regression_tables_and_graphs

July 6, 2021

1 Python Tutorial Program: Creating Regression Tables and Line Graphs using Pandas and Matplotlib

By Kenneth Burchfiel

For more background on this program, please see my correspodning blog post here: [link forthcoming]

This program demonstrates how Python can be used to determine the statistical significance of an intervention via a Difference-in Differences analysis, then display the effects of that intervention through a series of graphs.

It performs the following tasks:

1. Import a table containing **fictional** beginning-of-year and end-of-year test score data for students in multiple high schools and grades
2. Create individual Pandas DataFrames for each school
3. Perform a regression for each school-grade pair that incorporates 3 variable: (1) time (beginning of year vs. end of year); (2) intervention group; and (3) time-intervention group interaction
4. Append these results to a table for easier comparison/analysis
5. Use this regression data to create and save charts that compare test score improvements for two groups of students
6. Create additional charts that visualize certain regression coefficients and indicate statistical significance
7. Output the table of regression results to an .xlsx worksheet

1.1 Scenario:

(Note: both the scenario and the data presented in this program are fictional.)

Orange Valley School District (OVSD) is piloting an after-school math program called Code the Concepts in each of its 8 high schools. This program aims to teach students relevant math skills through programming and computer science applications, and is available in grades 9-12.

Although the district believes that the programming knowledge taught in this course is highly valuable in itself, it also wants to evaluate the impact, if any, this program has on students' math skills. The means of evaluating the math skills will be the Higher Grades Math Assessment (HGMA), a test administered at the beginning and end of each school year.

In an earlier program, I showed how Python could be used to turn a dataset with HGMA scores into pivot tables and charts. Meanwhile, in this program, I will perform regression analyses of this data to determine (1) whether the increase in scores over time for students enrolled in Code the Concepts was significantly higher than the increase for non-enrolled students. These regressions will be followed by line and bar graphs that help visualize the output of the regression analysis.

1.2 An explanation of using regressions to run difference-in-differences analyses

If you are already familiar with the use of regression analyses to perform difference-in-difference analyses, feel free to skip over the following text block. If not, please consider reading it, as it will help explain my choices in coding the regression analyses within the program.

Note: A webpage titled "Difference-in-Difference Estimation" (accessible at publichealth.columbia.edu) was a valuable reference in writing the following summary. It also discusses precautions that I skip over here.

One means of determining whether an intervention had a significant effect is to collect an outcome measure at two different points of time for two different groups: the experimental group (who received the intervention) and the control group (who did not). The magnitude of the experimental group's change can then be compared with the control group's change, producing what is known as a difference-in-differences (or difference-in-difference) analysis.

For example, in this tutorial program, the difference-in-differences analysis will focus on beginning-of-year and end-of-year test scores for (1) students who were enrolled in an optional after-school math class and (2) students who were not. Suppose that the students enrolled in the optional program produced a 10-point score improvement during the school year. This may tempt you to conclude that the optional program was a great success. However, if students who weren't enrolled in the program *also* had a 10-point score improvement, it would seem more likely that other factors (such as students' regular within-school math instruction), rather than the optional program, were responsible for the growth in math scores.

Suppose instead that the enrolled students' scores increased by 15 points, compared to 13 points for non-enrolled students. In this case, enrolled' students scores increased 2 points higher on average than did non-enrolled students' scores. Is this increase statisticaly significant, or is the chance that the difference was a result of random variation too high?

One way to determine statistical significance is to run a linear regression analysis. Three independent variables (in addition to a constant value) need to be included: (1) time (beginning of year vs. end of year); (2) enrollment status; and (3) an interaction variable (in this case, one specifying whether the data point is from an enrolled student at the end of the year).

The p value of the time coefficient will help determine whether the growth (or decline) in scores across time was statistically significant for the student body as a whole. Meanwhile, the p value of the interaction coefficient provides insight into the intervention's significance. A p value below 0.05 would suggest that students enrolled in the optional program had a significantly larger increase (or decrease) in scores than did non-enrolled students.

