Data Analytics Group Project

1. Data Preparation

In [1]:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from matplotlib import colors
from IPython.display import Image
```

In [2]:

```
df = pd.read_excel('two-phase flow correlation data.xlsx')

def is_real_and_finite(x):
    if not np.isreal(x):
        return False
    elif not np.isfinite(x):
        return False
    else:
        return True

all_data = df[df.columns[1:]].values
numeric_map = df[df.columns[1:]].applymap(is_real_and_finite)
real_rows = numeric_map.all(axis = 1).copy().values
X = np.array(all_data[real_rows, 1:], dtype = 'float')
y = np.array(all_data[real_rows, 0], dtype = 'float')
y = y.reshape(-1, 1)
x_names = [str(x) for x in df.columns[2:]]
y_name = df.columns[1]

print('X matrix dimensions: {}'.format(X.shape))
print('y matrix dimensions: {}'.format(y.shape))
#print(X)
#print(x_names)
```

```
X matrix dimensions: (8057, 7)
y matrix dimensions: (8057, 1)
```

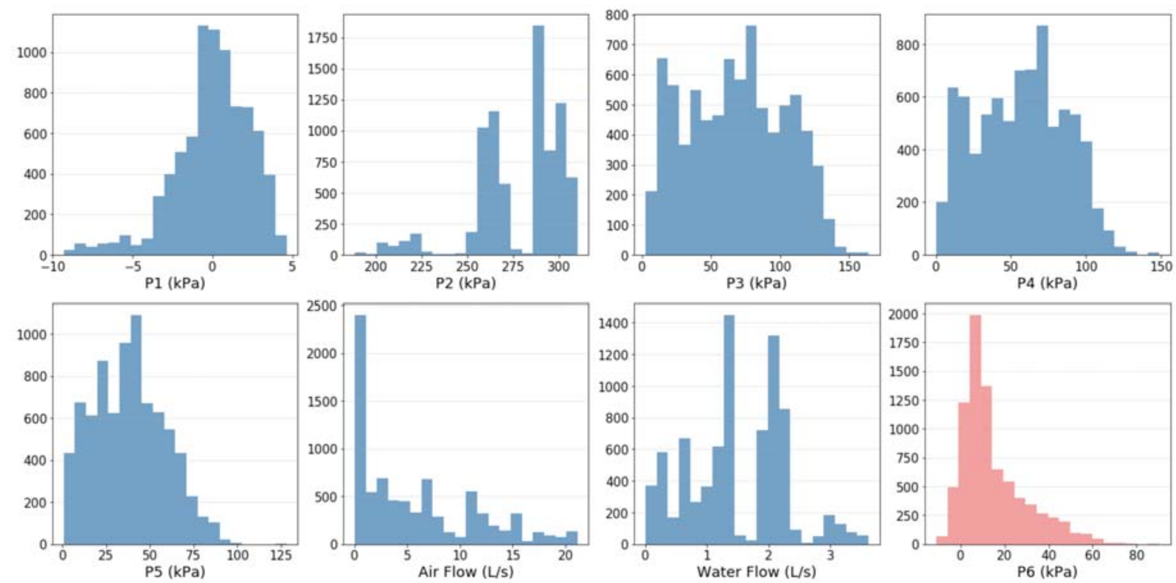
In [3]:

```
Image(filename='two-phase flow pressure drop.png')
```

Out[3]:

In [4]:

```
########################################## feature distribution ##############
fig, axes = plt.subplots(2, 4, figsize = (20, 10))
for i in range(2):
    for j in range(4):
        feature = i*4+j
        if feature <= 6:
            axes[i,j].grid(axis='y', linestyle='-', linewidth=1, alpha=0.3)
            axes[i,j].hist(X[:,feature], bins=20, facecolor='steelblue', alph
            axes[i,j].set_xlabel(x_names[feature], fontsize=18)
            axes[i,j].tick_params(axis='x', labelsize=15)
            axes[i,j].tick_params(axis='y', labelsize=15)
#distribution of target variable
axes[1,-1].grid(axis='y', linestyle='-', linewidth=1, alpha=0.3)
axes[1,-1].hist(y, bins=20, facecolor='lightcoral', alpha=0.75)
axes[1,-1].set_xlabel(y_name, fontsize=18)
plt.xticks(fontsize=15); plt.yticks(fontsize=15)
plt.tight_layout()
#plt.savefig('feature distribution.png', transparent=True, bbox_inches="tight
```



Data distributions

1. "normal": P1, P3, P4
2. "random": P2, Air Flow, Water Flow
3. "damped": P5, P6

2. Data management and exploratory analysis

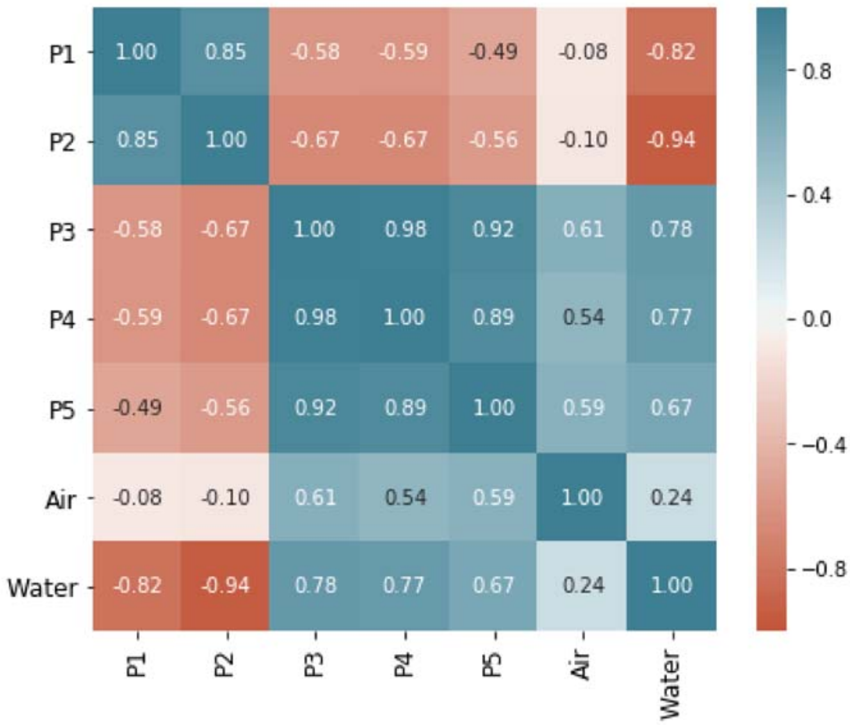
In [5]:

```
############################################## data preprocessing #############

################### feature scaling
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()
scaler.fit(X)
X_scaled = scaler.transform(X)
print('Total data size = {}'. format(X_scaled.shape))

################### data spliting
from sklearn.model_selection import train_test_split

# 70%-training,30%-test
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.
print('Train size = {}, Test size = {}'.format(y_train.shape, y_test.shape))
```

```
Total data size = (8057, 7)
Train size = (5639, 1), Test size = (2418, 1)
```

In [6]:

```
############################################### correlation matrix ############
import pandas as pd
import seaborn as sns

fig, ax = plt.subplots(figsize = (7,7))
corr = df[df.columns[2:]].corr()
ticks = ['P1', 'P2', 'P3', 'P4', 'P5', 'Air', 'Water']
ax = sns.heatmap(corr, vmin=-1, vmax=1, center=0, cmap=sns.diverging_palette(
ax.set_xticklabels(ticks,fontsize = 12);
ax.set_yticklabels(ticks, fontsize = 12);
#plt.savefig('correlation matrix.png', transparent=True, bbox_inches="tight")
```



Discussion

1. P3 and P4 looks highly correlated
2. P2 and water flow rate are anti-correlated

In [7]:

```
########################################################## 1-D analysis ########
from sklearn.linear_model import LinearRegression
#to reduce computing time use every 3rd training data point for development
X_train_3rd = X_train[::3]
y_train_3rd = y_train[::3]

###################### Piecewise Linear Function
print('Piecewise:\n')
from sklearn.linear_model import LinearRegression
#to reduce computing time use every 3rd data point

def piecewise_linear(x_train, x_test=None):
    if x_test is None:
        x_test = x_train
    N = len(x_test) #<- number of data points
    M = len(x_train) #<- number of features
    X = np.zeros((N,M))
    for i in range(N):
        for j in range(M):
            X[i,j] = max(0, x_test[i] - x_train[j])
    return X


for i in range(X_train_3rd.shape[1]):
    X_train_pl = piecewise_linear(X_train_3rd[:,i])
    X_train_pl[:, -1] += 1                       #makes column of 1s at the

    model = LinearRegression(fit_intercept = False) #create a linear regressi
    model.fit(X_train_pl, y_train_3rd)

    X_test_pl = piecewise_linear(X_train_3rd[:,i], X_test[:,i])
    yhat = model.predict(X_test_pl)

    r2_train = model.score(X_train_pl,y_train_3rd)
    r2_test = model.score(X_test_pl,y_test)
    print('For {} the Training r^2 = {}, Test r^2 = {}'.format(x_names[i], r2



############################### Radial Basis Function Kernel ################
print('\n\nRBF kernel:\n')

def rbf(x_train, x_test = None, gamma = 0.01):
    if x_test is None:
        x_test = x_train
    N = len(x_test)
    M = len(x_train)
    X = np.zeros((N, M))
    for i in range(N):
        for j in range(M):
            X[i, j] = np.exp(-gamma * (x_test[i] - x_train[j])**2)
    return X

sigmas = np.array([0.0005, 0.0006, 0.00075, 0.0009, 0.001, 0.005, 0.01])
gammas = 1. / 2 / sigmas**2
```

```
rows = X_train_3rd.shape[1] * len(gammas)

train_r2 = np.zeros((X_train_3rd.shape[1], len(gammas), 3)) #[feature (k), ga
test_r2 = np.zeros((X_train_3rd.shape[1], len(gammas), 3)) #[feature (k), gam

fig, axes = plt.subplots(7, 1, figsize = (10,20))

for k in range(X_train_3rd.shape[1]):
    for i, gamma in enumerate(gammas):

        X_train_rbf = rbf(X_train_3rd[:,k], x_test = None, gamma = gamma)

        model = LinearRegression()
        model.fit(X_train_rbf, y_train_3rd)

        X_test_rbf = rbf(X_train_3rd[:,k], X_test[:,k], gamma = gamma)
        yhat = model.predict(X_test_rbf)

        r2_train = model.score(X_train_rbf,y_train_3rd)
        r2_test = model.score(X_test_rbf,y_test)

        print('For {} the Training r^2 = {}, Test r^2 = {}'.format(x_names[k]

        train_r2[k,i,:] = [k, gamma, r2_train]
        test_r2[k,i,:] = [k, gamma, r2_test]

    axes[k].scatter(train_r2[k,:, 1], train_r2[k,:, 2], c = 'r')
    #axes[k].scatter(test_r2[k,:,1], test_r2[k,:,2], c = 'b')
    axes[k].set_title(x_names[k])
    axes[k].set_xlabel('gamma')
    axes[k].set_ylabel('R2')


fig.subplots_adjust(hspace = 0.8);
#plt.savefig('1-D regression.png', transparent=True, bbox_inches="tight")
```

