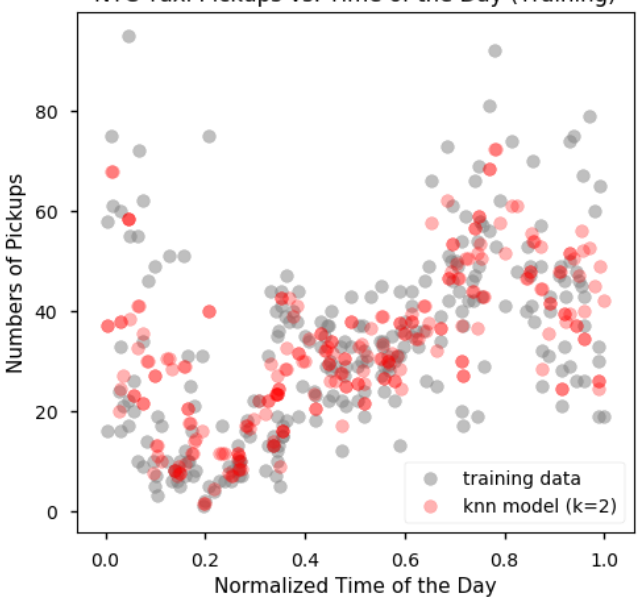
In [4]:
```
k_values = [1, 2, 10, 25, 50, 100, 200]
knn_models = [];
for i in range(len(k_values)):
    knn_model = KNeighborsRegressor(n_neighbors = k_values[i])
    # the fit function finds the nearest k neighbors of time_train, and take the average of
    # their corresponding pickup_train values as the predictted value
    knn_model.fit(time_train, pickup_train)
    knn_models.append(knn_model)
```

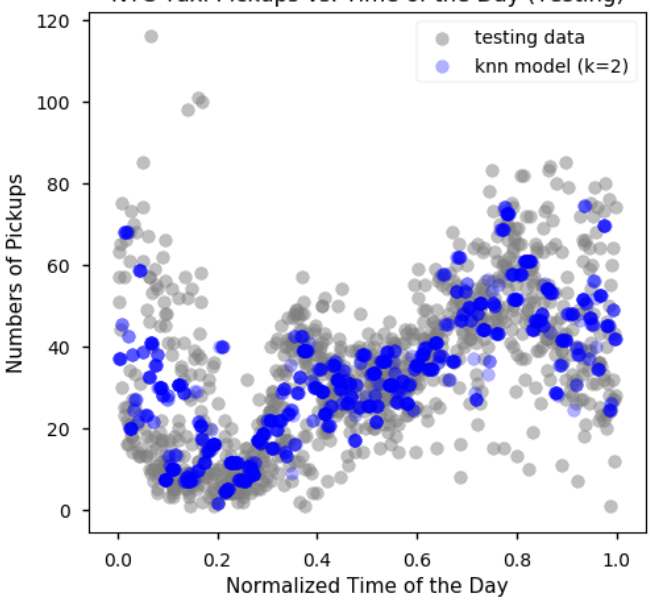
If k = n, then the prediction value of any x will be the average of the entire training set

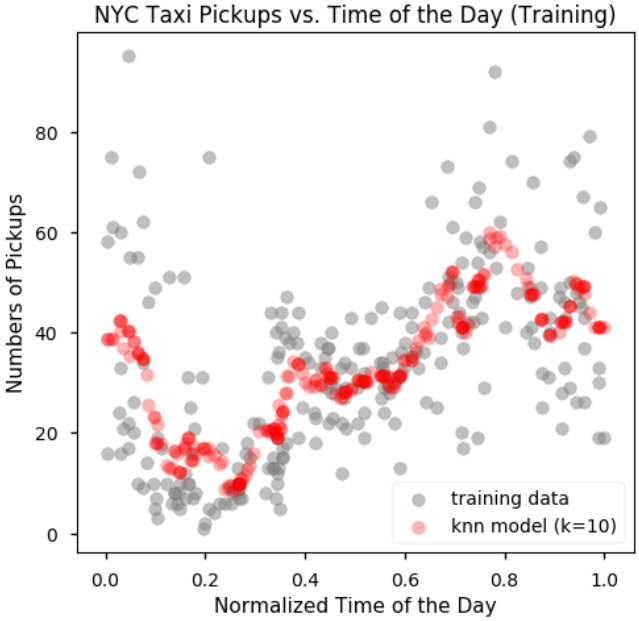
In [5]:
```
fig, ax = plt.subplots(len(k_values), 2, figsize=(12, 6*len(k_values)))
for i in range(len(k_values)):
    ax[i][0].scatter(time_train, pickup_train, color='gray', alpha=0.5, label='training data')
    ax[i][0].scatter(time_train, knn_models[i].predict(time_train), color='red', \
                     alpha=0.3, label='knn model (k=%d)' % k_values[i])
    ax[i][0].set_xlabel('Normalized Time of the Day')
    ax[i][0].set_ylabel('Numbers of Pickups')
    ax[i][0].set_title('NYC Taxi Pickups vs. Time of the Day (Training)')
    ax[i][0].legend(loc='best')

    ax[i][1].scatter(time_test, pickup_test, color='gray', alpha=0.5, label='testing data')
    ax[i][1].scatter(time_test, knn_models[i].predict(time_test), color='blue', \
                     alpha=0.3, label='knn model (k=%d)' % k_values[i])
    ax[i][1].set_xlabel('Normalized Time of the Day')
    ax[i][1].set_ylabel('Numbers of Pickups')
    ax[i][1].set_title('NYC Taxi Pickups vs. Time of the Day (Testing)')
    ax[i][1].legend(loc='best')

plt.show()
```

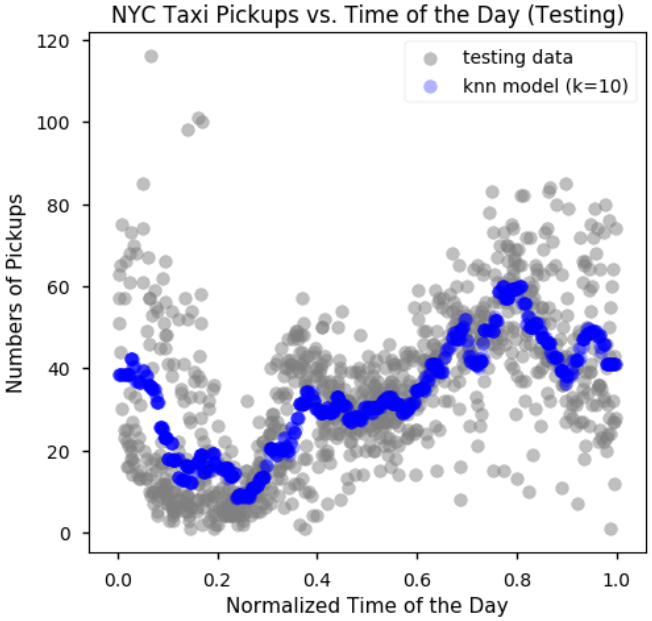

NYC Taxi Pickups vs. Time of the Day (Training)
Numbers of Pickups
Normalized Time of the Day
training data
knn model (k=1)

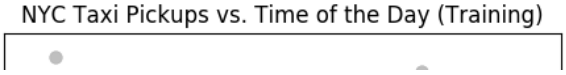

NYC Taxi Pickups vs. Time of the Day (Testing)
Numbers of Pickups
Normalized Time of the Day
testing data
knn model (k=1)

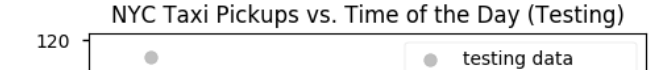

NYC Taxi Pickups vs. Time of the Day (Training)
Numbers of Pickups
Normalized Time of the Day
training data
knn model (k=2)

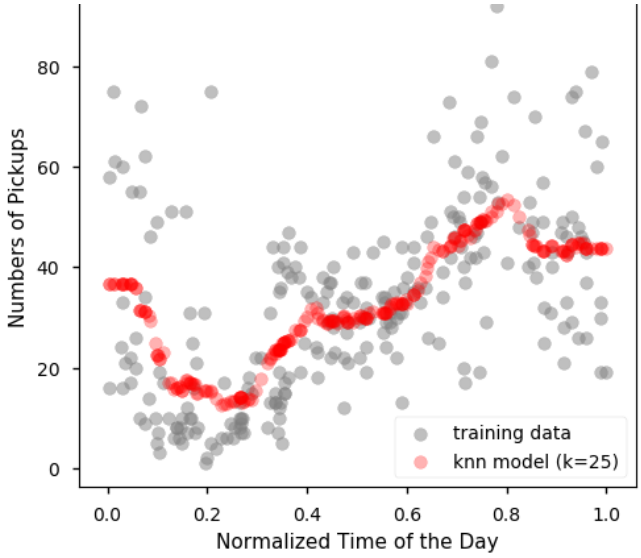

NYC Taxi Pickups vs. Time of the Day (Testing)
Numbers of Pickups
Normalized Time of the Day
testing data
knn model (k=2)

