In [80]:
```
import numpy as np
import math
import pandas as pd
import matplotlib
import matplotlib.pyplot as plt
from sklearn.metrics import r2_score
from sklearn.decomposition import PCA
import statsmodels.api as sm
from statsmodels.api import OLS
from sklearn.preprocessing import PolynomialFeatures
from sklearn.model_selection import cross_val_score
from sklearn.linear_model import Ridge
from sklearn.linear_model import Lasso
from sklearn.linear_model import RidgeCV
from sklearn.linear_model import LassoCV
from sklearn.linear_model import LinearRegression

%matplotlib inline
pd.set_option('display.max_rows', 500)
```

In [81]:
```
# open the files
train_data = pd.read_csv('train_data_hw4.csv', sep=",", header=0)
test_data = pd.read_csv('test_data_hw4.csv', sep=",", header=0)
```

In [82]:
```
y_train = train_data['rentals'].values
X_train = train_data[['holiday', 'workingday', 'temp', 'atemp', 'humidity', 'windspeed', 'season_2',
                 'season_3', 'season_4', 'month_2', 'month_3', 'month_4', 'month_5', 'month_6',
                 'month_7', 'month_8', 'month_9', 'month_10', 'month_11', 'month_12', 'day_of_week_1',
                 'day_of_week_2', 'day_of_week_3', 'day_of_week_4', 'day_of_week_5', 'day_of_week_6', 'weather_2',
                 'weather_3']].values

y_test = test_data['rentals'].values
X_test = test_data[['holiday', 'workingday', 'temp', 'atemp', 'humidity', 'windspeed', 'season_2',
                 'season_3', 'season_4', 'month_2', 'month_3', 'month_4', 'month_5', 'month_6',
                 'month_7', 'month_8', 'month_9', 'month_10', 'month_11', 'month_12', 'day_of_week_1',
                 'day_of_week_2', 'day_of_week_3', 'day_of_week_4', 'day_of_week_5', 'day_of_week_6', 'weather_2',
                 'weather_3']].values
train_data = train_data.drop('Unnamed: 0', 1)
train_data = train_data.drop('Unnamed: 0.1', 1)
```

In [83]:
```
y_train = y_train.reshape(len(y_train), 1)
y_test = y_test.reshape(len(y_test), 1)

print(X_train.shape, y_train.shape, X_test.shape , y_test.shape)
```

```
(331, 28) (331, 1) (400, 28) (400, 1)
```

In [84]:
```
test_data.head(5)
```

Out[84]:

	Unnamed: 0	Unnamed: 0.1	holiday	workingday	temp	atemp	humidity	windspeed	rentals	season_2	season_3	season_4	month_2	month_3	month_
0	0	0	0.0	1.0	-1.341801	-1.363792	-0.500703	0.040945	3830.0	0	0	0	1	0	
1	1	1	0.0	1.0	-1.431146	-1.665877	0.132958	2.036025	2114.0	0	0	0	0	0	
2	2	2	0.0	1.0	1.695943	1.757749	-0.457103	-0.523392	915.0	1	0	0	0	0	
3	3	3	0.0	1.0	-0.805728	-0.759623	-0.997746	0.986696	4322.0	0	0	0	1	0	
4	4	4	0.0	0.0	0.981180	0.952190	0.441062	0.311061	6591.0	1	0	0	0	0	

In [85]:
```
alpha_values = [10**-5, 10**-4, 10**-3, 10**-2, 10**-1, 10**0, 10**1, 10**2, 10**3, 10**4, 10**5]
ridge_models = {}
lasso_models = {}

for alpha_val in alpha_values:

    # build the ridge and lasso regression model with specified lambda, ie, alpha
    ridge_reg = Ridge(alpha = alpha_val)
    lasso_reg = Lasso(alpha = alpha_val)

    # cross validate
    scores_r = cross_val_score(ridge_reg, X_train, y_train, cv=10)
    score_r = np.mean(scores_r)
    temp_dict_r = {alpha_val : score_r}
    ridge_models.update(temp_dict_r)

    scores_l = cross_val_score(lasso_reg, X_train, y_train, cv=10)
    score_l = np.mean(scores_l)
    temp_dict_l = {alpha_val : score_l}
    lasso_models.update(temp_dict_l)



best_lasso_alpha = max(lasso_models, key=lasso_models.get)
best_lasso_score =lasso_models.get(best_lasso_alpha)

best_ridge_alpha = max(ridge_models, key=ridge_models.get)
best_ridge_score =ridge_models.get(best_ridge_alpha)

print('The best ridge model has alpha=', best_ridge_alpha)
print('The best ridge model has score=', best_ridge_score)

print('The best lasso model has alpha=', best_lasso_alpha)
print('The best lasso model has score=', best_lasso_score)
```

```
 not converge. You might want to increase the number of iterations. Fitting data with very small alpha may cause precision pr
oblems.
  ConvergenceWarning)
/opt/anaconda3/lib/python3.6/site-packages/sklearn/linear_model/coordinate_descent.py:484: ConvergenceWarning: Objective did
 not converge. You might want to increase the number of iterations. Fitting data with very small alpha may cause precision pr
oblems.
  ConvergenceWarning)

/opt/anaconda3/lib/python3.6/site-packages/sklearn/linear_model/coordinate_descent.py:484: ConvergenceWarning: Objective did
 not converge. You might want to increase the number of iterations. Fitting data with very small alpha may cause precision pr
oblems.
  ConvergenceWarning)
/opt/anaconda3/lib/python3.6/site-packages/sklearn/linear_model/coordinate_descent.py:484: ConvergenceWarning: Objective did
 not converge. You might want to increase the number of iterations. Fitting data with very small alpha may cause precision pr
oblems.
  ConvergenceWarning)

The best ridge model has alpha= 10
The best ridge model has score= 0.436358716542
The best lasso model has alpha= 10
The best lasso model has score= 0.415939790448
```