```
862254997e+23
For P1 (kPa) the Training r^2 = 0.9294816049430153, Test r^2 = -5.130351
596568741e+22
For P1 (kPa) the Training r^2 = 0.9350830317334736, Test r^2 = -5.005376
361012304e+21
For P1 (kPa) the Training r^2 = 0.9088258475308635, Test r^2 = -14713254
4881587.6
For P1 (kPa) the Training r^2 = 0.6338293236114736, Test r^2 = -1.913233
8629139096e+16
For P2 (kPa) the Training r^2 = 0.9916519343221226, Test r^2 = -4.791256
1958282355e+19
For P2 (kPa) the Training r^2 = 0.9916601876666241, Test r^2 = -6.925027
223120209e+17
For P2 (kPa) the Training r^2 = 0.9926340571025064, Test r^2 = -92887900
6113781.8
For P2 (kPa) the Training r^2 = 0.9928755894910053, Test r^2 = -44232357
842595.0
For P2 (kPa) the Training r^2 = 0.9924529038629619, Test r^2 = -70691014
62145891.0
```

Discussion

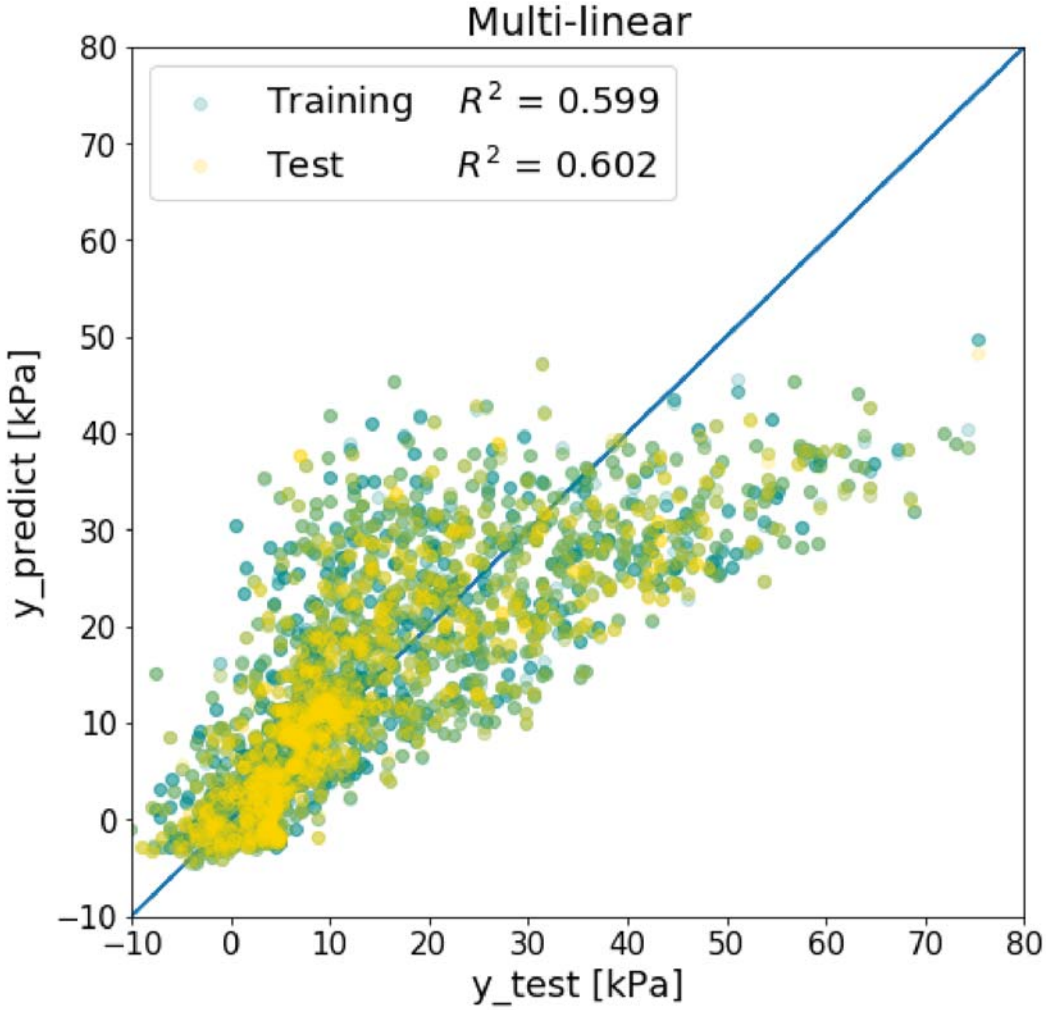
1. Piecewise linear model gives good r^2 on training set, but not for test set
2. RBF kernel model need further hyperparameter tuning to give better performence
3. The score for both non-parametric models are not good, indicating single feature is not enough for prediction

3. Baseline model

In [8]:

```
########################################## Baseline model 1 #################

######################### multi-linear regression
from sklearn.linear_model import LinearRegression

model = LinearRegression()
model.fit(X_train,y_train)
y_hat_test = model.predict(X_test)
y_hat_train = model.predict(X_train)

r2_train = model.score(X_train,y_train)
r2_test = model.score(X_test,y_test)

print('\n\nTraining r^2 = {}, Test r^2 = {}\n\n'.format(r2_train, r2_test))

fig, ax = plt.subplots(figsize = (8,8))
ax.plot(y_test, y_test, '-')
ax.plot(y_train, y_hat_train, 'o', label = 'Training    $R^2$ = 0.599', color
ax.plot(y_test, y_hat_test, 'o', label = 'Test          $R^2$ = 0.602', color
plt.xlim(-10, 80)
plt.ylim(-10, 80)
ax.set_xlabel('y_test [kPa]', fontsize = 18)
ax.set_ylabel('y_predict [kPa]', fontsize = 18)
ax.set_title('Multi-linear', fontsize=20)
plt.legend(loc = 2, fontsize = 18)
plt.xticks(fontsize=15); plt.yticks(fontsize=15);

#plt.savefig('parity_multi-linear.png', transparent=True, bbox_inches="tight'
```