With this explanation out of the way, I'll now turn to discussing the actual program.

I will start by importing a number of useful items:

[2]:
```
import time
start_time = time.time() # Allows the program's runtime to be measured
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.ticker as mtick
from adjustText import adjust_text
import statsmodels.api as sm
pd.set_option('display.max_rows',100)
pd.set_option('display.max_columns',100)
```

Next, I will import the beginning-of-year and end-of-year HGMA scores for all OVSD high schoolers into a Pandas DataFrame. (This fictional score data was generated within an earlier tutorial program.)

[3]:
```
df_scores = pd.read_excel('scores_by_program_enrollment_copy.xlsx')
df_scores.drop('Student_ID',axis=1,inplace=True)
df_scores
```

[3]:
```
        School  Grade Enrolled    Time  Score  Count
0     Bayville     12      Yes  2018_9     39      1
1     Bayville     12      Yes  2019_5     54      1
2     Bayville     12       No  2018_9     77      1
3     Bayville     12       No  2019_5     75      1
4        Eagle     12       No  2018_9     35      1
…          …     …        …      …      …      …
7675  Cardinal     11       No  2018_9     59      1
7676     Eagle     12      Yes  2019_5     52      1
7677     Eagle     12      Yes  2018_9     33      1
7678  Westwood     10      Yes  2018_9     66      1
7679  Westwood     10      Yes  2019_5     81      1

[7680 rows x 6 columns]
```

Next, I will create a list of all schools within the DataFrame, which will make it easier to produce an individual DataFrame for each school.

[4]:
```
school_list = list(pd.unique(df_scores['School'])) # This list will be looped␣
↪through in order to create DataFrames for each school
school_list.append('Total')
print(school_list)
```

```
['Bayville', 'Eagle', 'Westwood', 'Fair Lake', 'Central', 'East River', 'Olive',
'Cardinal', 'Total']
```

The following function takes a DataFrame with score data for an individual school, then converts it into a form that will make later regression analyses easier to perform.

[5]:
```
def create_data_source_for_regression(school, df): # The 'school' variable is␣
↪not needed for the function to run, but is included so that the DataFrame␣
↪output can be identified as belonging to a particular school.
    if school == 'Total': # DataFrame will encompass all students, so no need␣
↪to incorporate a query() statement
        df_source = df[['Grade', 'Enrolled', 'Time', 'Score']].copy() # The␣
↪'School' column is not necessary since this DataFrame will encompass all␣
↪schools as a whole
    else:
        df_source = df.query("School == @school")[['Grade', 'Enrolled', 'Time',␣
↪'Score']].copy()
    df_source = pd.get_dummies(df_source.copy(),drop_first=True) # Even if␣
↪'School' had been included as one of the columns earlier, this line would␣
↪end up removing the 'School' column for cases where only one school was␣
↪being queried. This is due to the fact that, with drop_first enabled, the␣
↪first dummy variable for School (School_[school_name]) will get removed. In␣
↪this case, School_[school_name] is the only dummy variable (since all␣
↪students attend that school), so no school column will show up in the final␣
↪output. This is fine, since this column does not provide any information (as␣
↪we already know each student in this DataFrame attends that school.)
    df_source.rename(columns={"Time_2019_5":"Time_EOY"},inplace=True)
    df_source['EOY_and_Enrolled'] = 0
    for i in range(len(df_source)):
        if (df_source.loc[df_source.index[i], 'Time_EOY'] == 1) and (df_source.
↪loc[df_source.index[i], 'Enrolled_Yes'] == 1):
            df_source.loc[df_source.index[i], 'EOY_and_Enrolled'] = 1
    return school, df_source # Stores the school along with the DataFrame in a␣
↪tuple so that DataFrames can be matched to schools more easily later on
```

