Contents

Liam Jackson HW1 BE700 ML

Question 1

```
%{
Ok, honestly this code is overcomplicated. I had to rewrite it 5
times because MATLAB didn't save my changes a couple days in a row. So
in the interest of time, I tried writing functions to accomplish the
analysis for question 1 that I could then recycle for question 2. It's
ugly, I switch data structure types all over the place. But hopefully I
approached the correct answers in the end.
%}
```

Part 1

```
close all, clear all, clc;
warning('off','MATLAB:polyfit:RepeatedPointsOrRescale')
warning('off','MATLAB:nearlySingularMatrix')
```

Importing / Sorting Data

```
[x1, x2, y] = textread('besseldata.txt', ' %f%f%f', 'headerlines', 1);
r = sqrt(x1.^2 + x2.^2);
r_norm = normalize(r);

data_arr = sortrows([x1, x2, r, r_norm, y], 3);
data_table = array2table(data_arr,...
    'VariableNames', {'x1','x2','r','r_norm','y'});
```

Bessel Approx

```
k_bes = 1;
bes_approx = besselj(0, k_bes*data_table.r);

fig1 = figure(1);
dot_sz = 0.2;
line_w = 2.5;
scatter(data_table.r, data_table.y, dot_sz, '.');
hold on;
plot(data_table.r, bes_approx, 'LineWidth', line_w);
hold off;
title({'Timpanic Memb Displacement', 'approximated by Bessel Fxn (J_0)'});
xlabel('r');
ylabel('Intensity');
legend({'Real Data', 'J_0'});
```

Polynomial Approximations

```
max_poly_order = 14;
model_data = poly_model_vals(data_table, max_poly_order);
ry_polyvals_table = model_data.ry_polyvals_table;
```

Calculate Residuals

```
residuals_table = res_table(ry_polyvals_table)
```

Plotting LS Poly fits

```
data_labels = ry_polyvals_table.Properties.VariableNames;

fig2 = figure(2);
sgtitle({'Membrane Displacement Data', 'vs. OLS Polynomial Fits'});
number_of_plots = max_poly_order;

for plot_id = 1:number_of_plots
    subplot(number_of_plots / 2, 2, plot_id);
    scatter(ry_polyvals_table.r, ry_polyvals_table.y, dot_sz, '.');
    hold on;
    plot(ry_polyvals_table.r, ry_polyvals_table.(string(data_labels(plot_id + 2))), 'LineWidth', line_w)
    hold off;
    xlabel('r');
    ylabel('Displacement');
    legend({'y real', string(data_labels(plot_id + 2))});
end
```

Part 2

20 rounds of (k = 5) Cross Validation

```
cv_rounds = 20;
k_cv = 5;

PE_arr = zeros([cv_rounds, max_poly_order]);
MSE_arr = zeros([max_poly_order, k_cv, cv_rounds]);
all_cv_poly_coeffs = zeros([max_poly_order + 1, max_poly_order, k_cv, cv_rounds]);

for cv_round = 1:cv_rounds
    binned_data_struct = bin_this_data(data_arr, k_cv);
    binned_data_cell = binned_data_struct.cell;
    bin_indices = 1:k_cv;

    for test_bin = 1:k_cv
        train_bins = bin_indices(1:end ~= test_bin);
        train_data_cell = binned_data_cell(train_bins);

        train_data_arr = sortrows(cat(1, train_data_cell{:}), 3);
        test_data_arr = sortrows(cell2mat(binned_data_cell(test_bin)), 3);

        train_data_table = array2table(train_data_arr,...
            'VariableNames', {'x1','x2','r','r_norm','y'});
        test_data_table = array2table(test_data_arr,...
            'VariableNames', {'x1','x2','r','r_norm','y'});

        model_train_struct = poly_model_vals(train_data_table, max_poly_order);
        cv_ry_polyvals_table = model_train_struct.ry_polyvals_table;
        cv_coeffs_arr = model_train_struct.coeffs_arr;
        all_cv_poly_coeffs(:, :, test_bin, cv_round) = cv_coeffs_arr;

        poly_zeros_pad = zeros([size(test_data_arr, 1), max_poly_order]);
        model_poly_vals = [test_data_arr, poly_zeros_pad];
        for poly_ord_ind = 1:max_poly_order
            n_coeffs = poly_ord_ind + 1;
            temp_coeffs = cv_coeffs_arr(1:n_coeffs, poly_ord_ind);
            model_poly_vals(:, poly_ord_ind + 5) = polyval(temp_coeffs, model_poly_vals(:, 4));
        end

        temp_ry_polyvals_table = array2table([model_poly_vals(:, 3), model_poly_vals(:, 5), model_poly_vals(:,6:end)],...
            'VariableNames', cv_ry_polyvals_table.Properties.VariableNames);
        temp_residuals_table = res_table(temp_ry_polyvals_table);
        MSE_arr(:, test_bin, cv_round) = temp_residuals_table.MSE;

    end
    PE_col = mean(squeeze(MSE_arr(:, :, cv_round)), 2);
    PE_arr(cv_round, :) = PE_col';
end

PE_var_labels = cv_ry_polyvals_table.Properties.VariableNames;
PE_var_labels = PE_var_labels(3:end);
PE_row_nums = 1:cv_rounds;
PE_row_labels = "rnd" + PE_row_nums;

PE_table = array2table(PE_arr,...
    'VariableNames', PE_var_labels,...
    'RowNames', PE_row_labels)
```

Plotting PE values for each CV Round

```
fig3 = figure(3);
plot(PE_table{:,:}.');
title('PE values for 20 rounds of (k=5)-CV');
xlabel('Polynomial Model Order');
ylabel('Predictive Error');
legend(PE_table.Properties.RowNames, 'location', 'eastoutside');
```

Part 3

```
%{
A polynomial OLS-fit of order 10 seems to have the best compromise of
accuracy and economy of variables. A substantial reduction in error
occurs from order-9 to order-10, with no substantial decrease with
additional (11, 12, 13, 14) order terms.
%}

char({'A polynomial OLS-fit of order 10 seems to have the best',...
    'compromise of accuracy and economy of variables. A substantial',...
    'reduction in error occurs from order-9 to order-10, with no',...
    'substantial decrease with additional (11, 12, 13, 14) order terms.'})
```

Part 4

```
data_table_opt = data_table;

x1 = data_table_opt.x1;
x2 = data_table_opt.x2;
r = data_table_opt.r;
y = data_table_opt.y;

opt_ord = 10;
beta_deg10 = ols_coeffs_data(r, y, opt_ord).beta;
beta_ud = flipud(beta_deg10);

[X1, X2] = meshgrid(-20:.2:20);
Y_opt = polyval(beta_ud, sqrt(X1.^2 + X2.^2));

% I'm removing the "wall" of the surf that approaches inf so the figure is easier to see
Y_opt(1:75, 1:75) = NaN;

J0 = besselj(k_bes, sqrt(X1.^2 + X2.^2));

fig4 = figure(4);
scatter3(x1, x2, y, 15, '.');
hold on;
opt = surf(X1, X2, Y_opt, 'EdgeColor', 'none');
colorbar
colormap(spring)
caxis([-1 1.5])
hold off;
title({'Polynomial (p=10) Model', 'vs. Real Displacement Data'});
xlabel('x1');
ylabel('x2');
zlabel('Displacement');
zlim([-.8, 1.5]);


fig5 = figure(5);
scatter3(x1, x2, y, 15, '.');
hold on;
surf(X1, X2, J0, 'EdgeColor', 'none');
title({'Bessel Fxn J_0', 'vs. Real Displacement Data'});
xlabel('x1');
ylabel('x2');
zlabel('Displacement');
```

```
residuals_table =

  14×3 table
```