In [86]:
```
# fit best validated Ridge and lasso models

ridge_reg = Ridge(alpha = best_ridge_alpha)
lasso_reg = Lasso(alpha = best_lasso_alpha)

#fit the model to training data
ridge_reg.fit(X_train, y_train)
lasso_reg.fit(X_train, y_train)

#save the beta coefficients
beta0_ridge = ridge_reg.intercept_
betas_ridge = ridge_reg.coef_

#save the beta coefficients
beta0_lasso = lasso_reg.intercept_
betas_lasso = lasso_reg.coef_

#make predictions everywhere
ypredict_ridge = ridge_reg.predict(X_train)
ypredict_lasso = lasso_reg.predict(X_train)

print('Ridge Beta0 is:', beta0_ridge)
print('Ridge Betas are:', betas_ridge)

print('Lasso Beta0 is:', beta0_lasso)
print('Lasso Betas are:', betas_lasso)
```

```
Ridge Beta0 is: [ 4001.41604352]
Ridge Betas are: [[-158.55971554  217.45335409  682.80825822  553.88994683 -567.72191933
  -266.41311936  393.17255806  172.1792805   761.78543763 -115.1218662
    88.89050681  369.53564813  133.37245386 -312.37487384 -529.54451686
   -89.29404068  676.88686869  503.42304275  159.09586506 -100.6582173
  -130.58113365 -125.49534264  124.31796857   64.47904074  126.17310553
   303.9886922    20.28051056 -676.14388831]]
Lasso Beta0 is: [ 3957.97842992]
Lasso Betas are: [   -0.           277.51867781   854.72446879   399.37258548  -555.84062186
  -254.36848284   543.88126524   113.79366043   898.94221619    -0.
    13.91707518   312.53427367     0.          -354.59953274  -496.15246234
    -0.           834.58471461   483.88131207    68.02103714    -0.
  -178.96330537  -129.13772753    17.59586395     0.            11.33513761
   309.9496593     -0.         -1058.29982989]
```

In [87]:
```
#--------  sample
# A function to select a random sample of size k from the training set
# Input:
#      x (n x d array of predictors in training data)
#      y (n x 1 array of response variable vals in training data)
#      k (size of sample)
# Return:
#      chosen sample of predictors and responses

def sample(x, y, k):
    n = x.shape[0] # No. of training points

    # Choose random indices of size 'k'
    subset_ind = np.random.choice(np.arange(n), k)

    # Get predictors and reponses with the indices
    x_subset = x[subset_ind, :]
    y_subset = y[subset_ind]

    return (x_subset, y_subset)
```

In [88]:
```
simpl_reg = LinearRegression()
sample_sizes = [100, 150, 200, 250, 300, 350, 400]
columns = ['sample_size', 'avg_training_ridge', 'SD_train_ridge', 'CI_train_ridge', 'avg_test_ridge', 'SD_test_ridge',
          'CI_test_ridge', 'avg_training_lasso', 'SD_train_lasso', 'CI_train_lasso', 'avg_test_lasso', 'SD_test_lasso',
          'CI_test_lasso', 'avg_training_simpl', 'SD_train_simpl', 'CI_train_simpl', 'avg_test_simpl', 'SD_test_simpl',
          'CI_test_simpl']
dicts = []
```

```python
# iterate through sample sizes
for sample_size in sample_sizes:

    scores_training_ridge = []
    scores_test_ridge = []
    scores_training_lasso = []
    scores_test_lasso = []
    scores_training_simpl = []
    scores_test_simpl = []

    # repeate 10 times for each fit
    for i in range(0, 10):
        sample_X, sample_y = sample(X_train, y_train, sample_size)
        ridge_reg.fit(sample_X, sample_y)
        lasso_reg.fit(sample_X, sample_y)
        simpl_reg.fit(sample_X, sample_y)

        pred_training_ridge = ridge_reg.predict(sample_X)
        pred_test_ridge = ridge_reg.predict(X_test)

        pred_training_lasso = lasso_reg.predict(sample_X)
        pred_test_lasso = lasso_reg.predict(X_test)

        pred_training_simpl =simpl_reg.predict(sample_X)
        pred_test_simpl = simpl_reg.predict(X_test)

        # r2 scores
        score_training_ridge = r2_score(sample_y, pred_training_ridge)
        scores_training_ridge.append(score_training_ridge)
        score_test_ridge = r2_score(y_test, pred_test_ridge)
        scores_test_ridge.append(score_test_ridge)

        score_training_lasso = r2_score(sample_y, pred_training_lasso)
        scores_training_lasso.append(score_training_lasso)
        score_test_ridge = r2_score(y_test, pred_test_lasso)
        scores_test_lasso.append(score_test_ridge)

        score_training_simpl = r2_score(sample_y, pred_training_simpl)
        scores_training_simpl.append(score_training_simpl)
        score_test_simpl = r2_score(y_test, pred_test_simpl)
        scores_test_simpl.append(score_test_simpl)

    # averages over the 10 trials
    avg_test_ridge = np.mean(scores_test_ridge)
    avg_test_lasso = np.mean(scores_test_lasso)
    avg_test_simpl = np.mean(scores_test_simpl)
    avg_training_ridge = np.mean(scores_training_ridge)
    avg_training_lasso = np.mean(scores_training_lasso)
    avg_training_simpl = np.mean(scores_training_simpl)

    # compute standard deviations of errors for training and test sets for each regression
    SD_train_ridge = np.std(scores_training_ridge)
    SD_train_lasso = np.std(scores_training_lasso)
    SD_train_simpl = np.std(scores_training_simpl)
    SD_test_ridge = np.std(scores_test_ridge)
    SD_test_lasso = np.std(scores_test_lasso)
    SD_test_simpl = np.std(scores_test_simpl)
    # compute confidance intervals
    CI_train_ridge = [avg_training_ridge-SD_train_ridge, avg_training_ridge+SD_train_ridge]
    CI_train_lasso = [avg_training_lasso-SD_train_lasso, avg_training_lasso+SD_train_lasso]
    CI_train_simpl = [avg_training_simpl-SD_train_simpl, avg_training_simpl+SD_train_simpl]
    CI_test_ridge = [avg_test_ridge-SD_train_ridge, avg_test_ridge+SD_test_ridge]
    CI_test_lasso = [avg_test_lasso-SD_train_lasso, avg_test_lasso+SD_test_lasso]
    CI_test_simpl = [avg_test_simpl-SD_train_simpl, avg_test_simpl+SD_test_simpl]



    # create a data structure to store our values
    temp_dict = {'sample_size': sample_size, 'avg_training_ridge' : avg_training_ridge, 'SD_train_ridge' : SD_train_ridge, 'CI_tr
                 'CI_test_simpl' : CI_test_simpl, 'CI_test_lasso' : CI_test_lasso, 'CI_test_ridge' : CI_test_ridge, 'CI_train_simp
                 'avg_training_simpl' : avg_training_simpl, 'SD_train_simpl' : SD_train_simpl, 'avg_test_simpl' : avg_test_simpl,
    dicts.append(temp_dict)

df = pd.DataFrame(dicts)
df = df[columns]
print(df)
```

```
   sample_size  avg_training_ridge  SD_train_ridge  \
0          100            0.584111        0.061805
1          150            0.592100        0.055601
2          200            0.590108        0.040498
3          250            0.591408        0.045712
4          300            0.587758        0.032593
5          350            0.600972        0.022318
6          400            0.576583        0.037033
```

```
                   CI_train_ridge  avg_test_ridge  SD_test_ridge  \
0  [0.522305720356, 0.645915672641]        0.228526       0.020185
1  [0.536498991647, 0.647700958723]        0.237000       0.020412
2  [0.549610243525, 0.630606648324]        0.240662       0.015697
3  [0.545695866803, 0.637120622897]        0.245105       0.016984
4   [0.555164295128, 0.62035076596]        0.239252       0.014442
5  [0.578653757389, 0.623289840309]        0.238852       0.015983
6  [0.539550397854, 0.613615565531]        0.246311       0.014794

                    CI_test_ridge  avg_training_lasso  SD_train_lasso  \
0  [0.166721249746, 0.248711377621]            0.658240        0.058756
1  [0.181399237581, 0.257412376422]            0.628095        0.056481
2  [0.200164215977, 0.256359660889]            0.615130        0.042261
3   [0.199392577533, 0.26208867026]            0.608617        0.044767
4  [0.206658745073, 0.253693502282]            0.598486        0.031241
5  [0.216534400969, 0.254835411852]            0.607589        0.022950
6  [0.209277917627, 0.261104559005]            0.581022        0.038925

                   CI_train_lasso  avg_test_lasso  SD_test_lasso  \
0  [0.599484737069, 0.716996208018]        0.168355       0.033415
1  [0.571613605175, 0.684575706423]        0.201860       0.042777
2  [0.572869636809, 0.657390923382]        0.231127       0.021269
3  [0.563850172043, 0.653384234213]        0.235626       0.018888
4  [0.567245027106, 0.629726456089]        0.233421       0.019176
5  [0.584639498744, 0.630539113617]        0.235696       0.021157
6  [0.542096389943, 0.619946657636]        0.249951       0.021360

                    CI_test_lasso  avg_training_simpl  SD_train_simpl  \
0   [0.10959922494, 0.201770134448]            0.685695        0.056708
1  [0.145379303288, 0.244637721039]            0.646911        0.055654
2  [0.188866347552, 0.252395990457]            0.633282        0.040676
```