```
Training r^2 = 0.6015141616284019, Test r^2 = 0.6091707275594684
```

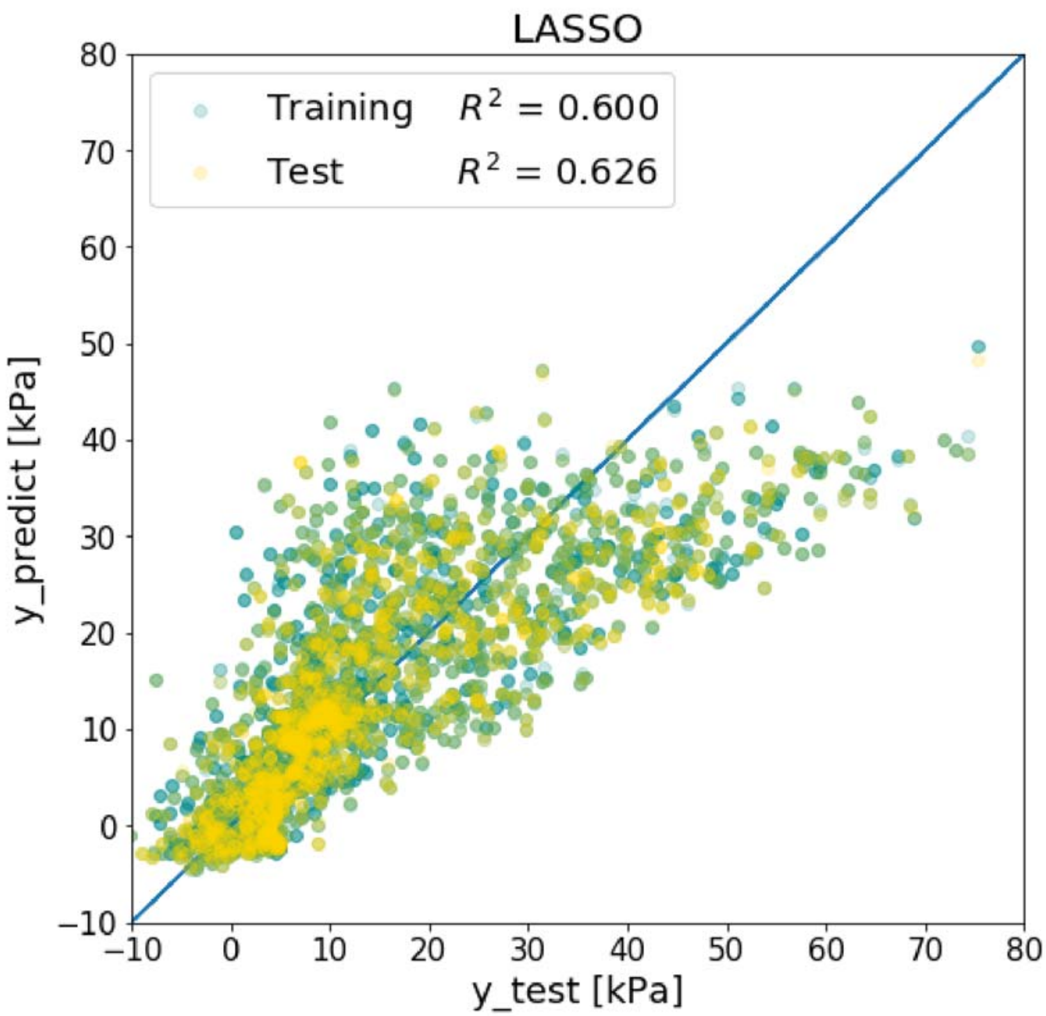

Multi-linear
Training $R^2 = 0.599$
Test $R^2 = 0.602$
y_predict [kPa]
y_test [kPa]
80
70
60
50
40
30
20
10
0
−10
−10
0
10
20
30
40
50
60
70
80

In [9]:

```
########################################### Baseline model 2 ################
# Lasso Regression - L1 norm regularization
from sklearn.linear_model import Lasso

alphas = np.array([1e-9, 1e-7, 1e-5, 1e-3])
sigmas = np.array([0.01, 0.05, 0.1, 0.25, 0.5])

findAlpha =[]
for alpha in alphas:
    findSigma = []
    for sigma in sigmas:
        gamma = 1. / 2 / sigma**2
        lasso = Lasso(alpha = alpha)

        lasso.fit(X_train, y_train)
        r2 = lasso.score(X_test, y_test)

        findSigma.append(r2)

    optSigmaIndex = findSigma.index(max(findSigma))
    suboptSigma = sigmas[optSigmaIndex]
    submaxR2 = max(findSigma)

    findAlpha.append((submaxR2, suboptSigma))

optAlphaIndex = findAlpha.index(max(findAlpha))
optAlpha = alphas[optAlphaIndex]
optSigma = findAlpha[optAlphaIndex][1]
maxR2 = findAlpha[optAlphaIndex][0]
print('\n\nOptimum Alpha = {}, Optimum Sigma = {}'.format(optAlpha, optSigma)

lasso = Lasso(alpha = alpha)
lasso.fit(X_train, y_train)
r2_test = lasso.score(X_test, y_test)
r2_train = lasso.score(X_train, y_train)
print('\nTraining r^2 = {}, Test r^2 = {}\n\n'.format(r2_train, r2_test))

# Technically improper to scale prior to GridSearchCV
# Optimum alpha & Sigma just go to lowest of range; negligible r2 impact
# Adding zero to alpha matrix causes nonconvergence errors

y_hat_test = lasso.predict(X_test)
y_hat_train = lasso.predict(X_train)

fig, ax = plt.subplots(figsize = (8,8))
ax.plot(y_test, y_test, '-')
ax.plot(y_train, y_hat_train, 'o', label = 'Training    $R^2$ = 0.600', color
ax.plot(y_test, y_hat_test, 'o', label = 'Test        $R^2$ = 0.626', color
plt.xlim(-10, 80)
plt.ylim(-10, 80)
ax.set_xlabel('y_test [kPa]', fontsize = 18)
ax.set_ylabel('y_predict [kPa]', fontsize = 18)
ax.set_title('LASSO', fontsize=20)
plt.legend(loc = 2, fontsize = 18)
plt.xticks(fontsize=15); plt.yticks(fontsize=15);
```