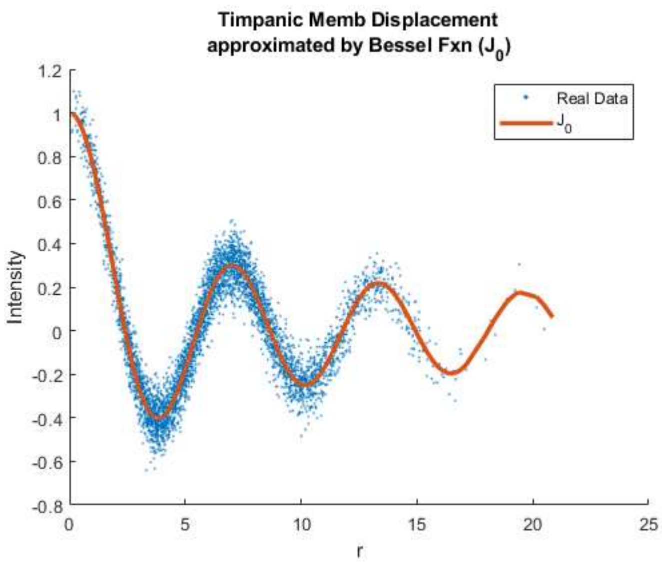
Polynomial_Order	Residual Sum	MSE
1	425.17	0.085034
2	412.42	0.082483
3	369.4	0.073879
4	268.61	0.053722
5	168.66	0.033733
6	164.73	0.032945
7	105.34	0.021069
8	47.384	0.0094768
9	43.922	0.0087844
10	28.828	0.0057656
11	28.809	0.0057619
12	27.775	0.0055551
13	27.718	0.0055437
14	27.697	0.0055394

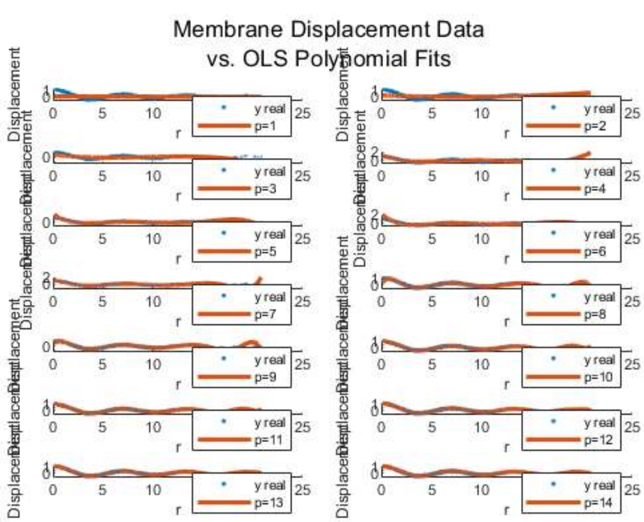
```
PE_table =

  20×14 table
```

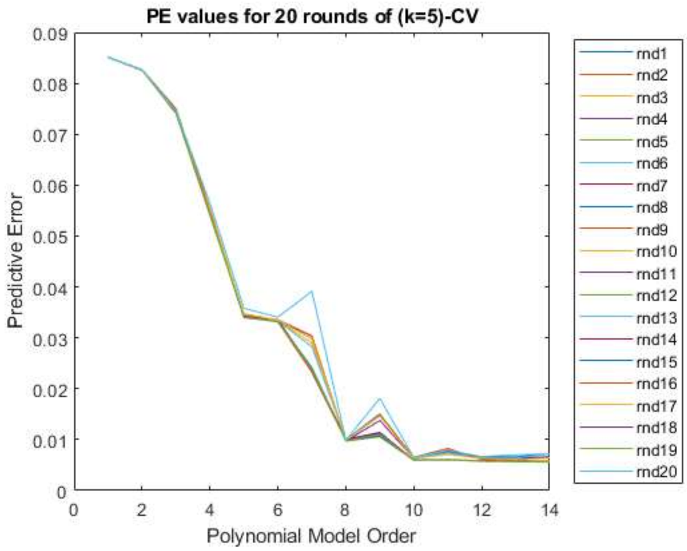
	p=1	p=2	p=3	p=4	p=5	p=6	p=7	p=8	p=9	p=10	p=11	p=12	p
rnd1	0.085065	0.082577	0.074357	0.054969	0.034563	0.033381	0.028947	0.0098362	0.01501	0.006425	0.0076866	0.0065539	

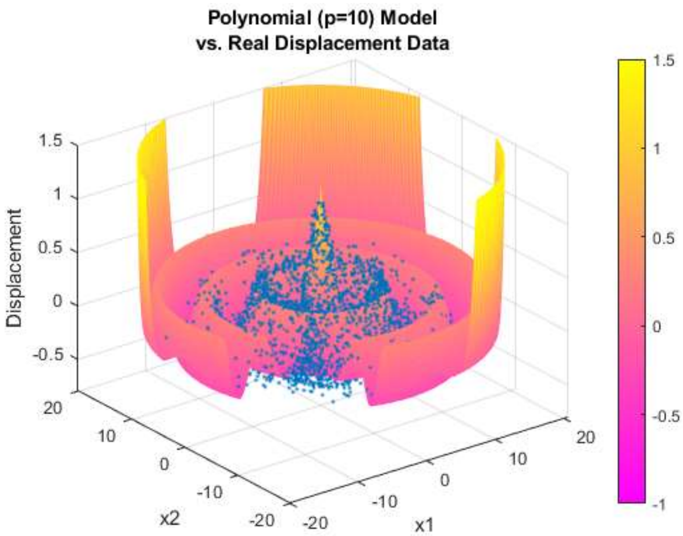
rnd2	0.085108	0.082569	0.074034	0.054003	0.034081	0.033134	0.02323	0.0098505	0.010745	0.0059828	0.0059781	0.0058127	0.0
rnd3	0.085127	0.082699	0.074457	0.054531	0.034317	0.03327	0.023955	0.0097887	0.010689	0.005926	0.0060101	0.0057978	0.0
rnd4	0.085134	0.082691	0.074525	0.054411	0.034338	0.033372	0.024019	0.010069	0.011438	0.0059981	0.0060173	0.0057865	0.0
rnd5	0.085151	0.082621	0.074146	0.054444	0.0343	0.033315	0.023954	0.0096891	0.010574	0.0059484	0.0061251	0.0057965	0.0
rnd6	0.085094	0.082574	0.07405	0.053968	0.034146	0.033214	0.024222	0.0098553	0.01086	0.0059703	0.0060338	0.0057492	0.0
rnd7	0.085065	0.082532	0.074127	0.054713	0.034718	0.033477	0.023915	0.010057	0.011002	0.0060462	0.0060709	0.0058321	0.0
rnd8	0.085165	0.08267	0.074137	0.054209	0.034092	0.033178	0.023494	0.0098238	0.010947	0.0059741	0.0060757	0.0058071	0.0
rnd9	0.085079	0.082579	0.074176	0.054251	0.034168	0.033249	0.023309	0.0098935	0.01112	0.0059834	0.0060462	0.0058289	0.0
rnd10	0.085113	0.08259	0.074421	0.05493	0.034551	0.03328	0.028994	0.0099285	0.015095	0.0063835	0.0071666	0.0062601	0.0
rnd11	0.085114	0.082641	0.074214	0.054661	0.034494	0.033468	0.030297	0.0097173	0.013768	0.0063214	0.0076979	0.0062456	0.0
rnd12	0.08514	0.082657	0.074335	0.054417	0.034287	0.03327	0.023885	0.0097416	0.010559	0.0059619	0.0060269	0.0057999	0.0
rnd13	0.085102	0.082591	0.074369	0.055	0.034708	0.033352	0.028202	0.010054	0.015138	0.006425	0.0074468	0.0063672	0.0
rnd14	0.085142	0.082635	0.074044	0.054013	0.034197	0.033606	0.023737	0.0099251	0.011152	0.0059168	0.0060737	0.0057321	0.0
rnd15	0.085182	0.082649	0.074122	0.054086	0.034096	0.033247	0.024024	0.0097745	0.01078	0.0059868	0.006066	0.0058092	0.0
rnd16	0.085159	0.082704	0.075006	0.055209	0.034545	0.033553	0.030413	0.0098368	0.014896	0.0065433	0.0082615	0.0060834	0.0
rnd17	0.08509	0.082586	0.074257	0.05483	0.034678	0.033489	0.029756	0.009869	0.014629	0.0062817	0.0070891	0.006335	0.0
rnd18	0.085052	0.082557	0.074247	0.05421	0.033971	0.033215	0.02378	0.0098048	0.011342	0.0059427	0.0060055	0.0058236	0.
rnd19	0.085087	0.082559	0.07409	0.054043	0.034102	0.033235	0.023818	0.0097457	0.010996	0.0059363	0.0060449	0.0057481	0.0
rnd20	0.085162	0.082648	0.074607	0.056379	0.035793	0.034067	0.039151	0.0099011	0.018097	0.0064823	0.0079107	0.0066796	0.0