2 [0.188866347552, 0.252395990457]   0.633282   0.040676

```
3   [0.19085901504, 0.254513914341]            0.625950        0.043840
4  [0.202180664517, 0.252596918123]            0.612080        0.031616
5  [0.212746318632, 0.256853357648]            0.622423        0.022351
6  [0.211025544128, 0.271310484174]            0.593761        0.040855

                   CI_train_simpl  avg_test_simpl  SD_test_simpl  \
0  [0.628987523738, 0.742403107772]        0.059502       0.061097
1    [0.59125696497, 0.70256543492]        0.113155       0.086066
2   [0.59260619523, 0.673957480372]        0.184191       0.040090
3  [0.582109731696, 0.669790429708]        0.189129       0.022281
4  [0.580464368149, 0.643696253099]        0.198303       0.022502
5   [0.600072288636, 0.64477364528]        0.198688       0.047105
6  [0.552906541443, 0.634615574755]        0.221268       0.036158

                      CI_test_simpl
0  [0.00279422681035, 0.120599109564]
1    [0.0575006405177, 0.19922060836]
2    [0.143515601024, 0.224281232824]
3    [0.145288359709, 0.211409420043]
4    [0.166686858086, 0.220804732959]
5    [0.176337162978, 0.245792547601]
6    [0.180413796837, 0.257426564103]
```

```
df
```

Out[89]:

	sample_size	avg_training_ridge	SD_train_ridge	CI_train_ridge	avg_test_ridge	SD_test_ridge	CI_test_ridge	avg_training_lasso	SD_train_lasso	CI_t
0	100	0.584111	0.061805	[0.522305720356, 0.645915672641]	0.228526	0.020185	[0.166721249746, 0.248711377621]	0.658240	0.058756	[0.599 0.716
1	150	0.592100	0.055601	[0.536498991647, 0.647700958723]	0.237000	0.020412	[0.181399237581, 0.257412376422]	0.628095	0.056481	[0.571 0.684
2	200	0.590108	0.040498	[0.549610243525, 0.630606648324]	0.240662	0.015697	[0.200164215977, 0.256359660889]	0.615130	0.042261	[0.572 0.657
3	250	0.591408	0.045712	[0.545695866803, 0.637120622897]	0.245105	0.016984	[0.199392577533, 0.26208867026]	0.608617	0.044767	[0.563 0.653
4	300	0.587758	0.032593	[0.555164295128, 0.62035076596]	0.239252	0.014442	[0.206658745073, 0.253693502282]	0.598486	0.031241	[0.567 0.629
5	350	0.600972	0.022318	[0.578653757389, 0.623289840309]	0.238852	0.015983	[0.216534400969, 0.254835411852]	0.607589	0.022950	[0.584 0.630
6	400	0.576583	0.037033	[0.539550397854, 0.613615565531]	0.246311	0.014794	[0.209277917627, 0.261104559005]	0.581022	0.038925	[0.542 0.619