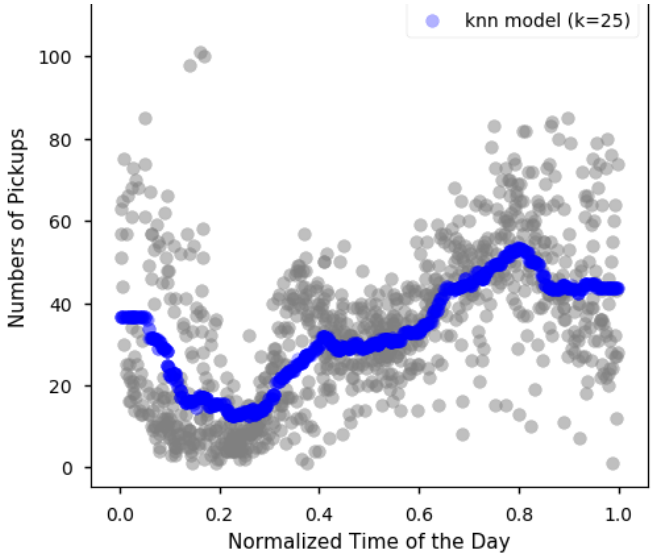

NYC Taxi Pickups vs. Time of the Day (Training)
Numbers of Pickups
Normalized Time of the Day
training data
knn model (k=10)


NYC Taxi Pickups vs. Time of the Day (Testing)
Numbers of Pickups
Normalized Time of the Day
testing data
knn model (k=10)


NYC Taxi Pickups vs. Time of the Day (Training)

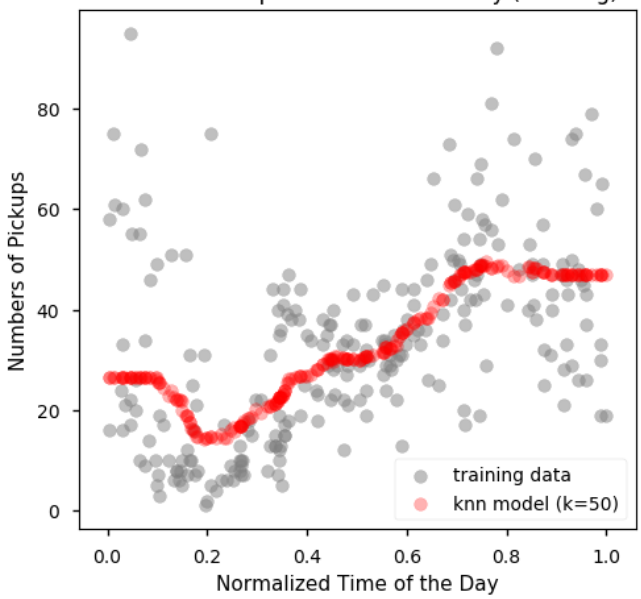

NYC Taxi Pickups vs. Time of the Day (Testing)
testing data

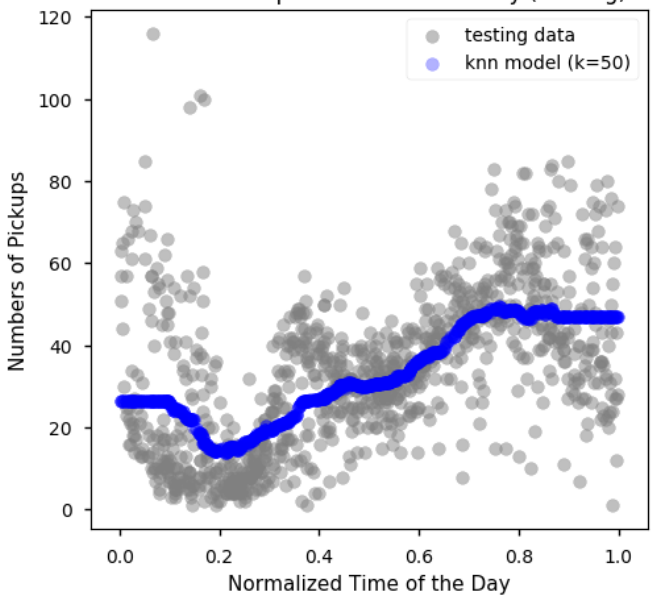




NYC Taxi Pickups vs. Time of the Day (Training)

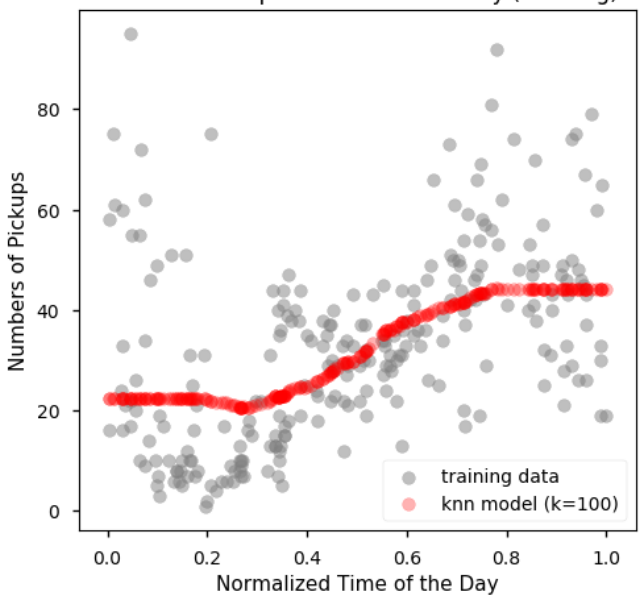
NYC Taxi Pickups vs. Time of the Day (Testing)

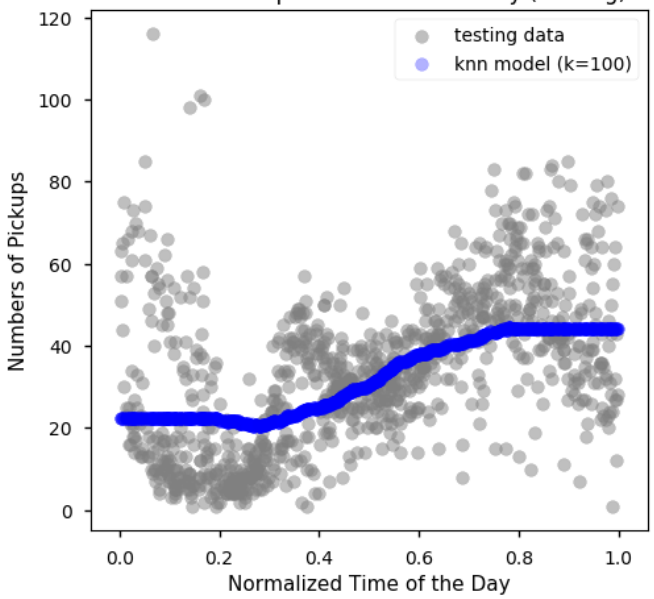




NYC Taxi Pickups vs. Time of the Day (Training)

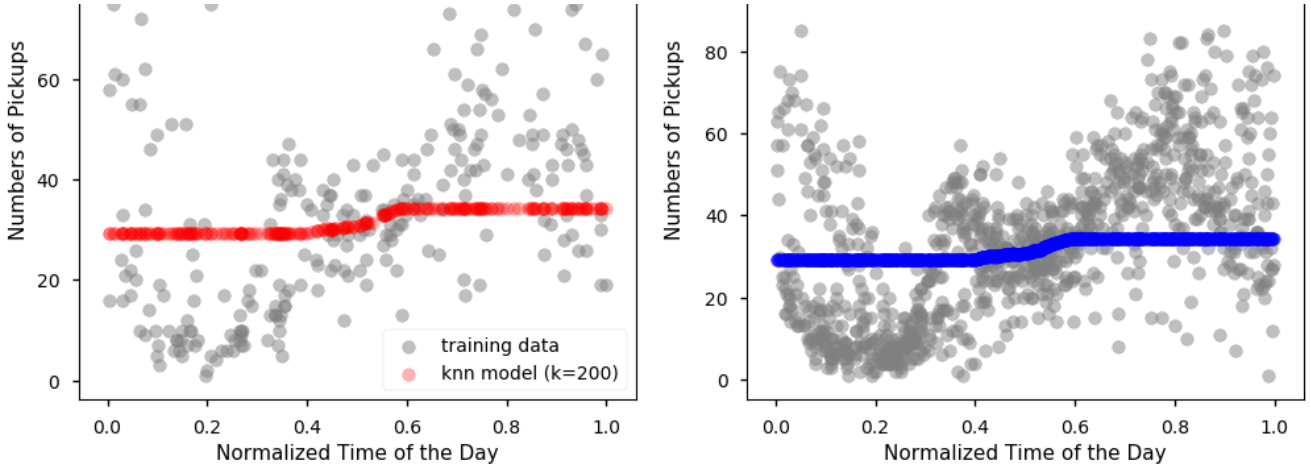
NYC Taxi Pickups vs. Time of the Day (Testing)





NYC Taxi Pickups vs. Time of the Day (Training)