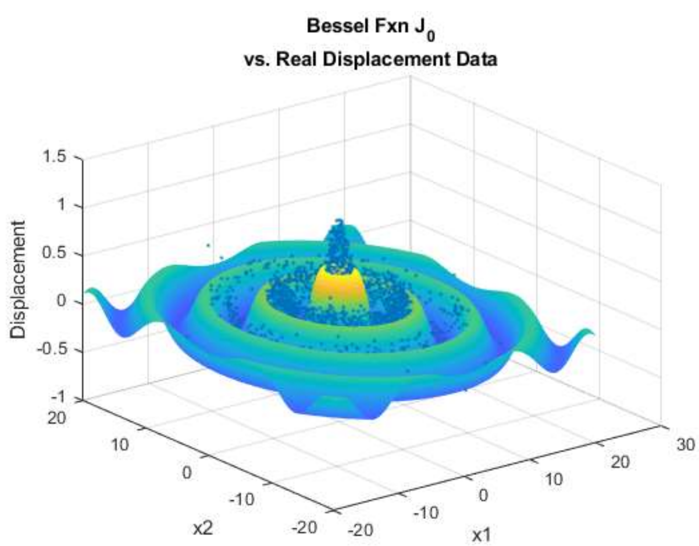
```
ans =

  4×66 char array

    'A polynomial OLS-fit of order 10 seems to have the best             '
    'compromise of accuracy and economy of variables. A substantial     '
    'reduction in error occurs from order-9 to order-10, with no        '
    'substantial decrease with additional (11, 12, 13, 14) order terms.'
```






PE values for 20 rounds of (k=5)-CV
Predictive Error
Polynomial Model Order
rnd1
rnd2
rnd3
rnd4
rnd5
rnd6
rnd7
rnd8
rnd9
rnd10
rnd11
rnd12
rnd13
rnd14
rnd15
rnd16
rnd17
rnd18
rnd19
rnd20


Polynomial (p=10) Model
vs. Real Displacement Data
Displacement
x2
x1


Bessel Fxn J_0
vs. Real Displacement Data
Displacement
x2
x1

Contents

Question 2

```
close all, clear all, clc;
warning('off','MATLAB:table:ModifiedAndSavedVarnames')
```

Part 1

Importing Data

```
data_table_raw = readtable('winequality_red.csv');

t = data_table_raw.('citricAcid');
u = (t - mean(t))./std(t); %standardizing t
y = data_table_raw.('fixedAcidity');

var_names = {'t','u','y'};
data_table = sortrows(array2table([t, u, y],...
    'VariableNames', var_names), 't');
data_arr = table2array(data_table);
```

Scatter Matrix

```
A = table2array(data_table_raw);
fig1 = figure(1);
plotmatrix(A);
title('Scatter Plot Matrix of Wine Data')
```

Part 2

100 PE Curves based on (p = 9) Poly Model w/ L2 Reg, \alpha = 0:50:3000

```
p = 9;
k_cv = 5;
alpha_range = 0:50:3000;
num_PE_curves = 100;

PE_arr = [alpha_range', zeros([length(alpha_range), num_PE_curves])];

for PE_curve = 1:1:num_PE_curves
```

```
    binned_data_struct = bin_this_data(data_arr, k_cv);

    binned_data_cell = binned_data_struct.cell;
    bin_indices = 1:k_cv;

    for alpha = alpha_range

        alpha_ind = find(alpha_range == alpha);
        MSE_5fold = zeros([k_cv, 1]);

        for test_bin = 1:k_cv
            train_bins = bin_indices(1:end ~= test_bin);
            train_data_cell = binned_data_cell(train_bins);

            train_data_arr = sortrows(cat(1, train_data_cell{:}));
            test_data_arr = sortrows(cell2mat(binned_data_cell(test_bin)));

            train_data_table = array2table(train_data_arr,...
                'VariableNames', var_names);
            test_data_table = array2table(test_data_arr,...
                'VariableNames', var_names);

            u_train = train_data_table.u;
            y_train = train_data_table.y;
            u_test = test_data_table.u;
            y_test = test_data_table.y;

            X = ols_coeffs_data(u_train, y_train, p).X;
            X(:,1) = [];

            B = ridge(y_train, X, alpha, 0);
            beta_flip = flipud(B);


            y_model = polyval(beta_flip, u_test);

            res_sq_sum = residuals(y_test, y_model);
            MSE_5fold(test_bin, 1) = res_sq_sum / length(y_test);

        end
        PE_arr(alpha_ind, PE_curve + 1) = mean(MSE_5fold);
    end
end
```

Plot all PE Curves

```
fig2 = figure(2);
for PE_curve_id = 1:num_PE_curves
    plot(PE_arr(:, 1), PE_arr(:, PE_curve_id + 1));
    hold on;
end
title({'100 Curves of Predictive Error', '(of a 9th order polynomial model)', 'vs. \alpha in L2-Regularization'});
ylim([0, 10])
xlabel('\alpha value')
ylabel('PE')
hold off;

opt_alpha = 400;
char('It looks the optimal alpha is maybe ~400 ish')
```

Part 3

```
u_full = data_table.u;
y_full = data_table.y;

X_full_std = ols_coeffs_data(u_full, y_full, 9).X;
X_full_std(:,1) = [];

w_opt_alpha400 = ridge(y_full, X_full_std, opt_alpha, 0);
w_opt_alpha400_flip = flipud(w_opt_alpha400);

w_alpha0 = ridge(y_full, X_full_std, 0, 0);
w_alpha0_flip = flipud(w_alpha0);

u_linear = linspace(min(u_full), max(u_full), 1500);
y_model_opt = polyval(w_opt_alpha400_flip, u_linear);

w_ols = ols_coeffs_data(u_full, y_full, 9).beta;
w_ols_flip = flipud(w_ols);

y_model_ols = polyval(w_ols_flip, u_linear);
```