In [90]:
```
f, (ax1, ax2, ax3) = plt.subplots(1, 3, sharey=True, figsize=(15,6))


ax1.errorbar(df.sample_size.values, df.avg_training_ridge.values, c='red', yerr=df.SD_train_ridge.values, fmt='o')
ax1.set_title("Ridge Training R2 Scores")
ax1.set_ylabel="R2"
ax1.set_xlabel="sample size"

ax2.errorbar(df.sample_size.values, df.avg_training_lasso.values, c='red', yerr=df.SD_train_lasso.values, fmt='o')
ax2.set_title("Lasso Training R2 Scores")
```

```
ax3.errorbar(df.sample_size.values, df.avg_training_simpl.values, c='red', yerr=df.SD_train_simpl.values, fmt='o')
ax3.set_title("Simple Regression Training R2 Scores")

f, (ax1, ax2, ax3) = plt.subplots(1, 3, sharey=True, figsize=(15,6) )
ax1.errorbar(df.sample_size.values, df.avg_test_ridge.values, c='red', yerr=df.SD_test_ridge.values, fmt='o')
ax1.set_title("Ridge Test R2 Scores")

ax2.errorbar(df.sample_size.values, df.avg_test_lasso.values, c='red', yerr=df.SD_test_lasso.values, fmt='o')
ax2.set_title("Lasso Test R2 Scores")

ax3.errorbar(df.sample_size.values, df.avg_test_simpl.values, c='red', yerr=df.SD_test_simpl.values, fmt='o')
ax3.set_title("Simple Regression Test R2 Scores")

plt.show()
```

```
*Analysis*

Our R2 on this part, compared to homework three is significantly better. Of our 40 predictors, 9 seem statistically significant
based upon p values obtained from OLS. A few others are very close, perhaps warranting inclusion. Our model still has high
dimensionality and high collinaearity, however.
```

In [91]:
```
train_data['temp^2'] = train_data['temp']**2
test_data['temp^2'] = test_data['temp']**2
train_data['temp^3'] = train_data['temp']**3
test_data['temp^3'] = test_data['temp']**3
train_data['temp^4'] = train_data['temp']**4
test_data['temp^4'] = test_data['temp']**4

train_data['atemp^2'] = (train_data['atemp'])**2
test_data['atemp^2'] = (test_data['atemp'])**2
train_data['atemp^3'] = (train_data['atemp'])**3
test_data['atemp^3'] = (test_data['atemp'])**3
train_data['atemp^4'] = (train_data['atemp'])**4
test_data['atemp^4'] = (test_data['atemp'])**4

train_data['humidity^2'] = (train_data['humidity'])**2
test_data['humidity^2'] = (test_data['humidity'])**2
train_data['humidity^3'] = (train_data['humidity'])**3
test_data['humidity^3'] = (test_data['humidity'])**3
train_data['humidity^4'] = (train_data['humidity'])**4
test_data['humidity^4'] = (test_data['humidity'])**4
```

```
train_data['windspeed^2'] = (train_data['windspeed'])**2
test_data['windspeed^2'] = (test_data['windspeed'])**2
train_data['windspeed^3'] = (train_data['windspeed'])**3
test_data['windspeed^3'] = (test_data['windspeed'])**3
train_data['windspeed^4'] = (train_data['windspeed'])**4
test_data['windspeed^4'] = (test_data['windspeed'])**4
```

In [92]:

```
# create an array of values for our regression
y_train = train_data['rentals'].values
X_train = train_data[['holiday', 'workingday', 'temp', 'atemp', 'humidity', 'windspeed', 'season_2',
                    'season_3', 'season_4', 'month_2', 'month_3', 'month_4', 'month_5', 'month_6',
                    'month_7', 'month_8', 'month_9', 'month_10', 'month_11', 'month_12', 'day_of_week_1',
                    'day_of_week_2', 'day_of_week_3', 'day_of_week_4', 'day_of_week_5', 'day_of_week_6', 'weather_2',
                    'weather_3', 'temp^2', 'temp^3', 'temp^4', 'atemp^2', 'atemp^3', 'atemp^4', 'humidity^2', 'humidity^3',
                    'humidity^4', 'windspeed^2', 'windspeed^3', 'windspeed^4']].values

y_test = test_data['rentals'].values
X_test = test_data[['holiday', 'workingday', 'temp', 'atemp', 'humidity', 'windspeed', 'season_2',
                    'season_3', 'season_4', 'month_2', 'month_3', 'month_4', 'month_5', 'month_6',
                    'month_7', 'month_8', 'month_9', 'month_10', 'month_11', 'month_12', 'day_of_week_1',
                    'day_of_week_2', 'day_of_week_3', 'day_of_week_4', 'day_of_week_5', 'day_of_week_6', 'weather_2',
                    'weather_3', 'temp^2', 'temp^3', 'temp^4', 'atemp^2', 'atemp^3', 'atemp^4', 'humidity^2', 'humidity^3',
                    'humidity^4', 'windspeed^2', 'windspeed^3', 'windspeed^4']].values

y_train = y_train.reshape(len(y_train), 1)
y_test = y_test.reshape(len(y_test), 1)

print(X_train.shape, y_train.shape, X_test.shape , y_test.shape)


train_data.head(5)
```

```
(331, 40) (331, 1) (400, 40) (400, 1)
```

Out[92]:

	holiday	workingday	temp	atemp	humidity	windspeed	rentals	season_2	season_3	season_4	month_2	month_3	month_4	month_5	month_6
0	0.0	1.0	0.623798	0.650106	0.920664	-0.928758	6073.0	1	0	0	0	0	0	1	0
1	0.0	1.0	-0.180310	-0.054759	0.696852	-0.213502	6606.0	0	0	1	0	0	0	0	0
2	0.0	1.0	0.802489	0.851495	-0.448383	0.803926	7363.0	1	0	0	0	0	0	0	1
3	0.0	0.0	-1.520492	-1.565182	-0.332113	-0.269099	2431.0	0	0	1	0	0	0	0	0
4	0.0	1.0	0.534453	0.348021	1.975789	-1.199027	1996.0	0	1	0	0	0	0	0	0