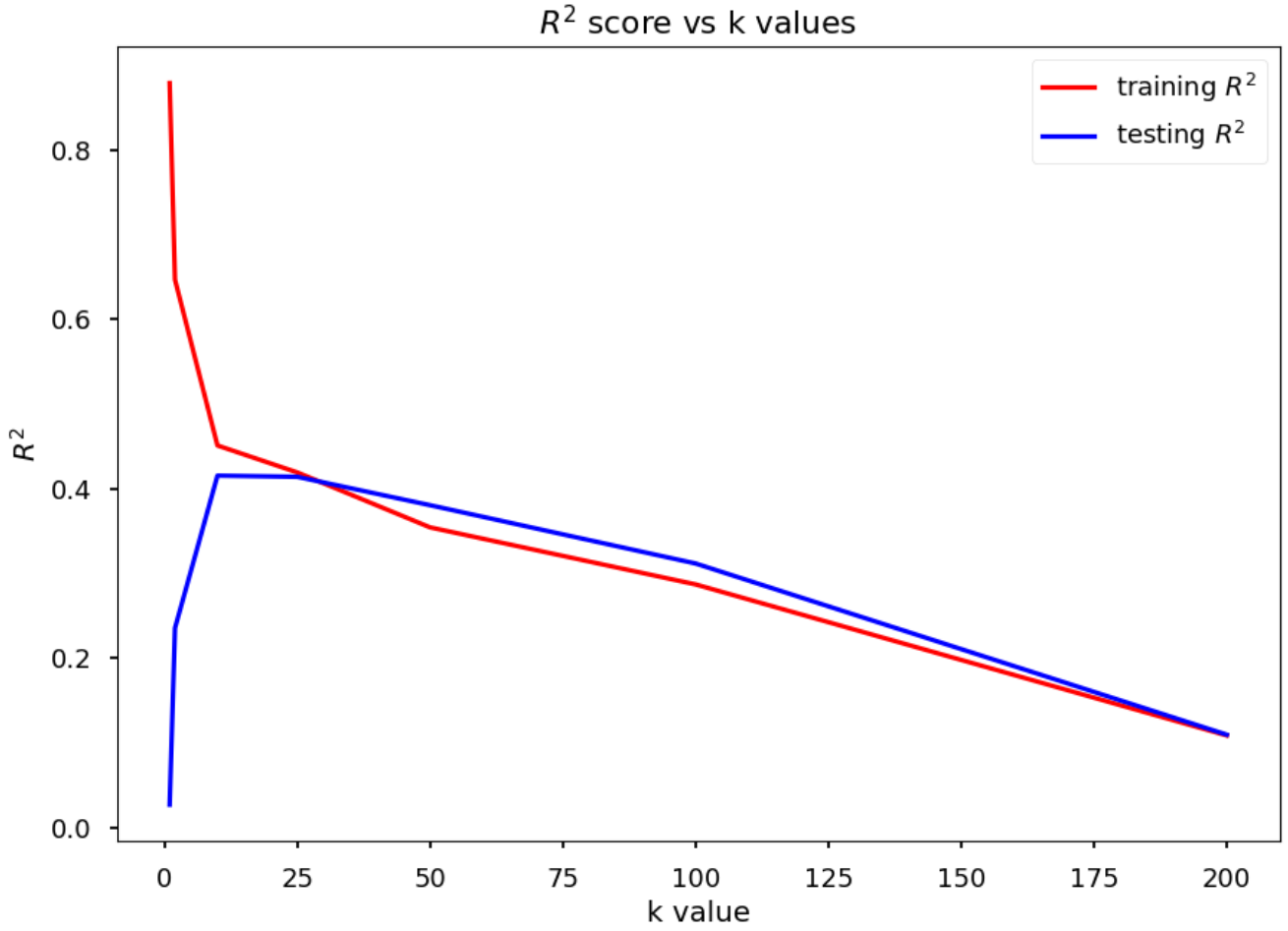
NYC Taxi Pickups vs. Time of the Day (Testing)



Just from visualization point of view, small k seems to have better prediction and large k has a more piecewise linear property

In [6]:
```
R_train = []
R_test = [];

for i in range(len(k_values)):
    R_train.append(knn_models[i].score(time_train, pickup_train))
    R_test.append(knn_models[i].score(time_test, pickup_test))
```

In [7]:
```
sns.reset_defaults()
sns.set_context('talk')
plt.plot(k_values, R_train, color='red', label='training $R^2$')
plt.plot(k_values, R_test, color='blue', label='testing $R^2$')
plt.xlabel('k value')
plt.ylabel('$R^2$')
plt.title('$R^2$ score vs k values')
plt.legend(loc='best')
plt.show()
```



None of the R^2 value is negative R^2 = 0 essentially mean the sum of squared error of the prediction model equals to the sum of squared error of a model whose prediction is the average of the entire set The training R^2 and the testing R^2 shows different trends for small values of k. For small k the training R^2 is high and decreasing while the testing R^2 score is low but increaseing After some moderate level of k (~25 in this case), the training and testing R^2 score show aalmost identical trend

Part (b): Simple Linear Regression

We next consider parametric approaches for regression, starting with simple linear regression, which assumes that the response variable has a linear relationship with the predictor. Do you see any advantages in using a parametric regression model over k-NN regression?

We suggest that you 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. Create a `OLS` class instance, use the `fit` method in the instance for fitting a linear regression model, and use the `predict` method to make predictions. To include an intercept term in the regression model, you will need to append a column of 1's to the array of predictors using the `sm.add_constant` method. The `fit` method returns a `results` instance. Use the `results.summary` method to obtain a summary of the regression fit, the `results.params` attribute to get the estimated regression parameters, and the `conf_int` method to compute confidence intervals for the estimated parameters. You may use the `r2_score` function to compute R^2.

Using the suggested built-in functions, answer the following questions:

- Fit a linear regression model to the training set, and evaluate its R^2 value on both the training and test sets (you may notice something peculiar about how they compare).
- How does the test R^2 score compare with the best test R^2 value obtained with k-NN regression in Part (a)?
- Compute confidence intervals:
  - Print the slope and intercept values for the fitted linear model. What does the sign of the slope convey about the data?
  - Compute the 95% confidence interval for the slope and intercept. Based on this information, do you consider the estimates of the model parameters to be reliable?
  - 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.
- Analyze residual plots:
  - Make a plot of the residuals $e = y - \hat{y}$ of the model on the training set as a function of the predictor variable x (i.e. time of day). Draw a horizontal line denoting the zero residual value on the Y-axis.
  - Using this residual plot, comment on whether the assumption of linearity is valid for this data.

In [8]:
```
T_train = sm.add_constant(time_train)
regres_model = OLS(pickup_train, T_train)
```

In [9]:
```
regres_result = regres_model.fit()
regres_result.summary()
```

Out[9]:

OLS Regression Results

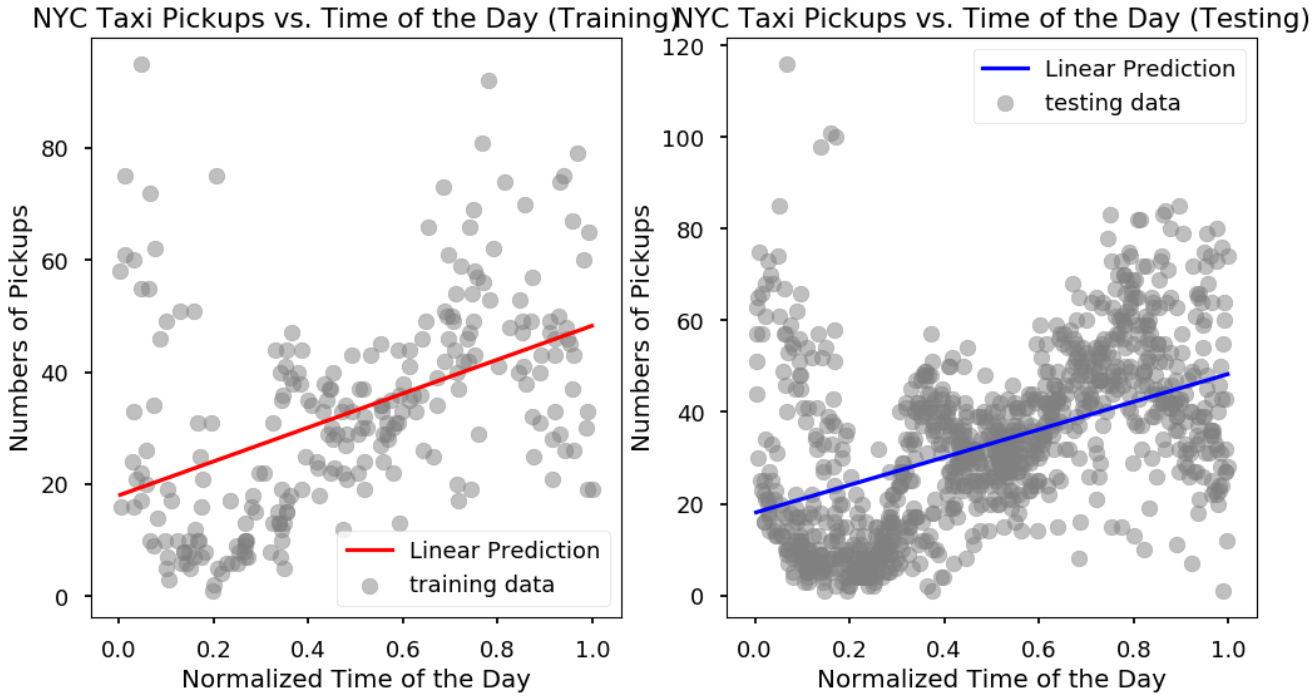
Dep. Variable:	y	**R-squared:**	0.207
Model:	OLS	**Adj. R-squared:**	0.204
Method:	Least Squares	**F-statistic:**	64.82
Date:	Mon, 25 Sep 2017	**Prob (F-statistic):**	3.43e-14
Time:	20:59:34	**Log-Likelihood:**	-1060.1
No. Observations:	250	**AIC:**	2124.
Df Residuals:	248	**BIC:**	2131.
Df Model:	1		
Covariance Type:	nonrobust		

	coef	**std err**	**t**	**P>|t|**	**[0.025**	**0.975]**
const	18.0264	2.121	8.501	0.000	13.850	22.203
x1	30.2890	3.762	8.051	0.000	22.879	37.699