Part 4

```
fig3 = figure(3);
scatter(u_full, y_full, 100, '.')
hold on;
plot(u_linear, y_model_ols, 'r--')
plot(u_linear, y_model_opt, 'g-.')
title({'Raw Data', 'vs OLS fit', 'vs L2 Regularized Fit (\alpha = 400)'});
xlabel('u (standardized t) [Citric Acid Content]')
ylabel('Fixed acidity')
legend({'Raw Data', 'OLS (p = 9)', 'L2 Reg'}, 'location', 'southwest')

char('L2 Ridge (alpha = 0) yields OLS coeffs')
coeff_table_all = array2table([w_ols, w_alpha0, w_opt_alpha400],...
    'VariableNames', {'w_ols', 'w_L2 (alpha = 0)', 'w_L2_opt (alpha = 400)'},...
    'RowNames', {'p=1','p=2','p=3','p=4','p=5','p=6','p=7','p=8','p=9','p=10'})
```

Functions

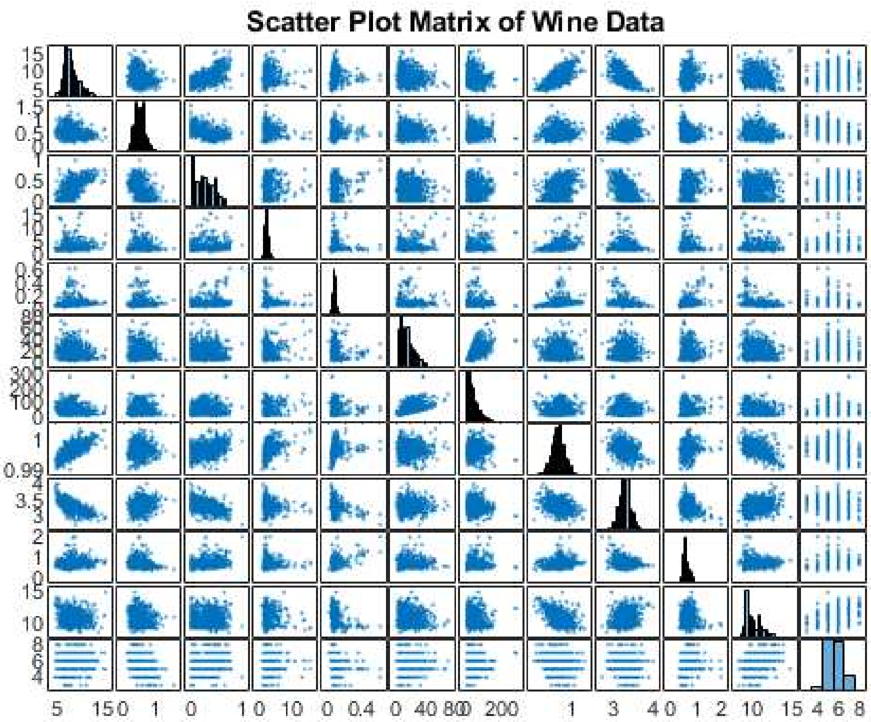
```
function res_squared = residuals(y_real, y_model)
res_squared = sum(abs(y_real - y_model).^2);
end

function residuals_table = res_table(ry_polyvals_table)

max_poly_order = size(ry_polyvals_table, 2) - 2;
residuals_arr = zeros([max_poly_order, 3]);

for order_ind = 1:max_poly_order
    y_real = ry_polyvals_table.y;
    y_model = table2array(ry_polyvals_table(:, order_ind + 2));
    residuals_arr(order_ind, 1) = order_ind;
    residuals_arr(order_ind, 2) = residuals(y_real, y_model);
    residuals_arr(order_ind, 3) = residuals_arr(order_ind, 2) ./ size(ry_polyvals_table, 1);
end

residuals_table = array2table(residuals_arr,...
    'VariableNames',{'Polynomial_Order', 'Residual Sum', 'MSE'});
end

function ols_data = ols_coeffs_data(x, y, poly_order)
```

```
ols_data = struct();

X = zeros(length(x), poly_order + 1);
X(:, 1) = 1;

for ord_ind = 1:poly_order
    X(:, ord_ind + 1) = x.^(ord_ind);
end

beta = (X' * X) \ (X' * y);

ols_data.beta = beta;
ols_data.X = X;
ols_data.res_squares_sum = norm(y - X*beta).^2;

end

function model_data_struct = poly_model_vals(data_table, max_poly_order)
model_data_struct = struct();

r = data_table.r;
r_norm = data_table.r_norm;
y = data_table.y;

n_coeffs = max_poly_order + 1;

coeffs_arr = zeros([n_coeffs, max_poly_order]); %15x14
poly_vals = zeros([length(r), max_poly_order]); %5000x14

for poly_order_ind = 1:max_poly_order
    beta = ols_coeffs_data(r_norm, y, poly_order_ind).beta;
    poly_coeffs = flipud(beta);
    for r_ind = 1:length(poly_coeffs)
        coeffs_arr(r_ind, poly_order_ind) = poly_coeffs(r_ind); %Array is in DESCENDING ORDER of poly coeffs
    end
    poly_vals(:, poly_order_ind) = polyval(poly_coeffs, r_norm);
end

data_poly_vals_arr = [r, y, poly_vals];

data_var_names = {'r', 'y'};
poly_var_nums = 1:length(poly_vals(1,:));
poly_var_names = "p=" + poly_var_nums;
var_names = {[data_var_names, poly_var_names]};

data_poly_vals_table = array2table(data_poly_vals_arr,...
    'VariableNames',var_names{1});

model_data_struct.coeffs_arr = coeffs_arr;
model_data_struct.ry_polyvals_arr = data_poly_vals_arr;
model_data_struct.ry_polyvals_table = data_poly_vals_table;

end

function binned_data_struct = bin_this_data(data_arr_to_bin, k_bins)
binned_data_struct = struct();

num_data_pts = size(data_arr_to_bin, 1);
rows_per_bin = floor(num_data_pts / k_bins);
extra_rows_needed = mod(num_data_pts, k_bins);

perm_ind = randperm(num_data_pts);
perm_data = data_arr_to_bin(perm_ind, :);
```

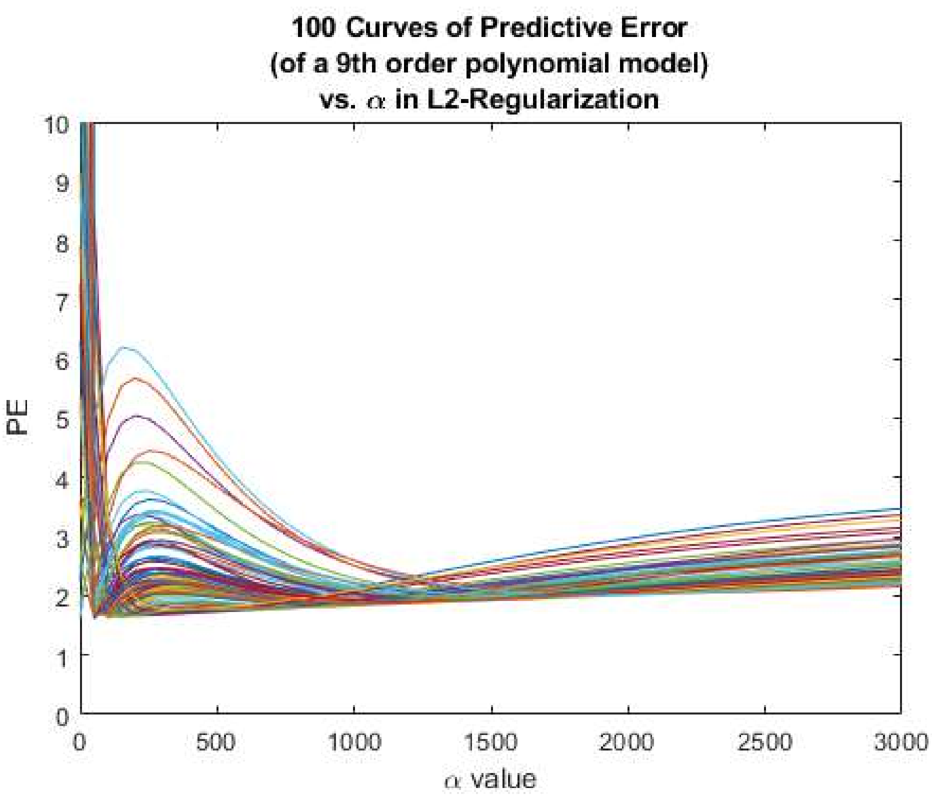
```
rowDist = rows_per_bin * ones(1, k_bins);

for extra_row = 1:extra_rows_needed
    rowDist(extra_row) = rowDist(extra_row) + 1;
end

binned_data_cell = mat2cell(perm_data, rowDist)';
bin_nums = 1:k_bins;
bin_names = "bin" + bin_nums;

binned_data_table = cell2table(binned_data_cell,...
    'VariableNames', bin_names);

binned_data_struct.cell = binned_data_cell;
binned_data_struct.table = binned_data_table;

end
```

```
ans =

    'It looks the optimal alpha is maybe ~400 ish'


ans =

    'L2 Ridge (alpha = 0) yields OLS coeffs'


coeff_table_all =

  10×3 table
```