In [93]:

```
# fit model with the additional interaction terms, compute metrics
lm = LinearRegression(fit_intercept=True)
lm.fit(X_train, y_train)

y_pred = lm.predict(X_test)

print('The equation of the regression plane is: {} + {} * x'.format(lm.intercept_, lm.coef_))

train_MSE= np.mean((y_train - lm.predict(X_train))**2)
test_MSE= np.mean((y_test - lm.predict(X_test))**2)
print('The train MSE is {}, the test MSE is {}'.format(train_MSE, test_MSE))

train_R_sq = lm.score(X_train, y_train)
test_R_sq = lm.score(X_test, y_test)
print('The train R^2 is {}, the test R^2 is {}'.format(train_R_sq, test_R_sq))
```

```
The equation of the regression plane is: [ 5035.27125887] + [[ -189.7675006    351.27394405   771.48662326   897.2756023   -668.
91446319
   -446.50850455   766.43070371  1578.75436674  1523.2288234   -325.06857397
   -304.84911267  -418.02446177 -1037.20424547 -1456.18565573
  -1416.98816779 -1715.93889094 -1073.40080145  -925.87103867
   -825.53284158  -555.66756465   -93.32647706  -133.42791146   147.7312741
     30.59243395   209.93712392   471.0834343     59.01188326
  -1043.99967412 -1811.01797233     8.60775572   -45.191024    1175.50049569
   -303.93578541   -20.76855912   -53.67085686   -16.05763165   -24.8367323
    -34.16534669    44.8338792    -20.17694855]] * x
The train MSE is 1233551.6134555764, the test MSE is 3157061.4234926915
The train R^2 is 0.6696562402214016, the test R^2 is 0.27723843508615387
```

In [94]:

```
# statsmodel regression to easily obtain metrics

# create the X matrix by appending a column of ones to x_train
X = sm.add_constant(X_train)
X_test_sm = sm.add_constant(X_test)
# build the OLS model from the training data
smm = sm.OLS(y_train, X)

#save regression info in results_sm
results_sm = smm.fit()
```

```
print(results_sm.summary())
print('Parameters: ', results_sm.params)
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                      y   R-squared:                       0.670
Model:                            OLS   Adj. R-squared:                  0.625
Method:                 Least Squares   F-statistic:                     15.13
Date:                Thu, 12 Oct 2017   Prob (F-statistic):           7.98e-50
Time:                        02:47:40   Log-Likelihood:                -2790.9
No. Observations:                 331   AIC:                             5662.
Df Residuals:                     291   BIC:                             5814.
Df Model:                          39
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const       5035.2713    460.417     10.936      0.000    4129.101    5941.442
x1          -189.7675    365.157     -0.520      0.604    -908.451     528.916
x2           351.2739    150.615      2.332      0.020      54.841     647.707
x3           771.4866    760.117      1.015      0.311    -724.536    2267.510
x4           897.2756    713.172      1.258      0.209    -506.353    2300.904
x5          -668.9145    157.356     -4.251      0.000    -978.615    -359.214
x6          -446.5085    148.929     -2.998      0.003    -739.623    -153.394
x7           766.4307    454.546      1.686      0.093    -128.185    1661.046
x8          1578.7544    519.364      3.040      0.003     556.569    2600.940
x9          1523.2288    467.580      3.258      0.001     602.961    2443.496
x10         -325.0686    409.611     -0.794      0.428   -1131.245     481.108
x11         -304.8491    446.028     -0.683      0.495   -1182.700     573.002
x12         -418.0245    639.524     -0.654      0.514   -1676.703     840.654
x13        -1037.2042    677.186     -1.532      0.127   -2370.008     295.599
x14        -1456.1857    697.520     -2.088      0.038   -2829.010     -83.362
x15        -1416.9882    749.751     -1.890      0.060   -2892.610      58.634
x16        -1715.9389    743.240     -2.309      0.022   -3178.747    -253.131
x17        -1073.4008    660.859     -1.624      0.105   -2374.069     227.268
x18         -925.8710    617.522     -1.499      0.135   -2141.247     289.505
x19         -825.5328    591.138     -1.397      0.164   -1988.981     337.916
x20         -555.6676    479.543     -1.159      0.248   -1499.481     388.146
x21          -93.3265    156.015     -0.598      0.550    -400.387     213.734
x22         -133.4279    184.734     -0.722      0.471    -497.012     230.156
x23          147.7313    195.071      0.757      0.449    -236.197     531.660
x24           30.5924    187.547      0.163      0.871    -338.528     399.713
x25          209.9371    182.024      1.153      0.250    -148.313     568.187
x26          471.0834    246.557      1.911      0.057     -14.178     956.345
x27           59.0119    196.208      0.301      0.764    -327.155     445.179
x28        -1043.9997    546.051     -1.912      0.057   -2118.709      30.710
x29        -1811.0180    816.910     -2.217      0.027   -3418.820    -203.216
x30            8.6078    275.731      0.031      0.975    -534.071     551.287
x31          -45.1910    171.419     -0.264      0.792    -382.570     292.188
x32         1175.5005    788.864      1.490      0.137    -377.102    2728.103
x33         -303.9358    246.097     -1.235      0.218    -788.292     180.420
x34          -20.7686    147.605     -0.141      0.888    -311.276     269.739
x35          -53.6709    155.383     -0.345      0.730    -359.488     252.146
x36          -16.0576     44.892     -0.358      0.721    -104.412      72.297
x37          -24.8367     31.481     -0.789      0.431     -86.796      37.122
x38          -34.1653    126.952     -0.269      0.788    -284.026     215.695
x39           44.8339     65.459      0.685      0.494     -83.999     173.667
x40          -20.1769     30.327     -0.665      0.506     -79.864      39.510
edu/user/71142612/tree ========================================================
                           29.995   Durbin-Watson:                   1.959
```

In [95]:
```
train_data['month12_temp'] = train_data['month_12'] * train_data['temp']

train_data['workday_weather'] = np.where((train_data['workingday'] ==1) & (train_data['weather_2'] ==0) &
                                     (train_data['weather_3']==0), 1, 0)

test_data['month12_temp'] = test_data['month_12'] * test_data['temp']

test_data['workday_weather'] = np.where((test_data['workingday'] ==1) & (test_data['weather_2'] ==0) &
                                     (test_data['weather_3']==0), 1, 0)
```

In [96]:
```
pd.set_option('display.max_columns', None)
train_data.head()
```

Out[96]:

	holiday	workingday	temp	atemp	humidity	windspeed	rentals	season_2	season_3	season_4	month_2	month_3	month_4	month_5	month_6
0	0.0	1.0	0.623798	0.650106	0.920664	-0.928758	6073.0	1	0	0	0	0	0	1	0
1	0.0	1.0	-0.180310	-0.054759	0.696852	-0.213502	6606.0	0	0	1	0	0	0	0	0
2	0.0	1.0	0.802489	0.851495	-0.448383	0.803926	7363.0	1	0	0	0	0	0	0	1
3	0.0	0.0	-1.520492	-1.565182	-0.332113	-0.269099	2431.0	0	0	1	0	0	0	0	0
4	0.0	1.0	0.534453	0.348021	1.975789	-1.199027	1996.0	0	1	0	0	0	0	0	0