[6]:
```
school_df_pair_list = [] # Stores tuple outputs of␣
↪create_data_source_for_regression for each school and for the district as a␣
↪whole
for school in school_list:
    school_df_pair_list.append(create_data_source_for_regression(school,␣
↪df_scores)) # The output, like the input, is a tuple
```

2 Here with editing–note how dummy variables, including one for the interaction, were created below

[21]:
```
school_df_pair_list[0][0]
```

[21]: 'Bayville'

[7]:
```
school_df_pair_list[0][1]
```

[7]:
```
       Grade  Score  Enrolled_Yes  Time_EOY  EOY_and_Enrolled
0         12     39             1         0                 0
1         12     54             1         1                 1
2         12     77             0         0                 0
3         12     75             0         1                 0
26         9     61             1         0                 0
…      …     …            …         …               …
7619      11     32             0         0                 0
7658       9     58             0         1                 0
7659       9     62             0         0                 0
7672      10     54             1         1                 1
7673      10     30             1         0                 0

[904 rows x 5 columns]
```

[8]:
```python
grade_test = 'Grade 9'
grade_int = int(grade_test.replace('Grade ', ''))
school_df_pair_list[0][1].query("Grade == @grade_int")['Score']
```

[8]:
```
26      61
27      69
208     36
209     38
282     43
        ..
7351    63
7358    57
7359    48
7658    58
7659    62
Name: Score, Length: 204, dtype: int64
```

[9]:
```python
def create_regression_table(school, data_source_for_regression):
    regression_table = pd.DataFrame(index=['Non_Enrolled_BOY_Mean',␣
 ↪'Non_Enrolled_EOY_Mean', 'Enrolled_BOY_Mean', 'Enrolled_EOY_Mean',␣
 ↪'Time_Coeff', 'Time_Pval', 'Time_Significant', 'EOY_and_Enrolled_Coeff',␣
 ↪'EOY_and_Enrolled_Pval', 'EOY_and_Enrolled_Significant', 'R_Squared',␣
 ↪'Adj_R_Squared'], columns = ['Grade 9', 'Grade 10', 'Grade 11', 'Grade 12',␣
 ↪'Total'])
    for i in range(len(regression_table.columns)):
        if regression_table.columns[i] == 'Total':
            filtered_df = data_source_for_regression.copy()
        else:
            grade_int = int(regression_table.columns[i].replace('Grade ', ''))
            filtered_df = data_source_for_regression.query("Grade ==␣
 ↪@grade_int")
        y = filtered_df['Score']
```

```
        x_vars = filtered_df[['Time_EOY', 'Enrolled_Yes', 'EOY_and_Enrolled']]
        x_vars = sm.add_constant(x_vars)
        model = sm.OLS(y,x_vars)
        results = model.fit()

        regression_table.loc['Time_Coeff'][regression_table.columns[i]] =␣
↪results.params['Time_EOY']
        regression_table.loc['Time_Pval'][regression_table.columns[i]] =␣
↪results.pvalues['Time_EOY']
        if results.pvalues['Time_EOY'] < 0.05:
            regression_table.loc['Time_Significant'][regression_table.
↪columns[i]] = 1
        else:
            regression_table.loc['Time_Significant'][regression_table.
↪columns[i]] = 0

        regression_table.loc['EOY_and_Enrolled_Coeff'][regression_table.
↪columns[i]] = results.params['EOY_and_Enrolled']
        regression_table.loc['EOY_and_Enrolled_Pval'][regression_table.
↪columns[i]] = results.pvalues['EOY_and_Enrolled']
        if results.pvalues['EOY_and_Enrolled'] < 0.05:
            regression_table.
↪loc['EOY_and_Enrolled_Significant'][regression_table.columns[i]] = 1
        else:
            regression_table.
↪loc['EOY_and_Enrolled_Significant'][regression_table.columns[i]] = 0

        regression_table.loc['Non_Enrolled_BOY_Mean'][regression_table.
↪columns[i]] = results.params['const'] # These methods of calculating means␣
↪by adding different regression coefficients together produced the same␣
↪results as calculating them by querying specific parts of the DataFrame and␣
↪taking the means of those parts.

        regression_table.loc['Non_Enrolled_EOY_Mean'][regression_table.
↪columns[i]] = results.params['const'] + results.params['Time_EOY']

        regression_table.loc['Enrolled_BOY_Mean'][regression_table.columns[i]]␣
↪= results.params['const'] + results.params['Enrolled_Yes']

        regression_table.loc['Enrolled_EOY_Mean'][regression_table.columns[i]]␣
↪= results.params['const'] + results.params['Time_EOY'] + results.
↪params['Enrolled_Yes'] + results.params['EOY_and_Enrolled']

        regression_table.loc['R_Squared'][regression_table.columns[i]] =␣
↪results.rsquared
```

```
        regression_table.loc['Adj_R_Squared'][regression_table.columns[i]] =␣
↪results.rsquared_adj

    return school, regression_table  # the school is included in the output to␣
↪help with identification.
```