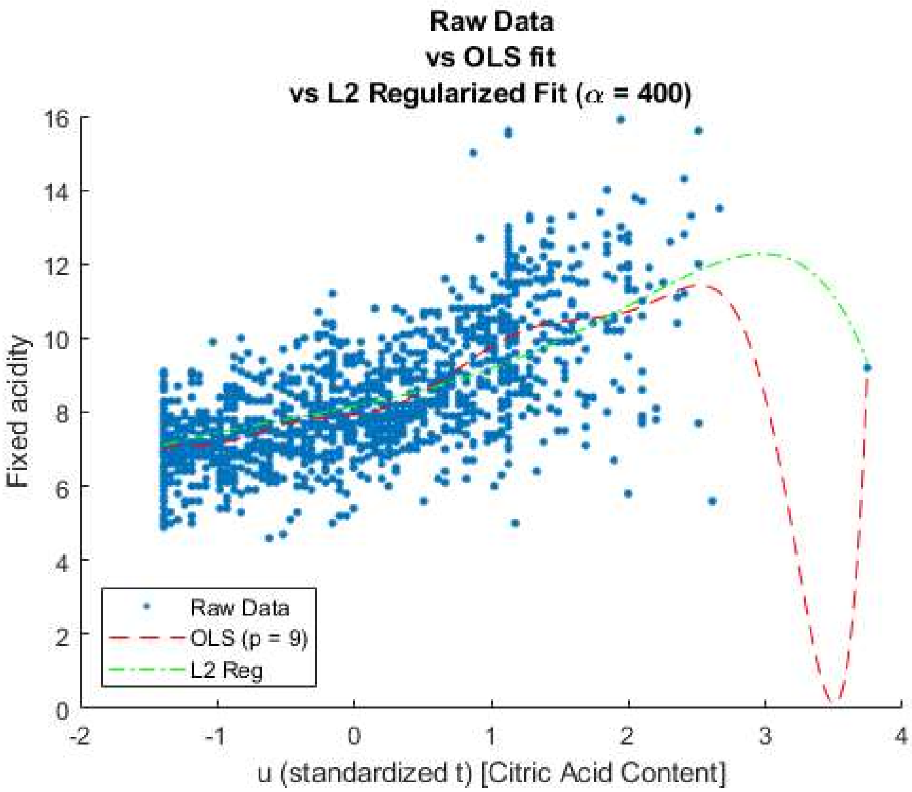
	w_ols	w_L2 (alpha = 0)	w_L2_opt (alpha = 400)
p=1	7.9409	7.9409	8.1601
p=2	0.46649	0.46649	0.77032
p=3	0.87997	0.87997	0.15954
p=4	1.9283	1.9283	0.095509
p=5	-0.78113	-0.78113	-0.0068747
p=6	-1.3615	-1.3615	-0.00094424
p=7	0.58475	0.58475	-0.00085789
p=8	0.25095	0.25095	-0.0001685
p=9	-0.16263	-0.16263	-4.0926e-05
p=10	0.021596	0.021596	-8.5678e-06



Scatter Plot Matrix of Wine Data



100 Curves of Predictive Error
(of a 9th order polynomial model)
vs. α in L2-Regularization
PE
α value



Published with MATLAB® R2020b

hw1q3

March 4, 2021

0.0.1 Liam Jackson

0.0.2 BE700 ML with Andy Fan

0.0.3 HW1q3

[1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
# %matplotlib inline
```

1. Loading Dataset

[2]:
```
data_df = pd.read_csv('data.csv')
num_cols = data_df.columns.size
print(f"There are {num_cols} columns in the raw dataframe. There is one Unnamed␣
 ↪column, the 33rd column, which contains no data.")
for ind, name in enumerate(data_df.columns):
    print(f"Column number: {ind +1}, Column name: {name}")
```

```
There are 33 columns in the raw dataframe. There is one Unnamed column, the 33rd
column, which contains no data.
Column number: 1, Column name: id
Column number: 2, Column name: diagnosis
Column number: 3, Column name: radius_mean
Column number: 4, Column name: texture_mean
Column number: 5, Column name: perimeter_mean
Column number: 6, Column name: area_mean
Column number: 7, Column name: smoothness_mean
Column number: 8, Column name: compactness_mean
Column number: 9, Column name: concavity_mean
Column number: 10, Column name: concave points_mean
Column number: 11, Column name: symmetry_mean
Column number: 12, Column name: fractal_dimension_mean
Column number: 13, Column name: radius_se
Column number: 14, Column name: texture_se
Column number: 15, Column name: perimeter_se
Column number: 16, Column name: area_se
Column number: 17, Column name: smoothness_se
```