In [97]:
```
# create an array of values for our regression
y_train = train_data['rentals'].values
X_train = train_data[['holiday', 'workingday', 'temp', 'atemp', 'humidity', 'windspeed', 'season_2',
                   'season_3', 'season_4', 'month_2', 'month_3', 'month_4', 'month_5', 'month_6',
                   'month_7', 'month_8', 'month_9', 'month_10', 'month_11', 'month_12', 'day_of_week_1',
                   'day_of_week_2', 'day_of_week_3', 'day_of_week_4', 'day_of_week_5', 'day_of_week_6', 'weather_2',
                   'weather_3', 'temp^2', 'temp^3', 'temp^4', 'atemp^2', 'atemp^3', 'atemp^4', 'humidity^2', 'humidity^3',
                   'humidity^4', 'windspeed^2', 'windspeed^3', 'windspeed^4', 'workday_weather', 'month12_temp']].values

y_test = test_data['rentals'].values
X_test = test_data[['holiday', 'workingday', 'temp', 'atemp', 'humidity', 'windspeed', 'season_2',
                   'season_3', 'season_4', 'month_2', 'month_3', 'month_4', 'month_5', 'month_6',
                   'month_7', 'month_8', 'month_9', 'month_10', 'month_11', 'month_12', 'day_of_week_1',
                   'day_of_week_2', 'day_of_week_3', 'day_of_week_4', 'day_of_week_5', 'day_of_week_6', 'weather_2',
                   'weather_3', 'temp^2', 'temp^3', 'temp^4', 'atemp^2', 'atemp^3', 'atemp^4', 'humidity^2', 'humidity^3',
                   'humidity^4', 'windspeed^2', 'windspeed^3', 'windspeed^4', 'workday_weather', 'month12_temp']].values

y_train = y_train.reshape(len(y_train), 1)
y_test = y_test.reshape(len(y_test), 1)

print(X_train.shape, y_train.shape, X_test.shape , y_test.shape)

train_data.head(5)
```

```
(331, 42) (331, 1) (400, 42) (400, 1)
```

In [98]:
```
# fit model with the additional interaction terms, compute metrics
lm = LinearRegression(fit_intercept=True)
lm.fit(X_train, y_train)

y_pred = lm.predict(X_test)

print('The equation of the regression plane is: {} + {} * x'.format(lm.intercept_, lm.coef_))

train_MSE= np.mean((y_train - lm.predict(X_train))**2)
test_MSE= np.mean((y_test - lm.predict(X_test))**2)
print('The train MSE is {}, the test MSE is {}'.format(train_MSE, test_MSE))

train_R_sq = lm.score(X_train, y_train)
test_R_sq = lm.score(X_test, y_test)
print('The train R^2 is {}, the test R^2 is {}'.format(train_R_sq, test_R_sq))
```

```
The equation of the regression plane is: [ 5002.87217487] + [[ -1.74548786e+02   2.51998375e+02   7.95078743e+02   8.81715282e+0
2
   -6.76351913e+02  -4.47079697e+02   7.63637875e+02   1.56427958e+03
    1.49316259e+03  -3.25739496e+02  -3.11735875e+02  -4.29582296e+02
   -1.02485943e+03  -1.46665414e+03  -1.42767388e+03  -1.72063755e+03
   -1.05590929e+03  -8.82233606e+02  -7.73613283e+02  -4.80922153e+02
   -1.07457193e+02  -1.48195150e+02   1.29095641e+02   1.21349048e+01
    1.91871387e+02   4.62448641e+02   1.81873517e+02  -9.23040593e+02
   -1.80185812e+03   1.66279577e+00  -4.47557316e+01   1.17067291e+03
   -2.98989072e+02  -2.10470354e+01  -5.65186328e+01  -1.50310861e+01
   -2.39876686e+01  -3.78573222e+01   4.45758104e+01  -1.93589160e+01
    1.65026491e+02   4.44106026e+01]] * x
The train MSE is 1232388.953387242, the test MSE is 3131990.9875292666
The train R^2 is 0.6699675993036875, the test R^2 is 0.2829779330240658
```

In [ ]:

```
# The additional dimensions are interactions between the one-hot variables. While this analysis can capture some
# complex relationships, we only have 331 data items to begin with, and as the combinations of different parameters
# grows, you can quickly run out of data! When the number of factors exceeds the number of data items, the regression is
# unspecified. The purpose of PCA is to reduce the number of factors into groups of the most influential ones.

# The first three PCA components accounted for 46%, 21% and 19% of the variation in our data.

# We decided to stick with the predictors without the interaction terms, because we did not find them to be statistically
# significant.
```

In [113]:

```
#create polynomial features matrix
polynomial_features = PolynomialFeatures(degree=1, interaction_only=True, include_bias=True)
poly = polynomial_features.fit_transform(X_train)


#Using too many features can create an overfitting problem, particularly if they are of high degree - the training
#process will fit to the noise! Here, the degree of the factors isn't high (limited to 1), but with enough factors,
#you could wind up with more factors than you have data items, in which case the model is not specified.


pca1 = PCA(n_components=1)
pca1.fit(X_train)
X_train_pca1 = pca1.transform(X_train)
X_test_pca1 = pca1.transform(X_test)
print('Explained variance ratio:', pca1.explained_variance_ratio_)

pca2 = PCA(n_components=2)
pca2.fit(X_train)
X_train_pca2 = pca2.transform(X_train)
X_test_pca2 = pca2.transform(X_test)
print('Explained variance ratio:', pca2.explained_variance_ratio_)

pca3 = PCA(n_components=3)
pca3.fit(X_train)
X_train_pca3 = pca3.transform(X_train)
X_test_pca3 = pca3.transform(X_test)
print('Explained variance ratio:', pca3.explained_variance_ratio_)
```

```
Explained variance ratio: [ 0.46123346]
[[ -4.56350851e-04  -1.30637936e-03  -2.43585426e-02  -2.62500820e-02
   -1.88305024e-02   5.68188099e-02  -1.97999683e-03  -5.74257761e-03
   -3.80104140e-03   6.12190924e-03   3.60177162e-03   9.51634682e-04
   -2.64119912e-03  -1.34425553e-03  -2.74732289e-03  -1.58700340e-03
   -2.14616037e-03  -1.70033278e-03  -9.03383335e-04  -2.42813544e-04
    5.58731456e-04   1.48630748e-03  -1.83413829e-03  -2.63235240e-03
    6.58721549e-04   1.48868776e-03   7.67228134e-04   5.60494234e-04
    1.16305876e-02  -6.04297300e-02   7.30587192e-02   1.87160516e-02
   -8.02179436e-02   1.29792502e-01   3.22434422e-02  -5.44238937e-02
    1.12441696e-01   1.27833825e-01   3.38050800e-01   9.02754649e-01]]
Explained variance ratio: [ 0.46123346  0.21350874]
Explained variance ratio: [ 0.46123346  0.21350874  0.19390969]
```