```
#plt.savefig('parity_lasso.png', transparent=True, bbox_inches="tight")
```

```
Optimum Alpha = 0.001, Optimum Sigma = 0.01

Training r^2 = 0.60151338669526, Test r^2 = 0.6092351679111244
```

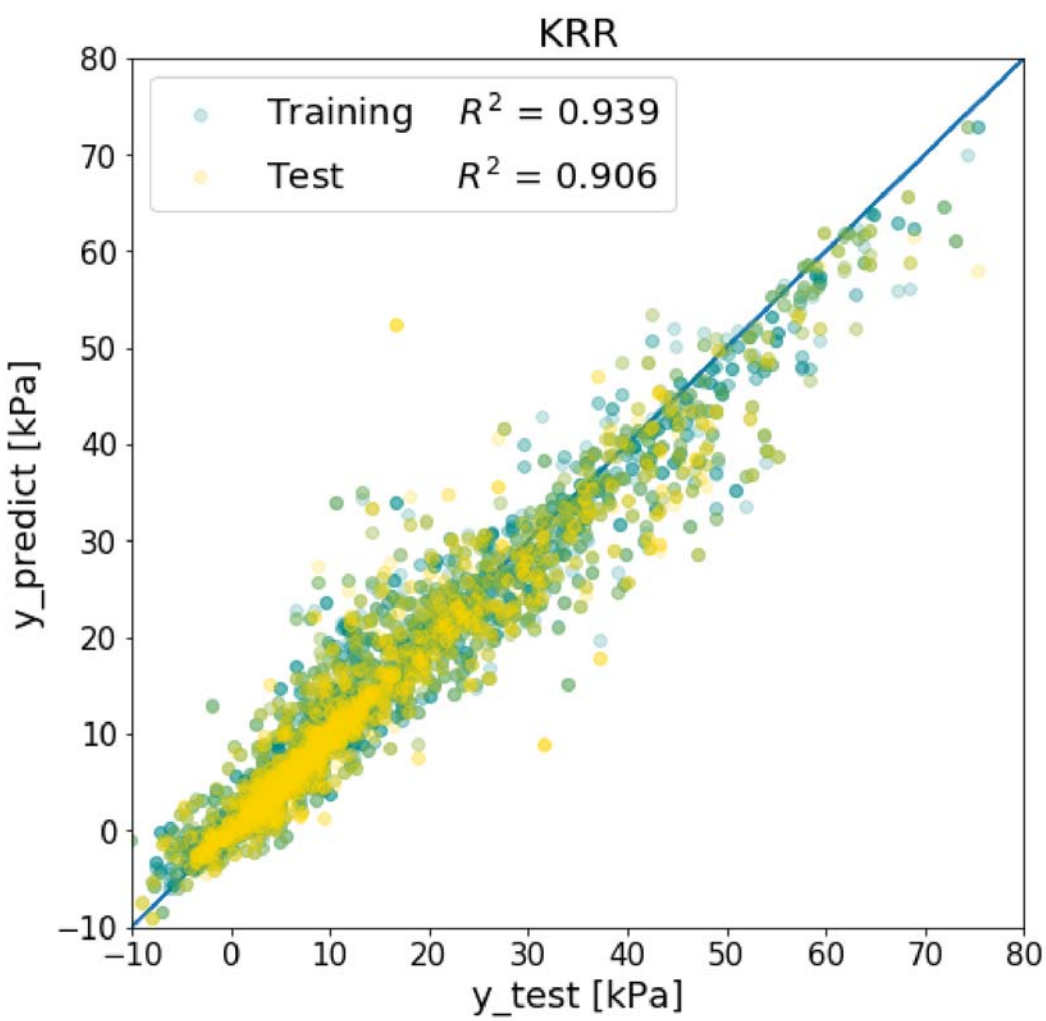


In [10]:

```
########################################## Baseline model 3 #################
###################### Kernel Ridge Regression - l2 norm regularization

from sklearn.model_selection import GridSearchCV
# alphas = np.array([1e-3, 1e-2, 1e-1, 1])
# sigmas = np.array([0.1, 0.2, 0.35, 0.60, 1])
alphas = np.array([0.1, 0.5, 1, 10])
sigmas = np.array([0.1, 0.5, 1, 10])
gammas = 1. / 2 / sigmas**2

from sklearn.kernel_ridge import KernelRidge
KRRparam_grid = {'alpha':alphas, 'gamma':gammas}
KRR = KernelRidge(kernel = 'rbf')
KRR_search = GridSearchCV(KRR, KRRparam_grid, cv = 3)
# Line below runs slowly, tried to optimize param grid some to speed up
KRR_search.fit(X_train, y_train)

optAlpha = KRR_search.best_estimator_.alpha
optGamma = KRR_search.best_estimator_.gamma
optSigma = np.sqrt(1. / 2 / optGamma)
r2_train = KRR_search.best_estimator_.score(X_train, y_train)
r2_test = KRR_search.best_estimator_.score(X_test, y_test)
print('\n\nOptimum Alpha = {}, Optimum Sigma = {}, Optimum Gamma = {}'.format
print('\nTraining r^2 = {}, Test r^2 = {}\n\n'.format(r2_train, r2_test))

model = KernelRidge(alpha = optAlpha, kernel = 'rbf', gamma=optGamma).fit(X_t
y_hat_test = model.predict(X_test)
y_hat_train = model.predict(X_train)
```

```
Optimum Alpha = 0.1, Optimum Sigma = 0.5, Optimum Gamma = 2.0

Training r^2 = 0.9406102446010042, Test r^2 = 0.9022710469462694
```

In [11]:
```
fig, ax = plt.subplots(figsize = (8,8))
ax.plot(y_test, y_test, '-')
ax.plot(y_train, y_hat_train, 'o', label = 'Training    $R^2$ = 0.939', color
ax.plot(y_test, y_hat_test, 'o', label = 'Test          $R^2$ = 0.906', color
plt.xlim(-10, 80)
plt.ylim(-10, 80)
ax.set_xlabel('y_test [kPa]', fontsize = 18)
ax.set_ylabel('y_predict [kPa]', fontsize = 18)
ax.set_title('KRR', fontsize=20)
plt.legend(loc = 2, fontsize = 18)
plt.xticks(fontsize=15); plt.yticks(fontsize=15);