```
Column number: 18, Column name: compactness_se
Column number: 19, Column name: concavity_se
Column number: 20, Column name: concave points_se
Column number: 21, Column name: symmetry_se
Column number: 22, Column name: fractal_dimension_se
Column number: 23, Column name: radius_worst
Column number: 24, Column name: texture_worst
Column number: 25, Column name: perimeter_worst
Column number: 26, Column name: area_worst
Column number: 27, Column name: smoothness_worst
Column number: 28, Column name: compactness_worst
Column number: 29, Column name: concavity_worst
Column number: 30, Column name: concave points_worst
Column number: 31, Column name: symmetry_worst
Column number: 32, Column name: fractal_dimension_worst
Column number: 33, Column name: Unnamed: 32
```

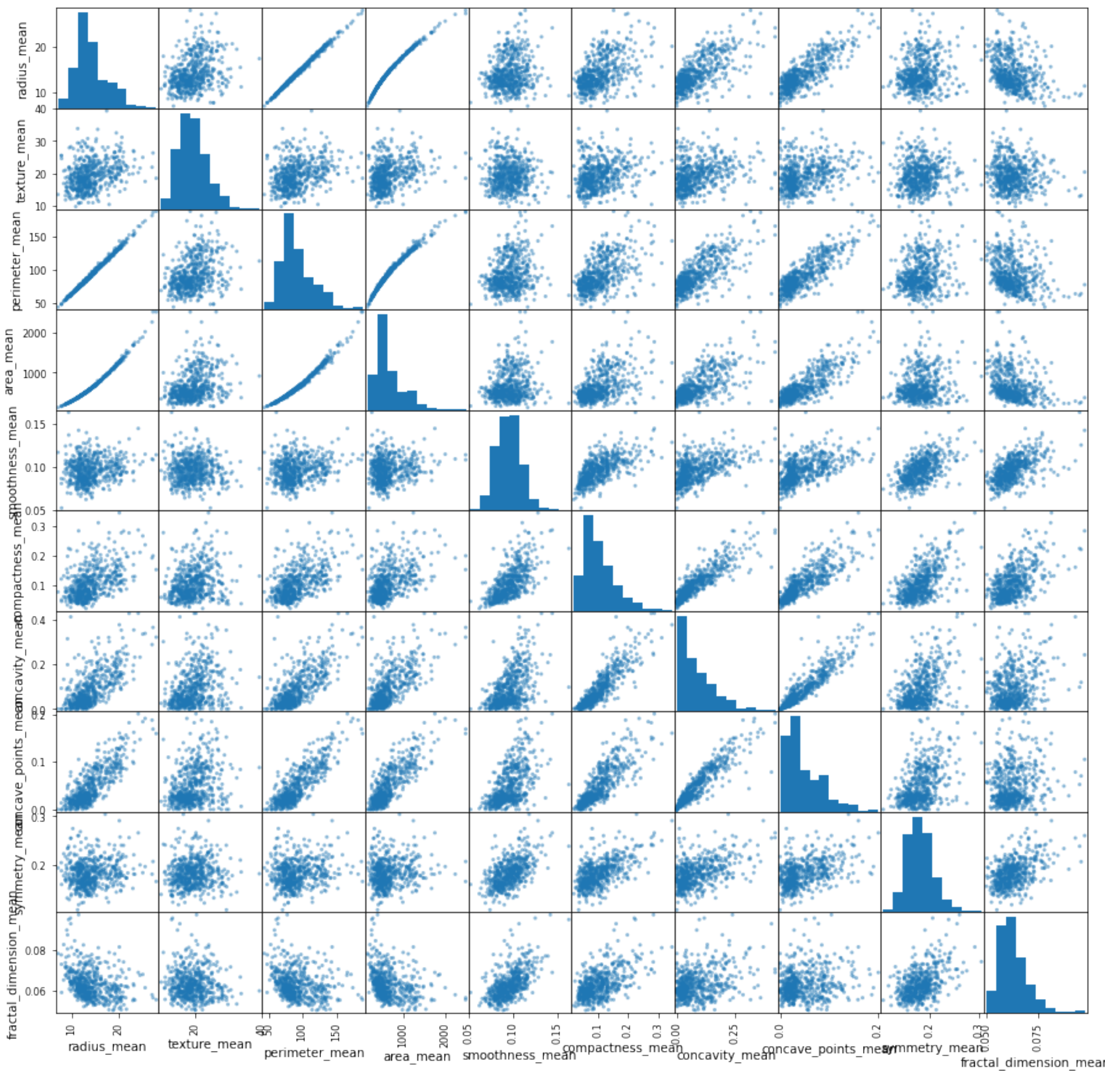
2. Generating a matrix scatter plot

[3]:
```
mean_cols = [mean_col for mean_col in data_df.columns if 'mean' in mean_col]
mean_cols = data_df.filter(regex = 'mean').columns
print(f"Just verifying there are {len(mean_cols)} columns with 'mean' in the␣
↪column name")
```

```
Just verifying there are 10 columns with 'mean' in the column name
```

[4]:
```
means_df = data_df.filter(items = mean_cols)
means_df.columns = means_df.columns.str.replace(' ', '_')
```

[5]:
```
mat_plot = pd.plotting.scatter_matrix(means_df, figsize = (15, 15))
```



3. Calculations/Statistics

[6]:
```python
num_ben = data_df['diagnosis'].value_counts()['B']
num_mal = data_df['diagnosis'].value_counts()['M']
print(f"There are {num_ben} Benign occurrences and {num_mal} Malignant occurrences.")
```

```
There are 357 Benign occurrences and 212 Malignant occurrences.
```

[7]:
```python
stats_df = means_df.describe().loc[["mean", "std"], :]
print("Statistics for both diagnoses: ")
stats_df
```

```
Statistics for both diagnoses:
```

[7]:
```
      radius_mean  texture_mean  perimeter_mean   area_mean  smoothness_mean  \
mean    14.127292     19.289649       91.969033  654.889104         0.096360
std      3.524049      4.301036       24.298981  351.914129         0.014064

      compactness_mean  concavity_mean  concave_points_mean  symmetry_mean  \
mean          0.104341        0.088799             0.048919       0.181162
std           0.052813        0.079720             0.038803       0.027414

      fractal_dimension_mean
mean                0.062798
std                 0.007060
```

[8]:
```python
ben_df = data_df[data_df['diagnosis'] == 'B'].filter(items = mean_cols)
ben_stats_df = ben_df.describe().loc[["mean", "std"], :]
print("Benign Occurrences Statistics: ")
ben_stats_df
```

```
Benign Occurrences Statistics:
```

[8]:
```
      radius_mean  texture_mean  perimeter_mean   area_mean  smoothness_mean  \
mean    12.146524     17.914762       78.075406  462.790196         0.092478
std      1.780512      3.995125       11.807438  134.287118         0.013446

      compactness_mean  concavity_mean  concave points_mean  symmetry_mean  \
mean          0.080085        0.046058             0.025717       0.174186
std           0.033750        0.043442             0.015909       0.024807

      fractal_dimension_mean
mean                0.062867
std                 0.006747
```

[9]:
```python
mal_df = data_df[data_df['diagnosis'] == 'M'].filter(items = mean_cols)
mal_stats_df = mal_df.describe().loc[["mean", "std"], :]
print("Malignant Occurrences Statistics: ")
mal_stats_df
```

```
Malignant Occurrences Statistics:
```

[9]:
```
      radius_mean  texture_mean  perimeter_mean   area_mean  smoothness_mean  \
mean    17.462830     21.604906      115.365377  978.376415         0.102898
std      3.203971      3.779470       21.854653  367.937978         0.012608

      compactness_mean  concavity_mean  concave points_mean  symmetry_mean  \
mean          0.145188        0.160775             0.087990       0.192909
std           0.053987        0.075019             0.034374       0.027638

      fractal_dimension_mean
```

```
mean                 0.062680
std                  0.007573
```

[10]:
```
num_ben_rad15 = ben_df[ben_df['radius_mean'] >= 15].shape[0]
per_ben_rad15 = round(100 * (num_ben_rad15 / num_ben), 2)
print(f"{per_ben_rad15} % of Benign occurrences have a cell radius of at least␣
 ↪15")
```

```
3.64 % of Benign occurrences have a cell radius of at least 15
```

4. Building OLS Model to predict area (y) given radius (x)

[11]:
```
xy_df = data_df[['radius_mean', 'area_mean']].sort_values('radius_mean')
xy_df
```

[11]:
```
     radius_mean  area_mean
101        6.981      143.5
539        7.691      170.4
538        7.729      178.8
568        7.760      181.0
46         8.196      201.9
..           …          …
82        25.220     1878.0
352       25.730     2010.0
180       27.220     2250.0
461       27.420     2501.0
212       28.110     2499.0

[569 rows x 2 columns]
```

[12]:
```
max_poly_ord = 2
poly_ord_range = list(range(1, max_poly_ord + 1))

ols_all_models_dict = {}

for model_ord in poly_ord_range:
    model_dict = {}
    model_df = xy_df.copy()
    x = model_df['radius_mean']
    y_r = model_df['area_mean']

    X = np.zeros([len(x), model_ord + 1])

    for X_col in list(range(0, X.shape[1])):
        X[:,X_col] = x ** X_col
    Xt = np.transpose(X)
    XtX = np.matmul(Xt, X)
    XtXinv = np.linalg.inv(XtX)
```