In [100]:

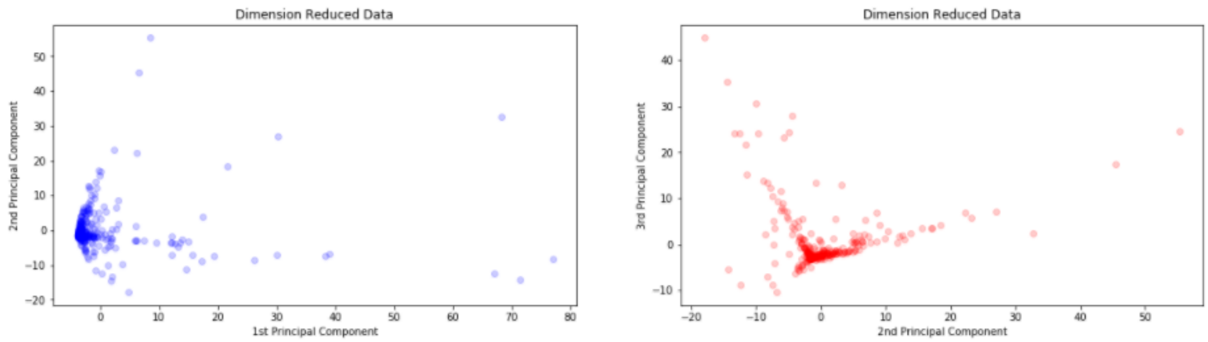
```
regression_model_pca1 = LinearRegression(fit_intercept=True)
regression_model_pca1.fit(X_train_pca1, y_train)
y_pred_1 = regression_model_pca1.predict(X_test_pca1)
score 1 = r2 score(y test, y pred 1)
print('PCA w 1 component Test R^2: {}'.format(score_1))

regression_model_pca2 = LinearRegression(fit_intercept=True)
regression_model_pca2.fit(X_train_pca2, y_train)

print('PCA w 2 component Test R^2: {}'.format(regression_model_pca2.score(X_test_pca2, y_test)))

regression_model_pca3 = LinearRegression(fit_intercept=True)
regression_model_pca3.fit(X_train_pca3, y_train)

print('PCA w 3 component Test R^2: {}'.format(regression_model_pca3.score(X_test_pca3, y_test)))
```

```
PCA w 1 component Test R^2: -0.058160712318159336
PCA w 2 component Test R^2: 0.012089582480906969
PCA w 3 component Test R^2: -0.020903459867501306
```

In [101]:

```
fig, ax = plt.subplots(1, 2, figsize=(20, 5))

ax[0].scatter(X_train_pca3[:, 0], X_train_pca3[:, 1], color='blue', alpha=0.2, label='train R^2')

ax[0].set_title('Dimension Reduced Data')
ax[0].set_xlabel('1st Principal Component')
ax[0].set_ylabel('2nd Principal Component')

ax[1].scatter(X_train_pca3[:, 1], X_train_pca3[:, 2], color='red', alpha=0.2, label='train R^2')

ax[1].set_title('Dimension Reduced Data')
ax[1].set_xlabel('2nd Principal Component')
```

```
ax[1].set_xlabel('2nd Principal Component')
ax[1].set_ylabel('3rd Principal Component')


plt.show()
```



Type *Markdown* and LaTeX: α^2

Part (i): Beyond Squared Error

In [102]:
```
#--------  rmsle
# A function for evaluating Root Mean Squared Logarithmic Error (RMSLE)
# of the linear regression model on a data set
# Input:
#      y_test (n x 1 array of response variable vals in testing data)
#      y_pred (n x 1 array of response variable vals in testing data)
# Return:
#      RMSLE (float)

def rmsle(y, y_pred):
    where_are_NaNs = np.isnan(np.log(y_pred+1))
    y_pred[where_are_NaNs] = 0
    rmsle_ = np.sqrt(np.mean(np.square(np.log(y+1) - np.log(y_pred+1))))
    print("RMSLE is ", rmsle_)

    return rmsle_
```

Use the above code to compute the training and test RMSLE for the polynomial regression model you fit in Part (g).

You are required to develop a strategy to fit a regression model by optimizing the RMSLE on the training set. Give a justification for your proposed approach. Does the model fitted using your approach yield lower train RMSLE than the model in Part (g)? How about the test RMSLE of the new model?

Note: We do not require you to implement a new regression solver for RMSLE. Instead, we ask you to think about ways to use existing built-in functions to fit a model that performs well on RMSLE. Your regression model may use the same polynomial terms used in Part (g).