Omnibus:	56.951	**Durbin-Watson:**	1.782
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	101.977
Skew:	1.202	**Prob(JB):**	7.18e-23
Kurtosis:	5.002	**Cond. No.**	4.42

In [10]:
```
T_test = sm.add_constant(time_test)
pickup_train_pred = regres_result.predict(T_train)
pickup_test_pred = regres_result.predict(T_test)
```

In [11]:
```
fig, ax = plt.subplots(1, 2, figsize=(12, 6))

ax[0].scatter(time_train, pickup_train, color='gray', alpha=0.5, label='training data')
ax[0].plot(time_train, pickup_train_pred, color='red', label='Linear Prediction')
ax[0].set_xlabel('Normalized Time of the Day')
ax[0].set_ylabel('Numbers of Pickups')
ax[0].set_title('NYC Taxi Pickups vs. Time of the Day (Training)')
ax[0].legend(loc='best')

ax[1].scatter(time_test, pickup_test, color='gray', alpha=0.5, label='testing data')
ax[1].plot(time_test, pickup_test_pred, color='blue', label='Linear Prediction')
ax[1].set_xlabel('Normalized Time of the Day')
ax[1].set_ylabel('Numbers of Pickups')
ax[1].set_title('NYC Taxi Pickups vs. Time of the Day (Testing)')
ax[1].legend(loc='best')

plt.show()
```



In [12]:
```
r2_linear_train = r2_score(pickup_train, pickup_train_pred)
r2_linear_test = r2_score(pickup_test, pickup_test_pred)
r2_linear_train, r2_linear_test
```

Out[12]:
```
(0.20721375209894033, 0.24771232994848624)
```

In [13]:
```
regres_result.params
```

Out[13]:
```
array([ 18.02638518,  30.28902299])
```

The positive slope indicates that as the normalized time of day increases (the later of a day), the number of pickup increases

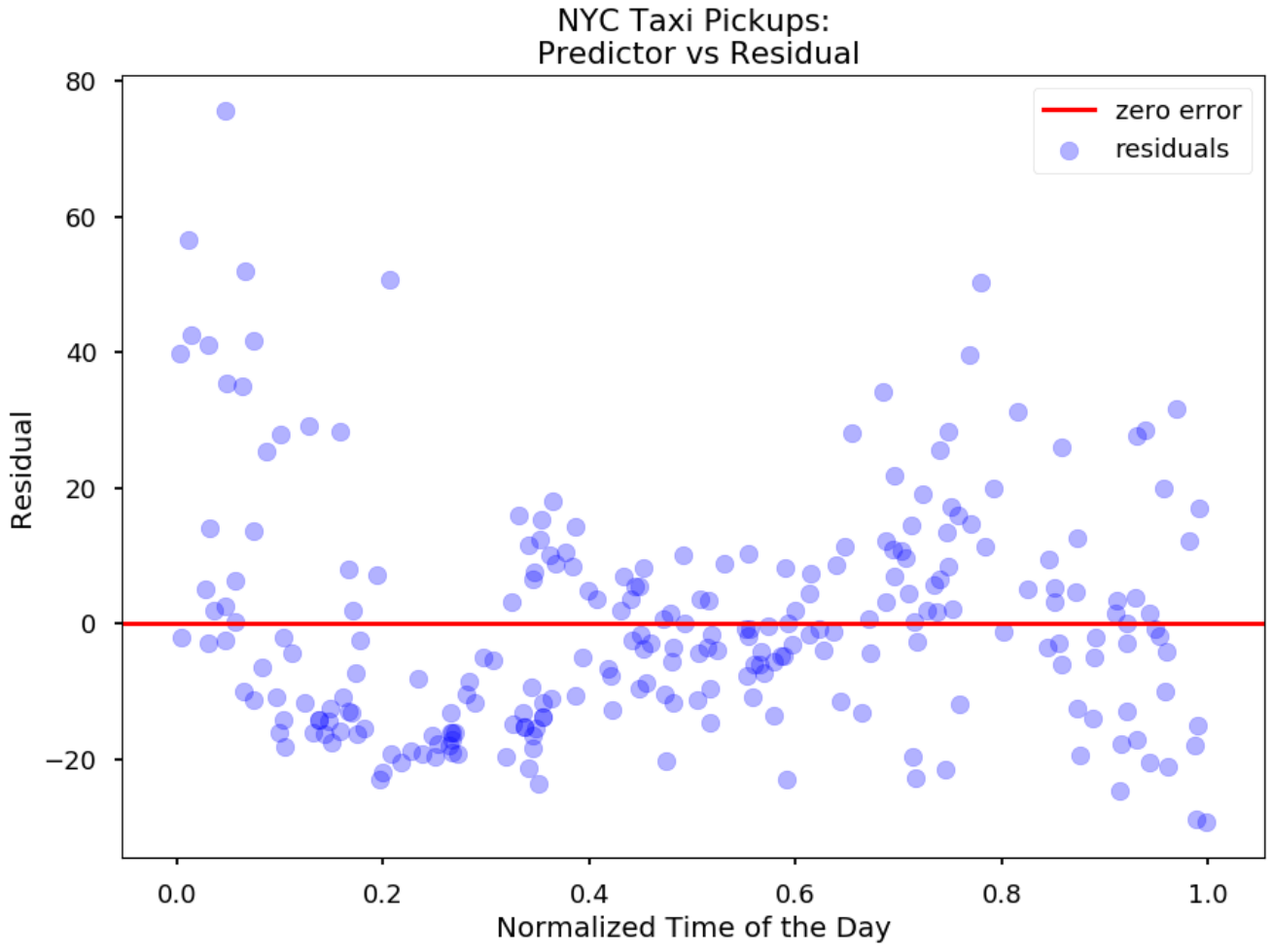
In [14]:
```
regres_result.conf_int()
```

Out[14]:
```
array([[ 13.84986472,  22.20290563],
       [ 22.879319  ,  37.69872697]])
```

The 95% interval gives a positive slope and a positive intercept, which in some sense prove this prediction model to be reliable. The 99% interval will be wider than the 95% one because it includes more low-probability coefficients into consideration

In [15]:
```
errors_linear = pickup_train - pickup_train_pred.reshape(len(pickup_train_pred), 1)
```

In [16]:
```
sns.reset_defaults()
sns.set_context('talk')
plt.scatter(time_train, errors_linear, color='blue', alpha=0.3, label='residuals')
plt.axhline(y=0, color='red', label='zero error')
plt.xlabel('Normalized Time of the Day')
plt.ylabel('Residual')
plt.title('NYC Taxi Pickups:\n Predictor vs Residual')
plt.legend(loc='best')
'''
fig, ax = plt.subplots(1, 1, figsize=(12, 6))

ax[0].scatter(time_train, errors_linear, color='blue', alpha=0.3, label='residuals')
ax[0].axhline(y=0, color='red', label='zero error')
ax[0].set_xlabel('Normalized Time of the Day')
ax[0].set_ylabel('Residual')
ax[0].set_title('NYC Taxi Pickups:\n Predictor vs Residual')
ax[0].legend(loc='best')

ax[1].hist(errors_linear, color='blue', alpha=0.3, label='residuals', bins=30, edgecolor='white', linewidth=2)
ax[1].axvline(x=0, color='red', label='zero error')
ax[1].set_xlabel('Residual')
ax[1].set_ylabel('Frequency')
ax[1].set_title('NYC Taxi Pickups:\n Predictor vs Residual')
ax[1].legend(loc='best')
'''
plt.show()
```



Based on the residual plot above, the assumption of linearity seems to be invalid as the residuals are not equally distributed from the zero-error line. During the earlier time of the day, most of the residuals are negative; Durinf the later time of the day, the residuals are mostly positive

Part (c): Polynomial Regression

We proceed to higher-order polynomial models for regression:

- By visual inspection, what polynomial degree do you think would provide the best fit for the data?