```python
    XtXinvXt = np.matmul(XtXinv, Xt)
    beta = np.matmul(XtXinvXt, y_r)
    beta_flip = np.flipud(beta)

    y_m = np.polyval(beta_flip, x)
    res = y_r.subtract(y_m)
    res_sq = abs(res) ** 2

    model_df['y_m'] = y_m
    model_df['res'] = res
    model_df['res_sq'] = res_sq

    res_sq_sum = sum(res_sq)
    MSE = res_sq_sum / len(x)

    model_dict['beta_flip'] = beta_flip
    model_dict['model_df'] = model_df
    model_dict['res_sq_sum'] = res_sq_sum
    model_dict['MSE'] = MSE

    model_key = "p = " + str(model_ord)
    ols_all_models_dict[model_key] = model_dict
```

[13]:
```python
ols_all_models_dict
```

[13]:
```
{'p = 1': {'beta_flip': array([  98.59821922, -738.0367042 ]),
  'model_df':      radius_mean  area_mean          y_m         res
res_sq
  101        6.981      143.5   -49.722536  193.222536   37334.948362
  539        7.691      170.4    20.282200  150.117800   22535.353941
  538        7.729      178.8    24.028932  154.771068   23954.083453
  568        7.760      181.0    27.085477  153.914523   23689.680417
  46         8.196      201.9    70.074300  131.825700   17378.015051
  ..           …          …          …          …            …
  82        25.220     1878.0  1748.610384  129.389616   16741.672622
  352       25.730     2010.0  1798.895476  211.104524   44565.119965
  180       27.220     2250.0  1945.806823  304.193177   92533.489030
  461       27.420     2501.0  1965.526467  535.473533  286731.904882
  212       28.110     2499.0  2033.559238  465.440762  216635.102985

  [569 rows x 5 columns],
  'res_sq_sum': 1767428.9562542248,
  'MSE': 3106.2020320812385},
 'p = 2': {'beta_flip': array([  3.10992516,   0.43684601, -10.5164038 ]),
  'model_df':      radius_mean  area_mean          y_m         res        res_sq
  101        6.981      143.5   144.093434   -0.593434      0.352164
```

```
539          7.691      170.4   176.800058   -6.400058     40.960745
538          7.729      178.8   178.638950    0.161050      0.025937
568          7.760      181.0   180.145751    0.854249      0.729742
46           8.196      201.9   201.971393   -0.071393      0.005097
..             …          …          …           …            …
82          25.220     1878.0  1978.563778 -100.563778  10113.073432
352         25.730     2010.0  2059.596420  -49.596420   2459.804862
180         27.220     2250.0  2305.606421  -55.606421   3092.074085
461         27.420     2501.0  2339.679053  161.320947  26024.448049
212         28.110     2499.0  2459.139436   39.860564   1588.864558

[569 rows x 5 columns],
'res_sq_sum': 123097.70230710595,
'MSE': 216.34042584728638}}
```

[14]:
```python
print(f"The Linear (p = 1) model coefficients are: {ols_all_models_dict['p =␣
↪1']['beta_flip']}")
print("The Linear (p = 1) model residuals are: ")
print(ols_all_models_dict['p = 1']['model_df']['res'])
```

```
The Linear (p = 1) model coefficients are: [  98.59821922 -738.0367042 ]
The Linear (p = 1) model residuals are:
101    193.222536
539    150.117800
538    154.771068
568    153.914523
46     131.825700
          …
82     129.389616
352    211.104524
180    304.193177
461    535.473533
212    465.440762
Name: res, Length: 569, dtype: float64
```

[15]:
```python
print(f"The Quadratic (p = 2) model coefficients are: {ols_all_models_dict['p =␣
↪2']['beta_flip']}")
print(f"The Quadratic (p = 2) model residuals are: ")
print(ols_all_models_dict['p = 2']['model_df']['res'])
```

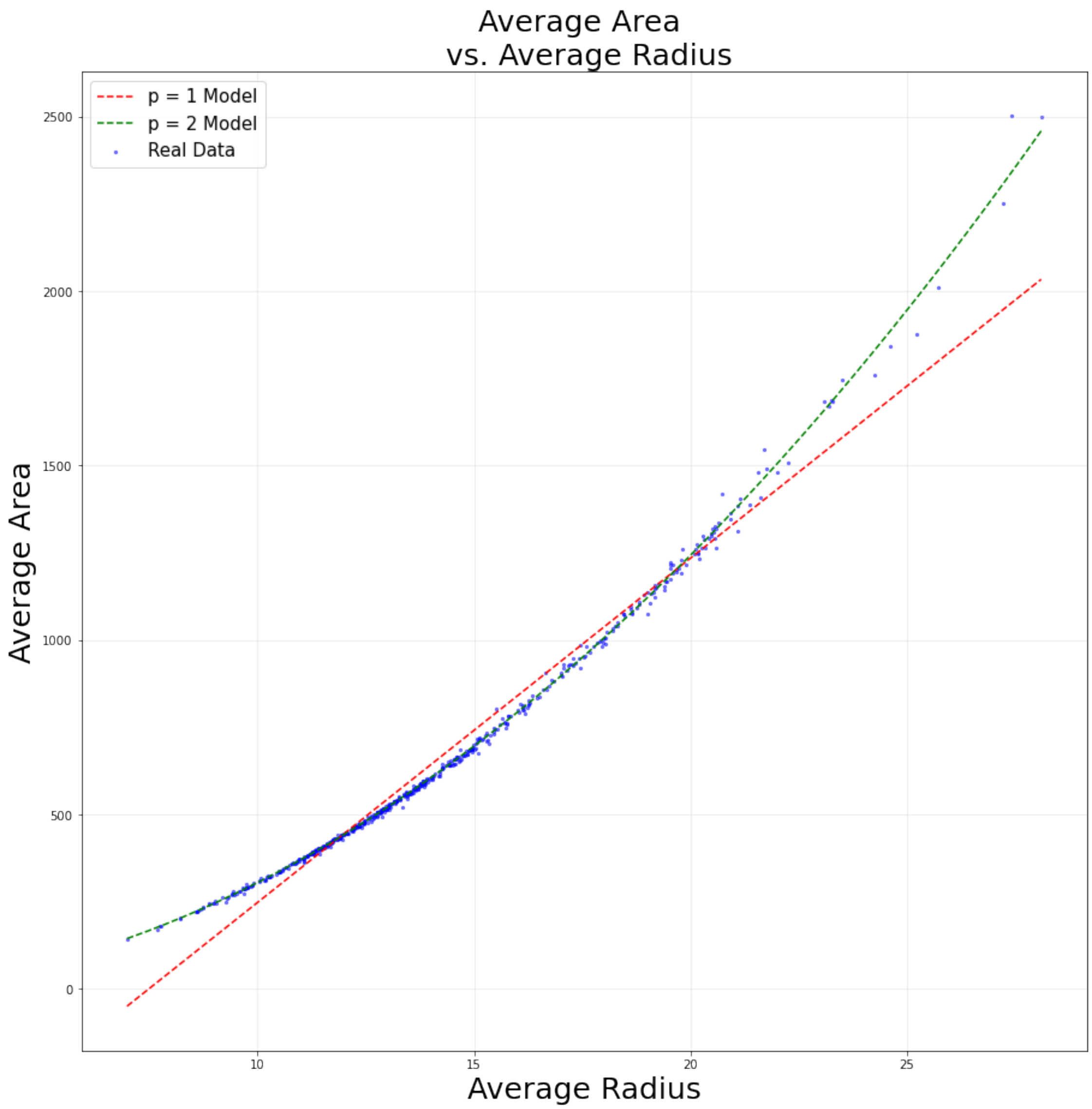
```
The Quadratic (p = 2) model coefficients are: [  3.10992516   0.43684601
-10.5164038 ]
The Quadratic (p = 2) model residuals are:
101     -0.593434
539     -6.400058
538      0.161050
568      0.854249
```