In [109]:
```
print("Train:")
RMSLE = rmsle(y_train, lm.predict(X_train))
print("Test:")
RMSLE = rmsle(y_test, lm.predict(X_test))

# Our best guess is that we would use RMSLE to estimate a kind of lambda factor. The idea is that it is a logarithmic measure and
# can measure actual versus predicted values according to this relationship: log((pi+1)/(ai+1)), where pi is the
# predicted value and ai is the actual. It can be used to penalize under-estimates on a different basis than
# over-estimates, as might be the case when the suppliers of the bicycles being studied here are trying to decide how
# many bikes to stock - if people expect to face a stock-out because it is a busy day, you're not so worried about
# estimating demand, but if you underestimate demand on a lighter day, you could make a number of customers angry.
#
# It should be noted that the lambda we are suggesting above would a *multiplier* to *increase* estimated rentals
# givne a high RMSLE than the way we've used it elsewhere here, as a reducer. And it may be outside the regression
# equation entirely, but rather as a "post-processing" multiplier where you estimate rentals given parameters provided
# by the regression, AND THEN apply the multiplier suggested by the RMSLE.
```

```
Train:
RMSLE is  0.721457659701
Test:
RMSLE is  0.857618314536
```

In [104]:

```
# your code here


for x in range(0, 331):
    if (train_data.get_value(x, 'month_2') == 1 or train_data.get_value(x, 'month_3') == 1):
        train_data.set_value('season_2', x, 0)
        train_data.set_value('season_3', x, 0)
        train_data.set_value('season_4', x, 0)
    elif (train_data.get_value(x, 'month_4') == 1 or train_data.get_value(x, 'month_5') == 1 or train_data.get_value(x, 'month_6'
        train_data.set_value('season_2', x, 1)
        train_data.set_value('season_3', x, 0)
        train_data.set_value('season_4', x, 0)
    elif (train_data.get_value(x, 'month_7') == 1 or train_data.get_value(x, 'month_8') == 1 or train_data.get_value(x, 'month_9'
        train_data.set_value('season_2', x, 0)
        train_data.set_value('season_3', x, 1)
        train_data.set_value('season_4', x, 0)
    elif (train_data.get_value(x, 'month_10') == 1 or train_data.get_value(x, 'month_11') == 1 or train_data.get_value(x, 'month_
        train_data.set_value('season_2', x, 0)
        train_data.set_value('season_3', x, 0)
        train_data.set_value('season_4', x, 1)



y_train = train_data['rentals'].values
y_train = y_train[0:-3]
y_train = y_train.reshape(len(y_train), 1)
```

```
X_train = train_data[['holiday', 'workingday', 'temp', 'atemp', 'humidity', 'windspeed', 'season_2',
                 'season_3', 'season_4', 'month_2', 'month_3', 'month_4', 'month_5', 'month_6',
                 'month_7', 'month_8', 'month_9', 'month_10', 'month_11', 'month_12', 'day_of_week_1',
                 'day_of_week_2', 'day_of_week_3', 'day_of_week_4', 'day_of_week_5', 'day_of_week_6', 'weather_2',
                 'weather_3', 'temp^2', 'temp^3', 'temp^4', 'atemp^2', 'atemp^3', 'atemp^4', 'humidity^2', 'humidity^3',
                 'humidity^4', 'windspeed^2', 'windspeed^3', 'windspeed^4']].values
X_train = X_train[0:-3]
y_test = test_data['rentals'].values
X_test = test_data[['holiday', 'workingday', 'temp', 'atemp', 'humidity', 'windspeed', 'season_2',
                 'season_3', 'season_4', 'month_2', 'month_3', 'month_4', 'month_5', 'month_6',
                 'month_7', 'month_8', 'month_9', 'month_10', 'month_11', 'month_12', 'day_of_week_1',
                 'day_of_week_2', 'day_of_week_3', 'day_of_week_4', 'day_of_week_5', 'day_of_week_6', 'weather_2',
                 'weather_3', 'temp^2', 'temp^3', 'temp^4', 'atemp^2', 'atemp^3', 'atemp^4', 'humidity^2', 'humidity^3',
                 'humidity^4', 'windspeed^2', 'windspeed^3', 'windspeed^4']].values

y_test = y_test.reshape(len(y_test), 1)

print(X_train.shape, y_train.shape, X_test.shape , y_test.shape)

lm = LinearRegression(fit_intercept=True)
lm.fit(X_train, y_train)

y_pred = lm.predict(X_test)

print('The equation of the regression plane is: {} + {} * x'.format(lm.intercept_, lm.coef_))

train_MSE= np.mean((y_train - lm.predict(X_train))**2)
test_MSE= np.mean((y_test - lm.predict(X_test))**2)
print('The train MSE is {}, the test MSE is {}'.format(train_MSE, test_MSE))
```

```python
train_R_sq = lm.score(X_train, y_train)
test_R_sq = lm.score(X_test, y_test)
print('The train R^2 is {}, the test R^2 is {}'.format(train_R_sq, test_R_sq))
```

```
(331, 40) (331, 1) (400, 40) (400, 1)
The equation of the regression plane is: [ 5035.27125887] + [[ -189.7675006    351.27394405   771.48662326   897.2756023   -668.
91446319
   -446.50850455   766.43070371  1578.75436674  1523.2288234   -325.06857397
   -304.84911267  -418.02446177 -1037.20424547 -1456.18565573
  -1416.98816779 -1715.93889094 -1073.40080145  -925.87103867
   -825.53284158  -555.66756465   -93.32647706  -133.42791146   147.7312741
     30.59243395   209.93712392   471.0834343     59.01188326
  -1043.99967412 -1811.01797233     8.60775572   -45.191024    1175.50049569
   -303.93578541   -20.76855912   -53.67085686   -16.05763165   -24.8367323
    -34.16534669    44.8338792    -20.17694855]] * x
The train MSE is 1233551.6134555764, the test MSE is 3157061.4234926915
The train R^2 is 0.6696562402214016, the test R^2 is 0.27723843508615387
```

In [105]:

```python
'''

Analysis

1. I would start off with checking the season and month figures for inconsistencies.
2. I would also check for consistencies between the working day binary parameter and the binary day flags.
3. I would check for sanity in the deltas between atemp and temp

We visually inspected all three, and found a number of errors in 1, i.e., a season number would be ascribed to a
tuple with a month that didn't match. We were only able to find one error for #2, and could not find any errors for #3.
(We visually inspected deltas between temp and atemp and didn't find anything noteworthy.)

Our solution: assign season automatically depending on month for the entire training data set.

It turned out that our r-squared was actually REDUCED from 0.282 to 0.277! Cleaning the data did not impact our
results in any meaningful way. However, it may be that there is some more creative way to clean the data that we
simply did not think of, so we leave the door open to the possibility of obtaining more meaningful results.

'''
```