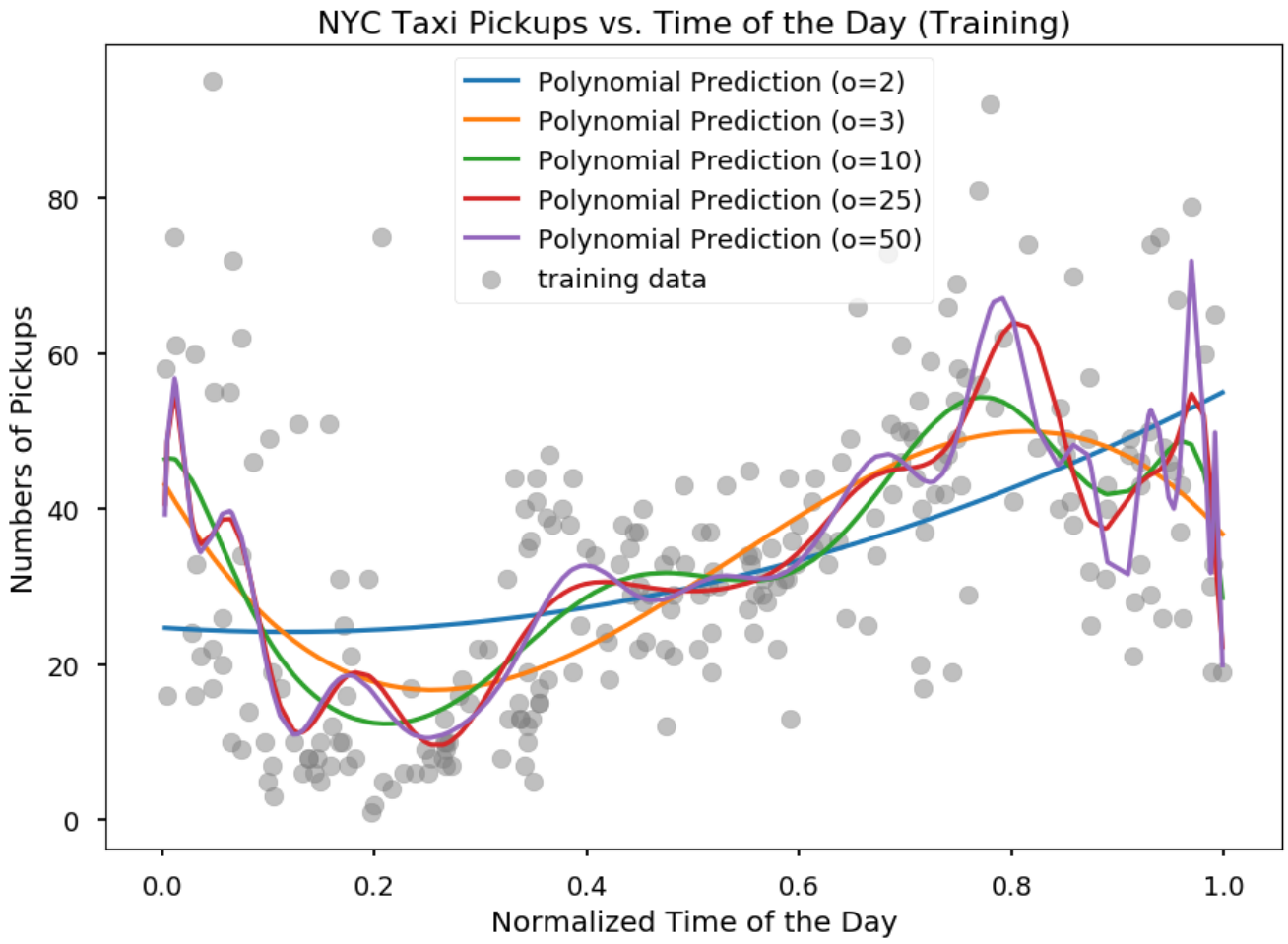
- At the start of this assignment, we had advised you to normalize the time predictor in the training and test sets to a value in [0,1], and noted that this would be helpful in fitting polynomial regression models. Had the time predictor not been normalized, what difficulties in implementing polynomial regression may have occurred?
- Fit polynomial regression models of degrees 2, 3, 10, 25 and 50 to the training set, and generate visualizations of the fitted models (in the same figure, plot the predicted value from all models as a function of time).
- Evaluate the R^2 value of the fitted models on both the training and test sets. Does a high training R^2 value necessarily indicate a high test R^2 value? How do the test R^2 values from the different polynomial models compare with the test R^2 from simple linear regression in Part (b), and the best test R^2 from k-NN regression in Part (a)?
- Generate residual plots for the different polynomial regression models (plot of residuals on training set vs. time). How does the increase in polynomial degree effect the residual plots?

Hint: You may use the `PolynomialFeatures` class to include polynomial terms in the regression model.

If the time (in min) was not normalized, higher order terms of larger time will be significantly larger than smaller time, making modeling extremely difficult

In [17]:
```
orders = [2, 3, 10, 25, 50]
poly_models = dict()
poly_results = dict()
T_train_poly = dict()
T_test_poly = dict()
pickup_train_poly_preds = dict()
pickup_test_poly_preds = dict()

for o in orders:
    key = str(o)
    poly = PolynomialFeatures(o)
    T_train_poly[key] = poly.fit_transform(time_train)
    T_test_poly[key] = poly.fit_transform(time_test)
    poly_models[key] = OLS(pickup_train, T_train_poly[key])
    poly_results[key] = poly_models[key].fit()
    pickup_train_poly_preds[key] = poly_results[key].predict(T_train_poly[key])
    pickup_test_poly_preds[key] = poly_results[key].predict(T_test_poly[key])
```

In [18]:
```
sns.reset_defaults()
sns.set_context('talk')

plt.scatter(time_train, pickup_train, color='gray', alpha=0.5, label='training data')
for o in orders:
    plt.plot(time_train, pickup_train_poly_preds[str(o)], label='Polynomial Prediction (o=%d)' % o)
plt.xlabel('Normalized Time of the Day')
plt.ylabel('Numbers of Pickups')
plt.title('NYC Taxi Pickups vs. Time of the Day (Training)')
plt.legend(loc='best')
plt.show()
```