```
46      -0.071393
          …
82    -100.563778
352    -49.596420
180    -55.606421
461    161.320947
212     39.860564
Name: res, Length: 569, dtype: float64
```

5. Plotting Data vs. Polynomial Models

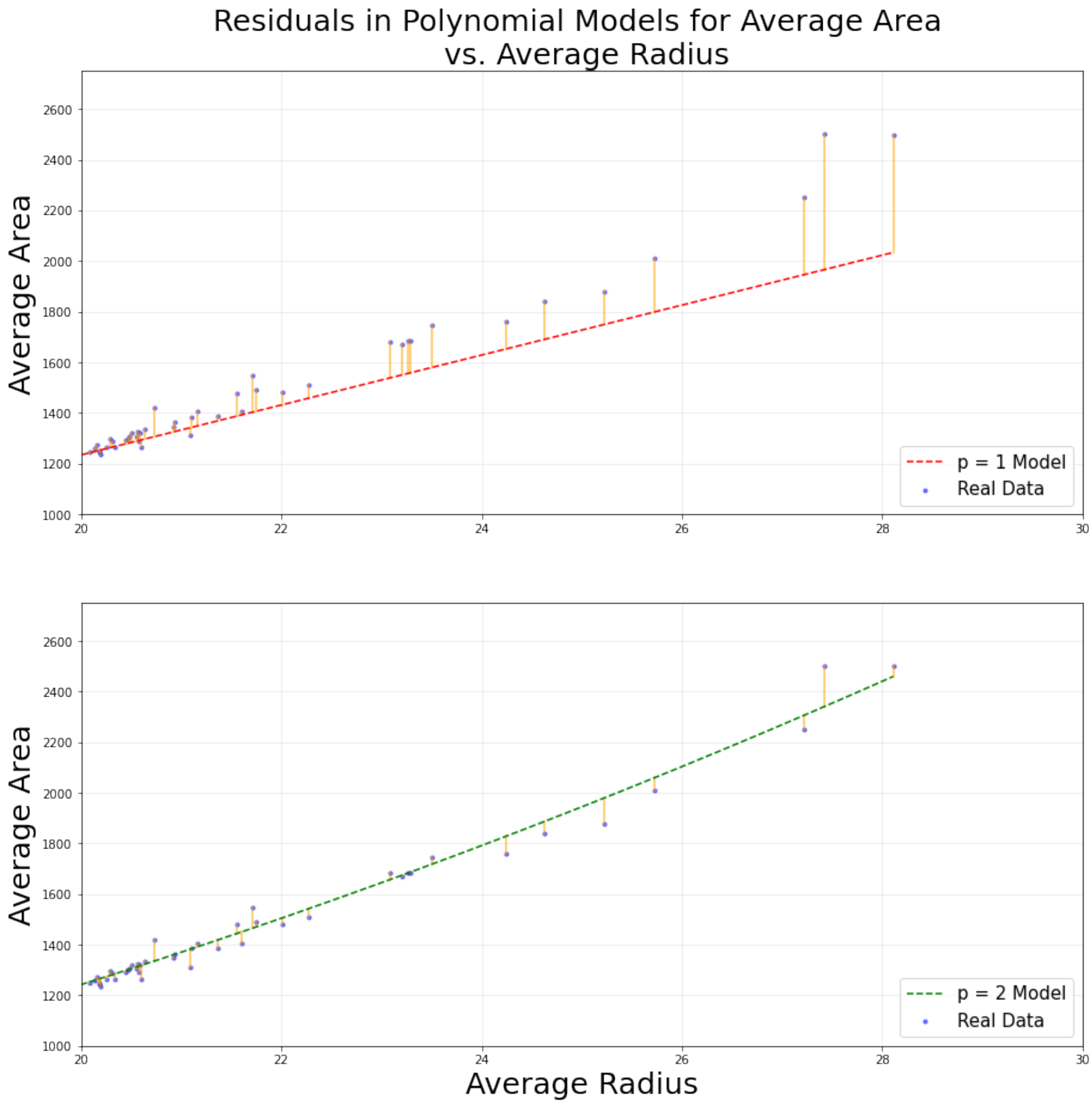
[16]:
```
p1_coeffs = ols_all_models_dict['p = 1']['beta_flip']
p2_coeffs = ols_all_models_dict['p = 2']['beta_flip']

p1_data_df = ols_all_models_dict['p = 1']['model_df']
p2_data_df = ols_all_models_dict['p = 2']['model_df']

x_r = p1_data_df['radius_mean']
y_r = p1_data_df['area_mean']
p1_y_m = p1_data_df['y_m']
p2_y_m = p2_data_df['y_m']

x_m = np.linspace(min(x_r), max(x_r))
p1_y_m_linspace = np.polyval(p1_coeffs, x_m)
p2_y_m_linspace = np.polyval(p2_coeffs, x_m)
```

[17]:
```
plt.figure(figsize = (15, 15))

plt.scatter(x_r, y_r, s=5, c='b', alpha=0.5, label = 'Real Data')
plt.plot(x_m, p1_y_m_linspace, 'r--', label = 'p = 1 Model')
plt.plot(x_m, p2_y_m_linspace, 'g--', label = 'p = 2 Model')
plt.grid(alpha = .25)

plt.title('Average Area \n vs. Average Radius', fontsize = 25)
plt.legend(fontsize = 15)
plt.xlabel('Average Radius', fontsize = 25)
plt.ylabel('Average Area', fontsize = 25)
plt.show()
```



[18]:
```
plt.figure(figsize = (15, 15))

plt.subplot(2, 1, 1)

for i in range(len(x_r)):
        p1_res_x = (x_r[i], x_r[i])
        p1_res_y = (y_r[i], p1_y_m[i])
        plt.plot(p1_res_x, p1_res_y, color = 'orange', linewidth = 2, alpha = 0.
 ↪5)

plt.scatter(x_r, y_r, s=10, c='b', alpha=0.5, label = 'Real Data')
plt.plot(x_m, p1_y_m_linspace, 'r--', label = 'p = 1 Model')
```

```
plt.grid(alpha = .25)
plt.title('Residuals in Polynomial Models for Average Area \n vs. Average␣
↪Radius', fontsize = 25)
plt.legend(fontsize = 15, loc = 'lower right')
plt.ylabel('Average Area', fontsize = 25)
plt.xlim([20, 30])
plt.ylim([1000, 2750])

plt.subplot(2, 1, 2)

for i in range(len(x_r)):
        p2_res_x = (x_r[i], x_r[i])
        p2_res_y = (y_r[i], p2_y_m[i])
        plt.plot(p2_res_x, p2_res_y, color = 'orange', linewidth = 2, alpha = 0.
↪5)

plt.scatter(x_r, y_r, s=10, c='b', alpha=0.5, label = 'Real Data')
plt.plot(x_m, p2_y_m_linspace, 'g--', label = 'p = 2 Model')

plt.grid(alpha = .25)
plt.legend(fontsize = 15, loc = 'lower right')
plt.xlabel('Average Radius', fontsize = 25)
plt.ylabel('Average Area', fontsize = 25)
plt.xlim([20, 30])
plt.ylim([1000, 2750])

plt.show()
```



Residuals in Polynomial Models for Average Area
vs. Average Radius
Average Area
p = 1 Model
Real Data
p = 2 Model
Real Data
Average Radius