#plt.savefig('parity_KRR.png', transparent=True, bbox_inches="tight")
```



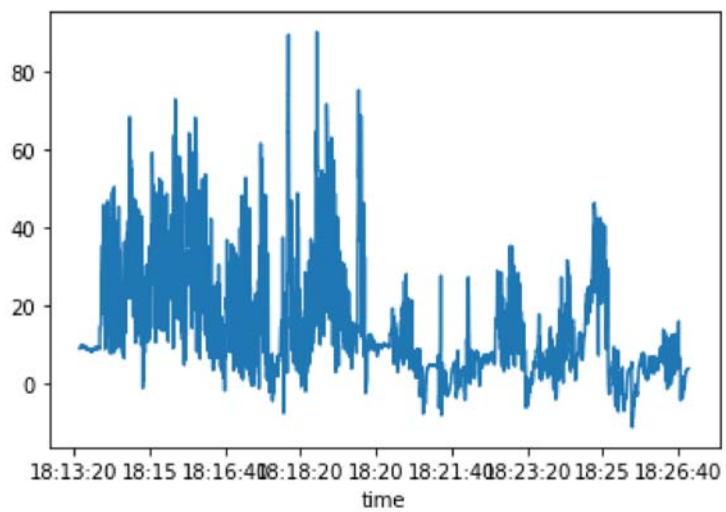
Discussion

1. Training data set and test data set have comparable R^2
2. Multilinear regression and LASSO tend to underestimated P6 in the high pressure ragion
3. Heteroscedasticity error
4. KRR modle is best
5. Will add time series in future

4. Time-series

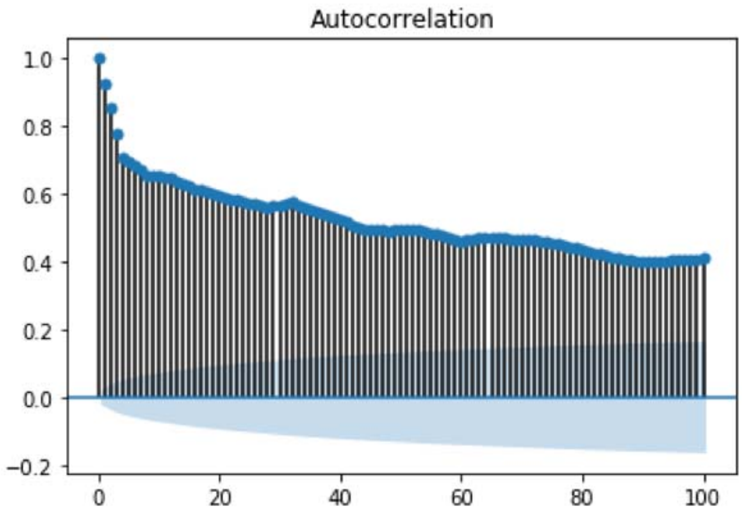
In the following section, we will give a preview of using time-series model. Intuitively, our database: the two-phase flow correlation should not be time-dependent. However the reason we incorprate time series model here lies in: 1. the trasition state between two steady state is time-depedent, and thus we would like to see the time-dependent effect in short term; 2. make sure the design of experiment is reasonable and there is no long-term time dependency.

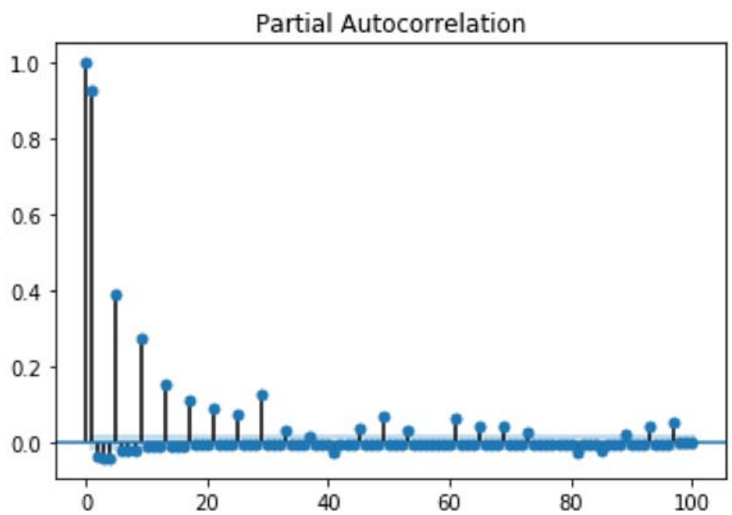
In [12]:
```
# data vitulization
pd.plotting.register_matplotlib_converters()
plt.plot(df[df.columns[0]],y);
```



In [13]:
```
from statsmodels.tsa.stattools import acf
from statsmodels.graphics.tsaplots import plot_acf, plot_pacf

autocorr = acf(y, nlags = 20,fft=False)
print(autocorr)
plot_acf(y, lags = 100);
plot_pacf(y, lags = 100);
```