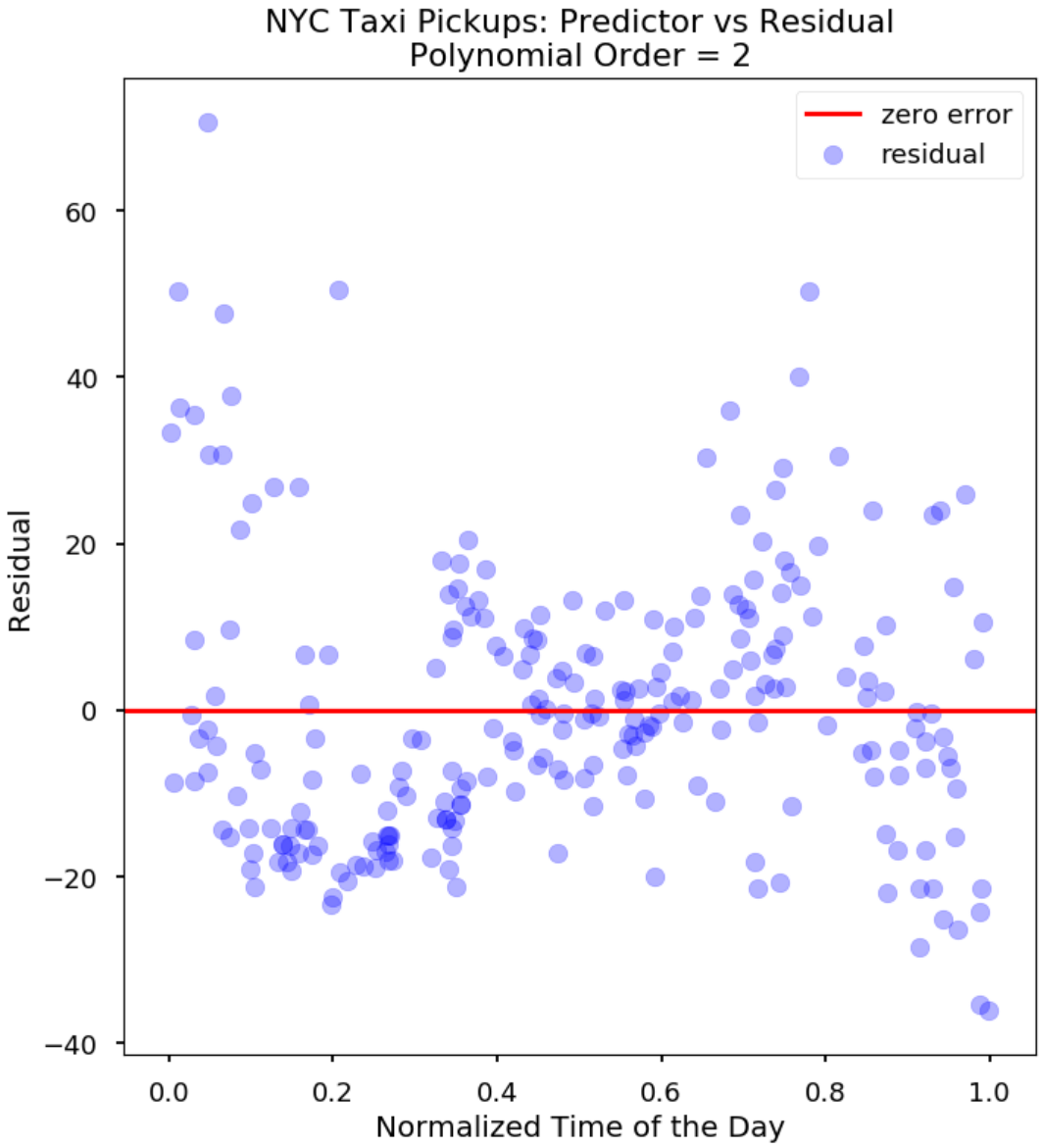


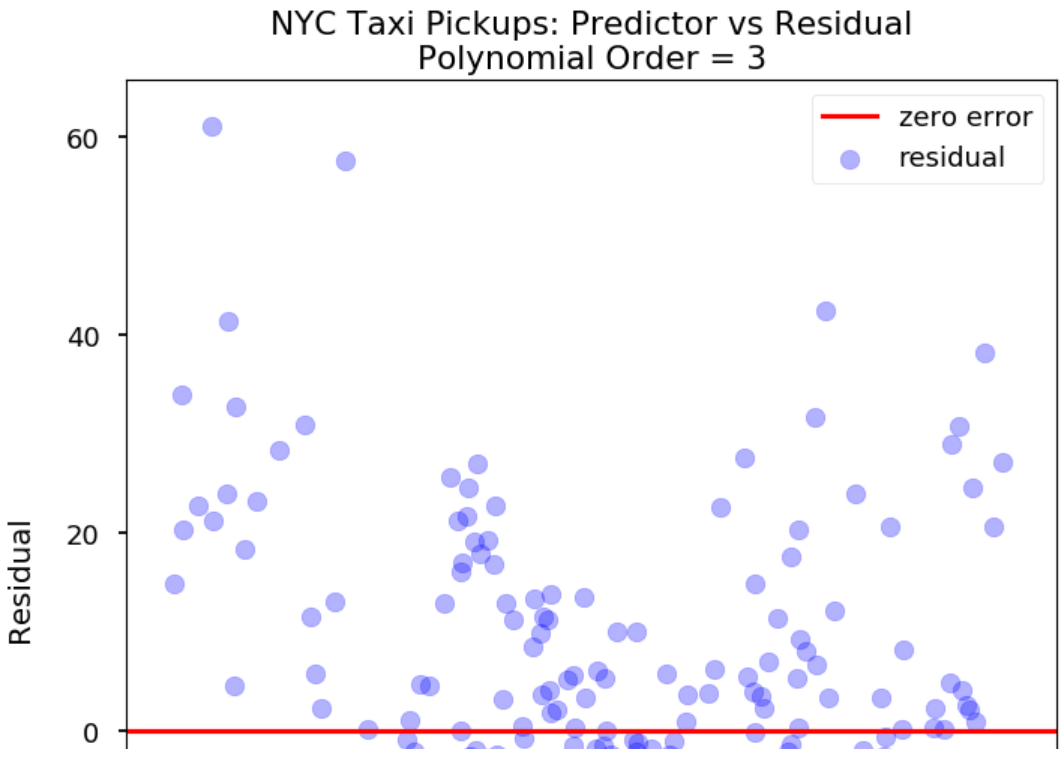
In [19]:
```
r2_poly_train = []
r2_poly_test = []

for o in orders:
    key = str(o)
    r2_poly_train.append(r2_score(pickup_train, pickup_train_poly_preds[key]))
    r2_poly_test.append(r2_score(pickup_test, pickup_test_poly_preds[key]))
```

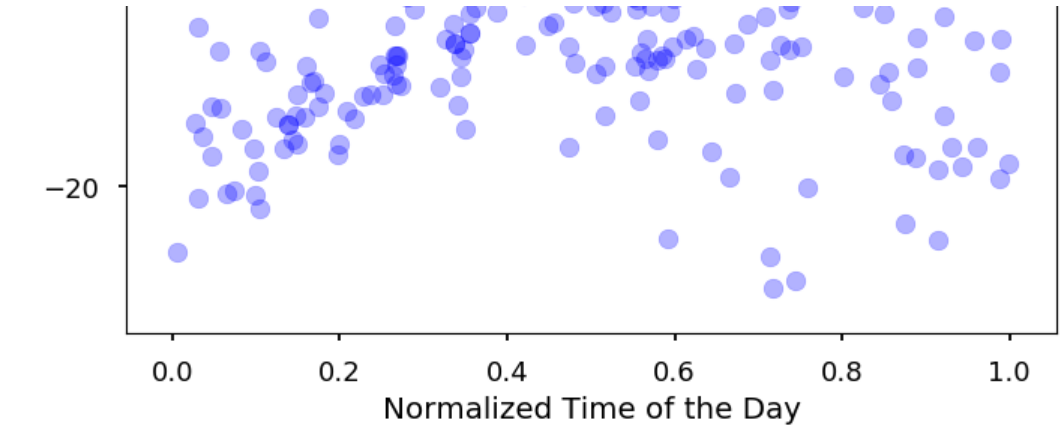
In [20]:
```
sns.reset_defaults()
sns.set_context('talk')
plt.plot(orders, r2_poly_train, color='red', label='training $R^2$')
plt.plot(orders, r2_poly_test, color='blue', label='testing $R^2$')
plt.xlabel('Polynomial Orders')
plt.ylabel('$R^2$')
plt.title('$R^2$ score vs Polynomial orders')
plt.legend(loc='best')
plt.show()
```

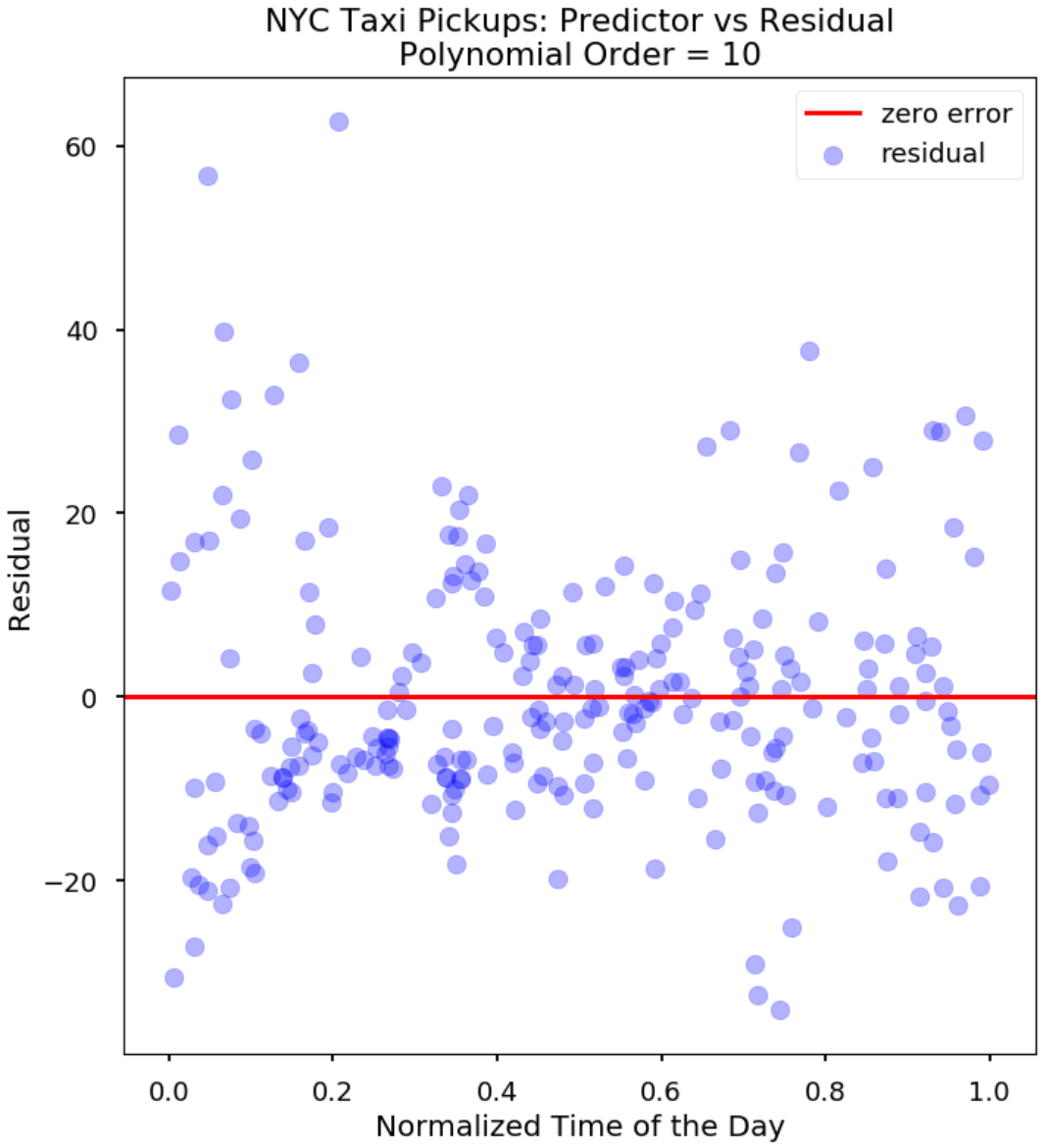
Polynomial fitting with order = 1 is essentially linear fitting, from the graph above and the R^2 score is always better for higher order fitting (up to 50 was tested). The polynomial fitting and the kNN method have very similar R^2 in the testing dataset, indicating that both models have a optimized fitting parameters (k=25 in the kNN and order=10 in the polynomial)

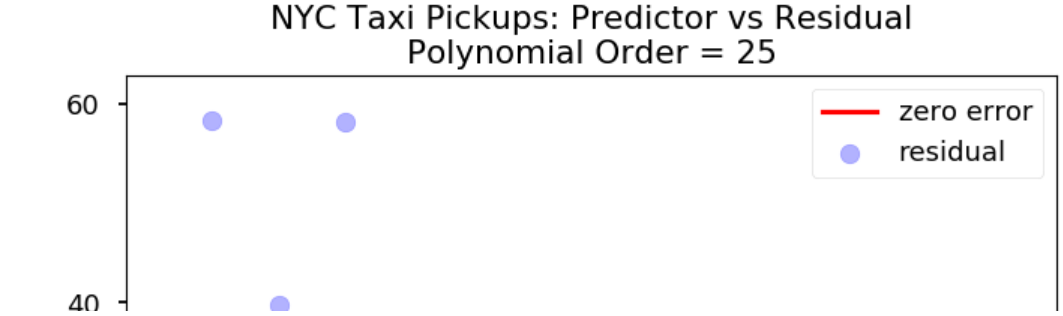
In [21]:
```
fig, ax = plt.subplots(len(orders), 1, figsize=(8, 10*len(orders)))
for i in range(len(orders)):
    key = str(orders[i])
    errors_poly = pickup_train - pickup_train_poly_preds[key].reshape(len(pickup_train_poly_preds[key]), 1)
    ax[i].scatter(time_train, errors_poly, color='blue', alpha=0.3, label='residual')
    ax[i].axhline(y=0, color='red', label='zero error')
    ax[i].set_xlabel('Normalized Time of the Day')
    ax[i].set_ylabel('Residual')
    ax[i].set_title('NYC Taxi Pickups: Predictor vs Residual\nPolynomial Order = %d'% orders[i])
    ax[i].legend(loc='best')

plt.show()
```

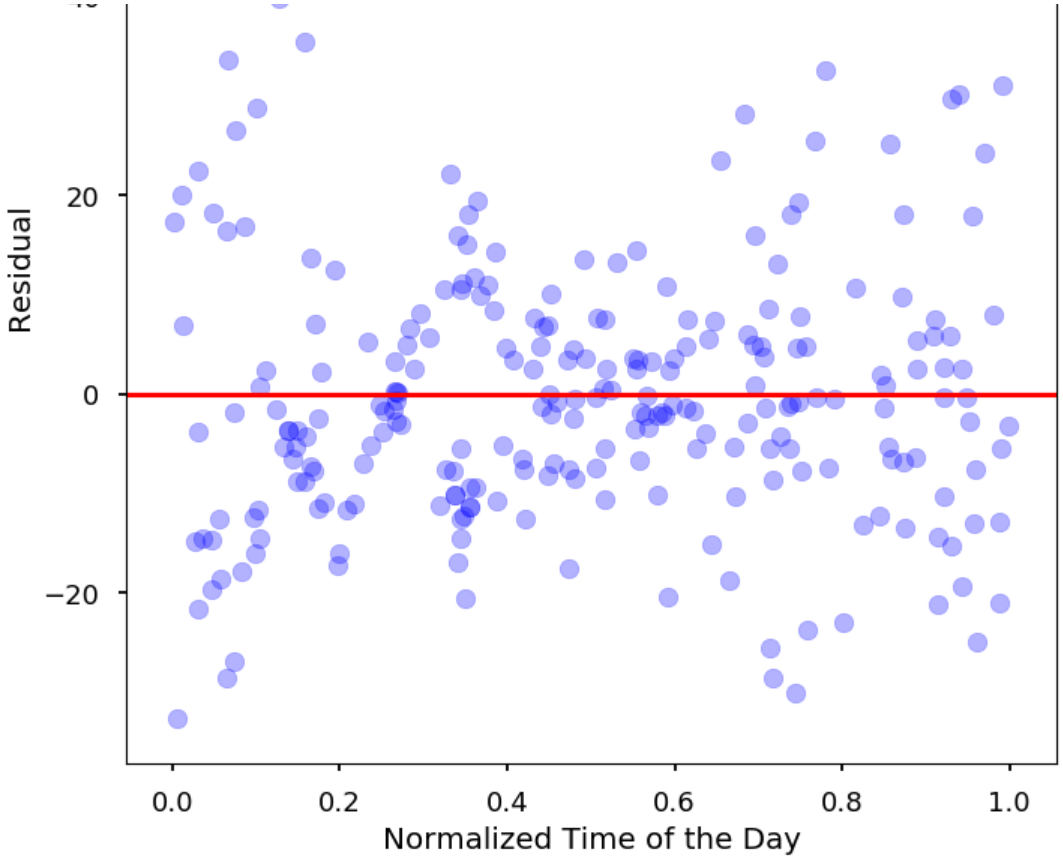

NYC Taxi Pickups: Predictor vs Residual
Polynomial Order = 2
zero error
residual
Residual
60
40
20
0
−20
−40
0.0
0.2
0.4
0.6
0.8
1.0
Normalized Time of the Day


NYC Taxi Pickups: Predictor vs Residual
Polynomial Order = 3
zero error
residual
Residual
60
40
20
0

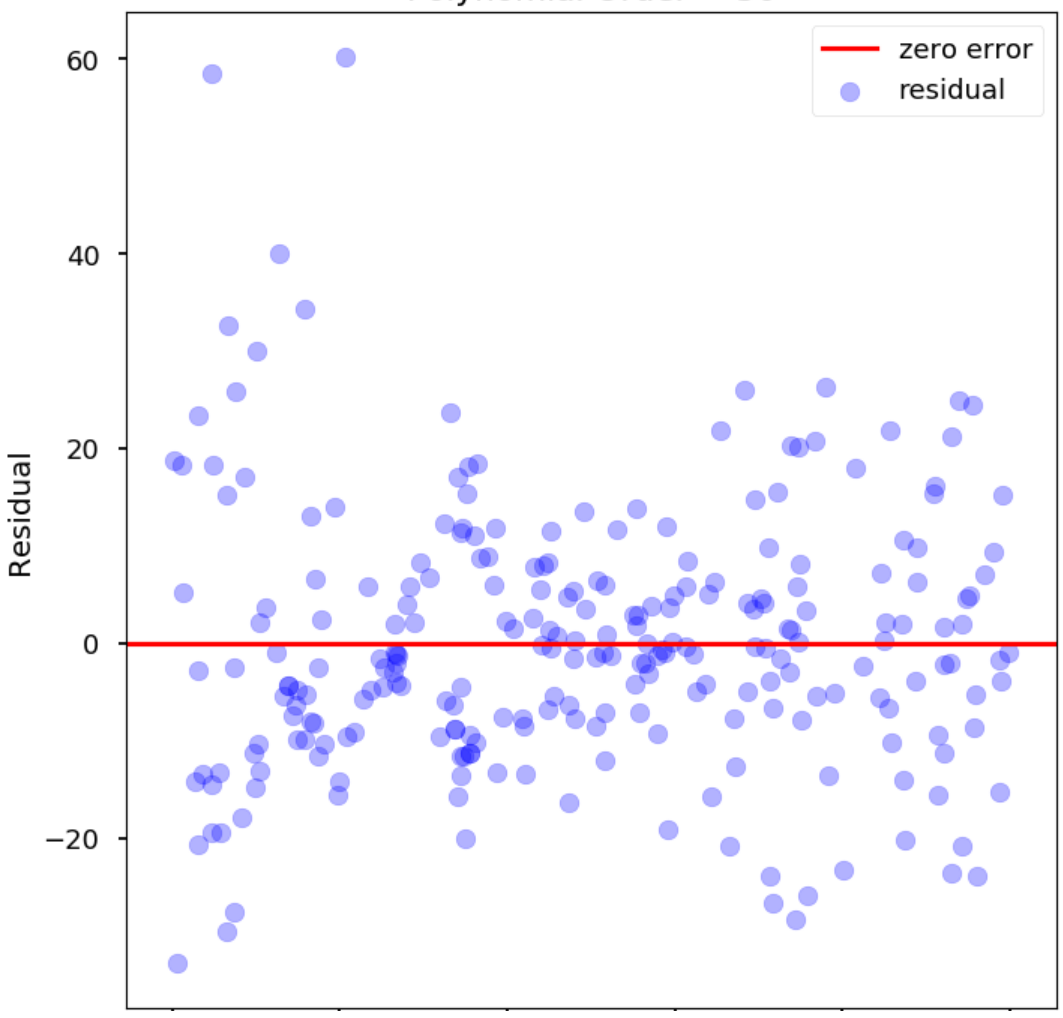

−20
0.0
0.2
0.4
0.6
0.8
1.0
Normalized Time of the Day

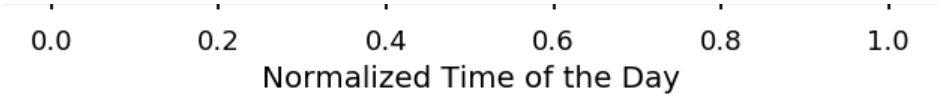

NYC Taxi Pickups: Predictor vs Residual
Polynomial Order = 10
zero error
residual
Residual
60
40
20
0
−20
0.0
0.2
0.4
0.6
0.8
1.0
Normalized Time of the Day


NYC Taxi Pickups: Predictor vs Residual
Polynomial Order = 25
zero error
residual
60
40



NYC Taxi Pickups: Predictor vs Residual
Polynomial Order = 50





Increasing order of the polynomial fitting makes the residual more equally distributed into both side of the the zero error line

Part (d): Summarize Results

In a brief paragraph (8 or fewer sentences), summarize which of the models seen above you would choose to predict the number of taxi cab pick-ups at any specific time of day. Be sure to explain your choice. Interpret the model you choose, including which predictors are significant and provide and interpret the CIs for their coefficients (if you choose a regression model). How well does you model predict the number of taxi cab pick-ups? How would you improve this model even further? Feel free to refer to visual(s) above or provide a new one to make your case.