```
[1.         0.92644135 0.85288225 0.77932315 0.70576406 0.69353974
 0.6813155  0.66909127 0.65686703 0.65404314 0.65121926 0.64839538
 0.6455715  0.63804791 0.63052376 0.62299961 0.61547546 0.60984241
 0.60420917 0.59857593 0.59294269]
```





We see that the our dataset is actually astonishingly autocorrelated past 100 prior points, although the autocorrelation decays rapidly. The partial autocorrelation falls off quickly, with only 2 prior points having significant partial autocorrelations, and no long-term seasonal variations are visisble.

This make senses because we expect the system reaches equilibrium relatively quick. Also we see some scatter data points priorer than 5 steps also have high partial correlation, this probably due to the experiment design flaw.

Overall, this is just a sample test for the incorpration of time-series theory while more details will be potentially included in the following report.

5. Model improvement

There is a lag between pressure step change and flow step change in the original data, so we tried to remove the lag data point in order to improve the correlation.

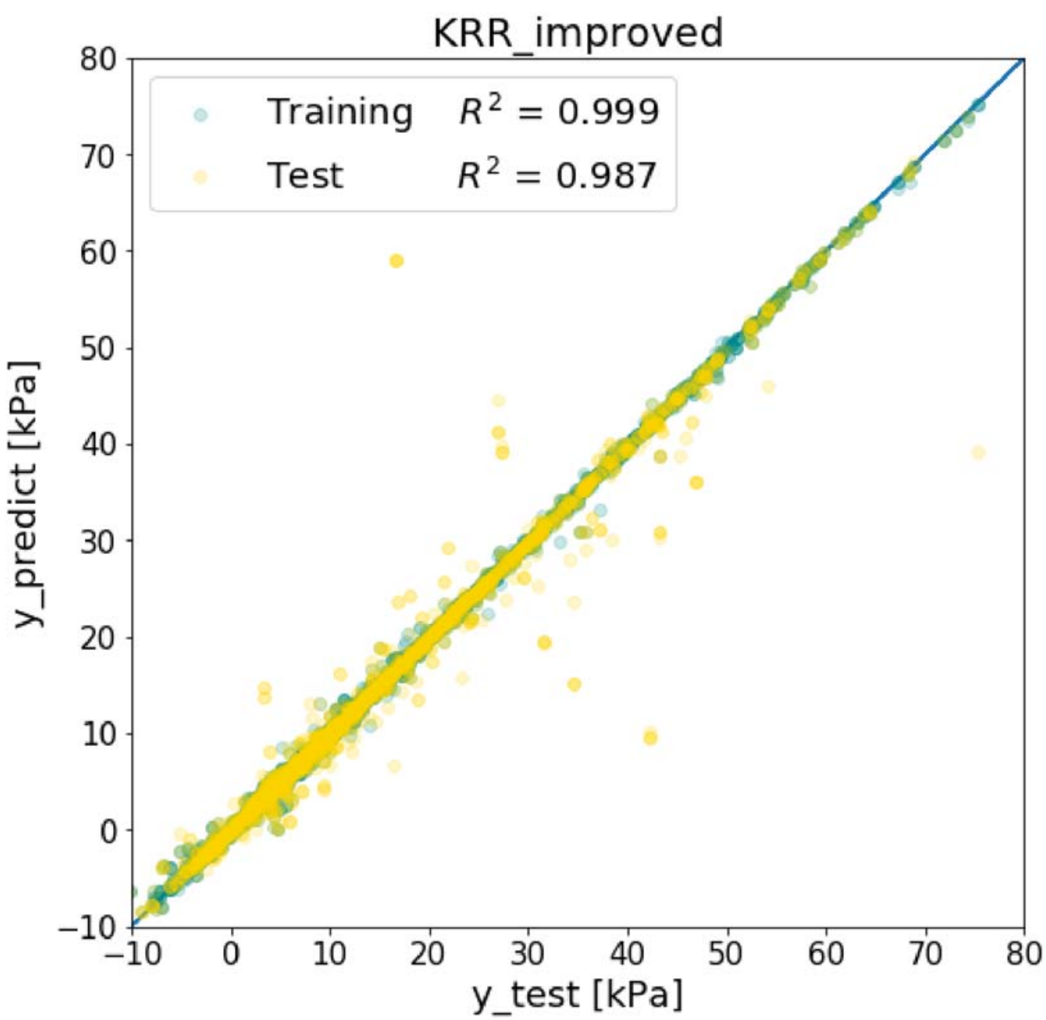
In [14]:

```
########################################### hyperparameter tuning ###########
###################### Kernel Ridge Regression - l2 norm regularization
from sklearn.model_selection import GridSearchCV
alphas = np.array([0.005, 0.01, 0.05, 0.1, 0.5])
sigmas = np.array([0.1, 0.3, 0.5, 0.8, 1])
gammas = 1. / 2 / sigmas**2

from sklearn.kernel_ridge import KernelRidge
KRRparam_grid = {'alpha':alphas, 'gamma':gammas}
KRR = KernelRidge(kernel = 'rbf')
KRR_search = GridSearchCV(KRR, KRRparam_grid, cv = 3)
# Line below runs slowly, tried to optimize param grid some to speed up
KRR_search.fit(X_train, y_train)

optAlpha = KRR_search.best_estimator_.alpha
optGamma = KRR_search.best_estimator_.gamma
optSigma = np.sqrt(1. / 2 / optGamma)
r2_train = KRR_search.best_estimator_.score(X_train, y_train)
r2_test = KRR_search.best_estimator_.score(X_test, y_test)
print('\n\nOptimum Alpha = {}, Optimum Sigma = {}, Optimum Gamma = {}'.format
print('\nTraining r^2 = {}, Test r^2 = {}\n\n'.format(r2_train, r2_test))

model = KernelRidge(alpha = optAlpha, kernel = 'rbf', gamma=optGamma).fit(X_t
y_hat_test = model.predict(X_test)
y_hat_train = model.predict(X_train)
```

```
Optimum Alpha = 0.01, Optimum Sigma = 0.3, Optimum Gamma = 5.55555555555555
5

Training r^2 = 0.9990159289139994, Test r^2 = 0.9621052245714583
```

In [15]:

```
fig, ax = plt.subplots(figsize = (8,8))
ax.plot(y_test, y_test, '-')
ax.plot(y_train, y_hat_train, 'o', label = 'Training    $R^2$ = 0.999', color
ax.plot(y_test, y_hat_test, 'o', label = 'Test        $R^2$ = 0.987', color
plt.xlim(-10, 80)
plt.ylim(-10, 80)
ax.set_xlabel('y_test [kPa]', fontsize = 18)
ax.set_ylabel('y_predict [kPa]', fontsize = 18)
ax.set_title('KRR_improved', fontsize=20)
plt.legend(loc = 2, fontsize = 18)
plt.xticks(fontsize=15); plt.yticks(fontsize=15);

#plt.savefig('KRR_improved.png', transparent=True, bbox_inches="tight")
```



In [16]:

```
############################################### loading no_lag data ###########
df = pd.read_csv('no_lag.csv')

def is_real_and_finite(x):
    if not np.isreal(x):
        return False
    elif not np.isfinite(x):
        return False
    else:
        return True

all_data = df[df.columns[2:]].values
numeric_map = df[df.columns[2:]].applymap(is_real_and_finite)
real_rows = numeric_map.all(axis = 1).copy().values
X = np.array(all_data[real_rows, 1:], dtype = 'float')
y = np.array(all_data[real_rows, 0], dtype = 'float')
y = y.reshape(-1, 1)
x_names = [str(x) for x in df.columns[3:]]
y_name = df.columns[2]

print('X matrix dimensions: {}'.format(X.shape))
print('y matrix dimensions: {}'.format(y.shape))

############################################ data preprocessing #############

################### feature scaling
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()
scaler.fit(X)
X_scaled = scaler.transform(X)
#print('Total data size = {}'. format(X_scaled.shape))

################### data spliting
from sklearn.model_selection import train_test_split

# 70%-training, 30%-test
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.
print('Train size = {}, Test size = {}'.format(y_train.shape, y_test.shape))
```