In [22]:
```
poly_results['10'].summary()
```

Out[22]:

OLS Regression Results

Dep. Variable:	y	**R-squared:**	0.428
Model:	OLS	**Adj. R-squared:**	0.404
Method:	Least Squares	**F-statistic:**	17.90
Date:	Mon, 25 Sep 2017	**Prob (F-statistic):**	3.13e-24
Time:	20:59:36	**Log-Likelihood:**	-1019.3
No. Observations:	250	**AIC:**	2061.
Df Residuals:	239	**BIC:**	2099.
Df Model:	10		
Covariance Type:	nonrobust		

	coef	**std err**	**t**	**P>|t|**	**[0.025**	**0.975]**
const	46.0698	10.517	4.381	0.000	25.353	66.787
x1	149.3642	704.278	0.212	0.832	-1238.020	1536.749
x2	-1.172e+04	1.61e+04	-0.728	0.468	-4.34e+04	2e+04
x3	1.416e+05	1.71e+05	0.826	0.409	-1.96e+05	4.79e+05
x4	-8.966e+05	1e+06	-0.897	0.371	-2.87e+06	1.07e+06
x5	3.471e+06	3.49e+06	0.994	0.321	-3.41e+06	1.03e+07
x6	-8.463e+06	7.61e+06	-1.112	0.267	-2.35e+07	6.53e+06
x7	1.292e+07	1.04e+07	1.238	0.217	-7.64e+06	3.35e+07
x8	-1.193e+07	8.75e+06	-1.364	0.174	-2.92e+07	5.3e+06
x9	6.067e+06	4.09e+06	1.484	0.139	-1.99e+06	1.41e+07
x10	-1.303e+06	8.17e+05	-1.594	0.112	-2.91e+06	3.07e+05

Omnibus:	42.967	**Durbin-Watson:**	2.440
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	76.832
Skew:	0.916	**Prob(JB):**	2.07e-17
Kurtosis:	5.004	**Cond. No.**	2.34e+07

We should use the testing set to evaluate the model prediction. From the R^2 plots we can notice that the kNN model with k = 25 and the polynomial model with order = 10 has similar performance. However, given the summary of the 10th order polynomial fitting above, we can see that the confidence interval of the coefficients are so wide that indluce negative to positive values. This impose concerns as changing the sign of the coefficients will change the shape of the fitting curve drastically. Hence, in my opinion, I will choose the kNN model with k = 25 (approximately) to be the pickup prediction model. kNN model also relief the computational efforts. There might be some other way to improve this model. For examples, we can use weighted kNN, in which closer neighbors will be given more weights in computation.

APCOMP209a - Homework Question

Read Sections 1 and 2 of this paper (https://www.researchgate.net/profile/Roberto_Togneri/publication/45094554_Linear_Regression_for_Face_Recognition/links/09e4150d243bd8b987000000/Line Regression-for-Face-Recognition.pdf).

Briefly, we have a number of cleaned images of people's faces. The model leverages the concept that "patterns from a single-object class lie on a linear subspace" and the fact that linear regression can be thought of as an orthogonal projection of the response vector (Y) onto the subspace spanned by the columns of the predictor matrix (X).

Question 1

Consider a space in $\mathbb{R}^5$, with two subspaces $S_0 \subset \mathbb{R}^5$ and $S_1 \subset \mathbb{R}^5$. For simplicity, let us consider the case where the subspace S_0 is spanned by the vectors $v_{00} = [1,0,0,0,0]^T$, $v_{01} = [0,1,0,0,0]^T$ and the subspace S_1 is spanned by the vectors $v_{10} = [0,0,0,1,0]^T$ and $v_{11} = [0,0,0,0,1]^T$.

Now let us assume that we have a dataset that consists of labeled vectors in subspaces S_0 and S_1 respectively. Our task here is to use the data in the training dataset to classify an unknown vector into either S_0 or S_1.

Let us consider the case where we construct a predictive matrix $\mathbf{X}$ from the 'training' data for which we know the labels (note: is not meant to match S_0 and S_1 above).

```
dataset = np.array([
            [1,0,0,0,0],
            [1,1,0,0,0],
            [0,0,0,1,1],
            [0,0,0,1,0],
            ])
labels = np.array([0,0,1,1])
```

(Briefly notice that the training data probably doesn't consist of vectors that are orthonormal in spanning S_0 or S_1).

And we have an unknown vector, for which we want clasify as either a noisy example of a vector in either S_0 or S_1: `y_0 = np.array([2,10,1,0,0])`.

In [23]:
```
# starter code
dataset = np.array([
            [1,0,0,0,0],
            [1,1,0,0,0],
            [0,0,0,1,1],
            [0,0,0,1,0],
            ])
labels = np.array([0,0,1,1])
y_0 = np.array([2,10,1,0,0])
```

Question 1a

Recall the solution for the Least Squares problem and the 'hat' matrix $\mathbf{H}$. Consider the case when Y lies in the same subspace as the columns of $\mathbf{X}$. In this case, how accurate should the prediction of $\hat{Y}$ be?

Question 1b

Given the projection of Y onto the space spanned by the columns of the X matrix. We can analyse the significance of the different predictor vectors (i.e. the building blocks that are used to reconstrct the $\hat{Y}$ vector). Discuss how this may help one when analysing a classification decision by the model.

Question 1c

As discussed in the linked paper, we have data that has vectors that are classified into one subspace or another (our training dataset). We can use the projection interpretation of linear regression to make a classification decision of a new (unseen) vector into either S_0 or S_1. In other words, construct X matrices from the known vectors, and project the unknown vector onto the subspaces spanned by the various X matrices using the "hat" interpretation of linear regression. To do classification, we can calculate the minimum euclidean distance (L_2 norm) between the original vector and the projection. Use this method to classify `y_0` as belonging to either S_0 or S_1.

Feel free to run this same example on the face data presented in the paper - you will be doing this for next week's homework regardless

Question 1d (unrelated to the parts above)

Explicitly derive the relationship between the F-statistic (from the usual F test) and the R^2 measure in multiple linear regression (i.e. write the F-statistic of a multiple linear regression model in terms of the R^2 of the same model).

My Answer starts here:

1a:

If Y lies in the same subspace as the columns of $\mathbf{X}$, then the projection of Y into such subspace (which is essentially the prediction $\hat{Y}$) will be identical to Y (i.e. $\hat{Y} = Y$ and the model is noiseless)

1b:

Given the projection $\hat{Y}$, we can find the distance from Y to $\hat{Y}$ $d = ||Y - \hat{Y}||$. The decision is in favor of the minimum distance d_i for the class models X_i.

1c

In [24]:
```
# X matrix from S0
X0 = dataset[labels == 0].transpose()
XTX_0 = np.dot(X0.transpose(), X0)
XT_0 = X0.transpose()
beta_0 = np.dot(np.linalg.inv(XTX_0), XT_0)
H_0 = np.dot(X0, beta_0)
y_hat_0 = np.dot(H_0, y_0)
y_hat_0
```

Out[24]:
```
array([  2.,  10.,   0.,   0.,   0.])
```

In [25]:
```
# X matrix from S1
X1 = dataset[labels == 1].transpose()
XTX_1 = np.dot(X1.transpose(), X1)
XT_1 = X1.transpose()
beta_1 = np.dot(np.linalg.inv(XTX_1), XT_1)
H_1 = np.dot(X1, beta_1)
y_hat_1 = np.dot(H_1, y_0)
y_hat_1
```

Out[25]:
```
array([ 0.,  0.,  0.,  0.,  0.])
```

In [26]:
```
d_0 = np.linalg.norm(y_0 - y_hat_0) # distance from y_0 to y_hat in S0
d_1 = np.linalg.norm(y_0 - y_hat_1) # distance from y_0 to y_hat in S1
d_0, d_1
```

Out[26]: (1.0, 10.246950765959598)

Since $d_0 < d_1$, `y_0` can be classified to be belonging to S0

1d

$f = \frac{(TSS-RSS)/p}{RSS/(n-p-1)} = \frac{TSS-RSS}{RSS}\frac{n-p-1}{p} = (\frac{TSS}{RSS} - 1)\frac{n-p-1}{p}$

Since $R^2 = 1 - \frac{RSS}{TSS}$, we have $\frac{TSS}{RSS} = \frac{1}{1-R^2}$

Then $f = (\frac{1}{1-R^2} - 1)\frac{n-p-1}{p} = \frac{R^2}{1-R^2}\frac{n-p-1}{p}$