```
X matrix dimensions: (1577, 7)
y matrix dimensions: (1577, 1)
Train size = (1103, 1), Test size = (474, 1)
```

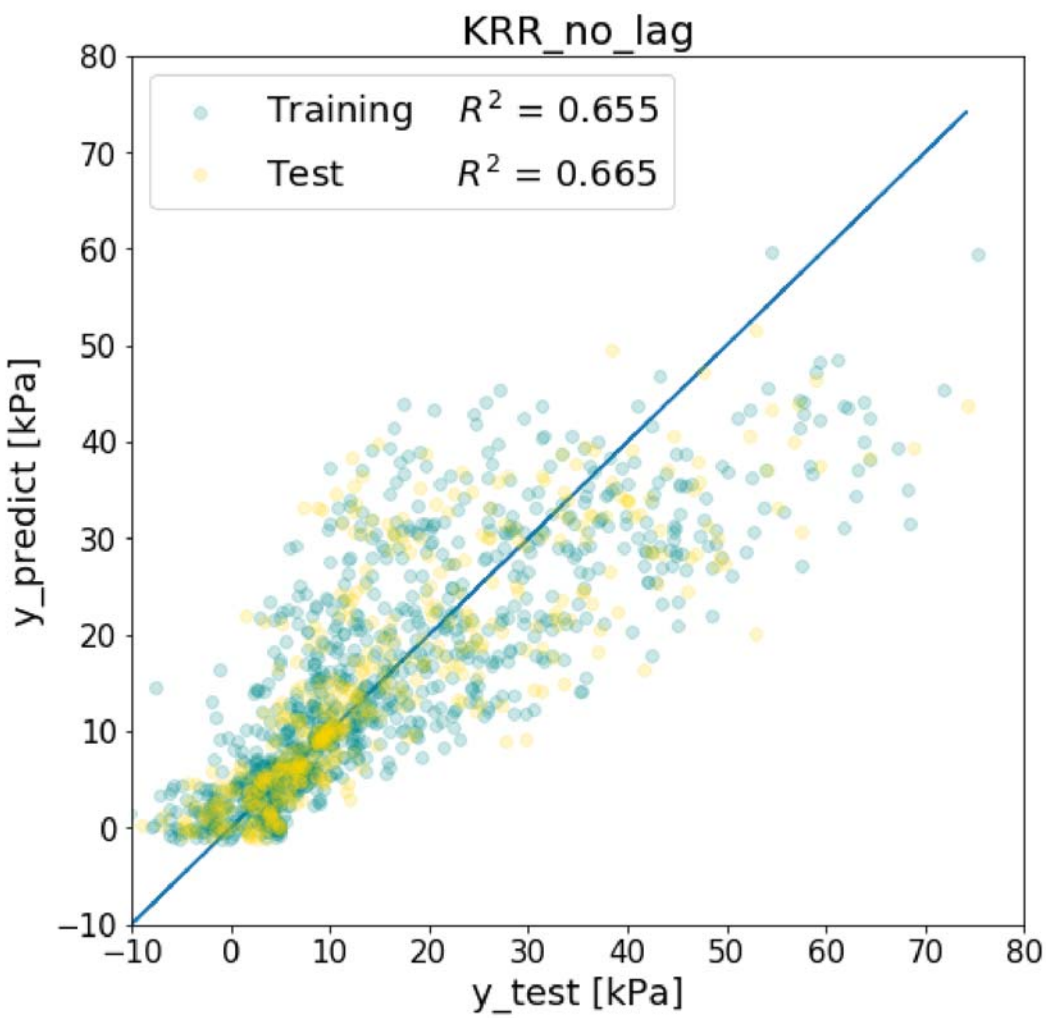
In [17]:

```
########################################### KRR with no_lag data ############
###################### Kernel Ridge Regression - l2 norm regularization
from sklearn.model_selection import GridSearchCV
alphas = np.array([0.01, 0.05, 0.1])
sigmas = np.array([0.8, 1, 5, 10])
gammas = 1. / 2 / sigmas**2

from sklearn.kernel_ridge import KernelRidge
KRRparam_grid = {'alpha':alphas, 'gamma':gammas}
KRR = KernelRidge(kernel = 'rbf')
KRR_search = GridSearchCV(KRR, KRRparam_grid, cv = 3)
# Line below runs slowly, tried to optimize param grid some to speed up
KRR_search.fit(X_train, y_train)

optAlpha = KRR_search.best_estimator_.alpha
optGamma = KRR_search.best_estimator_.gamma
optSigma = np.sqrt(1. / 2 / optGamma)
r2_train = KRR_search.best_estimator_.score(X_train, y_train)
r2_test = KRR_search.best_estimator_.score(X_test, y_test)
print('\n\nOptimum Alpha = {}, Optimum Sigma = {}, Optimum Gamma = {}'.format
print('\nTraining r^2 = {}, Test r^2 = {}\n\n'.format(r2_train, r2_test))

model = KernelRidge(alpha = optAlpha, kernel = 'rbf', gamma=optGamma).fit(X_t
y_hat_test = model.predict(X_test)
y_hat_train = model.predict(X_train)
```

```
Optimum Alpha = 0.05, Optimum Sigma = 5.0, Optimum Gamma = 0.02

Training r^2 = 0.6629242389120062, Test r^2 = 0.6605534283215361
```

In [18]:

```
fig, ax = plt.subplots(figsize = (8,8))
ax.plot(y_test, y_test, '-')
ax.plot(y_train, y_hat_train, 'o', label = 'Training    $R^2$ = 0.655', color
ax.plot(y_test, y_hat_test, 'o', label = 'Test        $R^2$ = 0.665', color
plt.xlim(-10, 80)
plt.ylim(-10, 80)
ax.set_xlabel('y_test [kPa]', fontsize = 18)
ax.set_ylabel('y_predict [kPa]', fontsize = 18)
ax.set_title('KRR_no_lag', fontsize=20)
plt.legend(loc = 2, fontsize = 18)
plt.xticks(fontsize=15); plt.yticks(fontsize=15);

#plt.savefig('KRR_no_lag.png', transparent=True, bbox_inches="tight")
```



In [ ]:

```

```