[10]:
```
regression_table_list = []

for i in range(len(school_df_pair_list)): # school_df_pair_list contains both␣
↪school names and dataframes for those schools. These names and DataFrames␣
↪can then be used as arguments for the create_regression_table function when␣
↪it loops through regression_table_list.
    regression_table_list.
↪append(create_regression_table(school_df_pair_list[i][0],␣
↪school_df_pair_list[i][1])) # i refers to the tuple in the list; [0] refers␣
↪to the school name; and [1] refers to the DataFrame returned by␣
↪create_data_source_for_regression.
```

[11]:
```
len(regression_table_list) # Determines how many sheets need to be added to the␣
↪Google Sheet that will store the DataFrame outputs
```

[11]: 9

[12]:
```
regression_table_list[0][0]
```

[12]: 'Bayville'

[20]:
```
regression_table_list[0][1]
```

[20]:
```
                             Grade 9   Grade 10   Grade 11   Grade 12  \
Non_Enrolled_BOY_Mean          54.66  52.050847  56.785714      52.95
Non_Enrolled_EOY_Mean          58.04  53.542373       58.5  53.916667
Enrolled_BOY_Mean          56.096154  53.232143  55.157895  54.693548
Enrolled_EOY_Mean          64.038462     70.125  63.982456  70.967742
Time_Coeff                      3.38   1.491525   1.714286   0.966667
Time_Pval                   0.247258   0.590873   0.552521   0.722588
Time_Significant                   0          0          0          0
EOY_and_Enrolled_Coeff      4.562308  15.401332   7.110276  15.307527
EOY_and_Enrolled_Pval       0.264748   0.000138   0.081081    0.00008
EOY_and_Enrolled_Significant       0          1          0          1
R_Squared                   0.058384   0.197423   0.046398   0.203692
Adj_R_Squared               0.044259   0.186769   0.033512   0.193738

                                Total
Non_Enrolled_BOY_Mean       54.048889
Non_Enrolled_EOY_Mean       55.875556
Enrolled_BOY_Mean           54.770925
```

```
Enrolled_EOY_Mean              67.418502
Time_Coeff                      1.826667
Time_Pval                       0.197455
Time_Significant                       0
EOY_and_Enrolled_Coeff         10.82091
EOY_and_Enrolled_Pval                0.0
EOY_and_Enrolled_Significant           1
R_Squared                       0.117464
Adj_R_Squared                   0.114522
```

[13]:

```
def plot_line(xpoints, ypoints, color=('#000000'), line_label='_',␣
↪point_label_1 = '_', point_label_2 = '_'): # X and Y points need to be in a␣
↪list format; if they derive from a DataFrame, apply list() to them␣
↪beforehand. Each list should have only two points.
    # Matplotlib documentation notes that "Specific lines can be excluded from␣
↪the automatic legend element selection by defining a label starting with an␣
↪underscore" (https://matplotlib.org/stable/api/_as_gen/matplotlib.pyplot.
↪legend.html), which is why '_' is used as the default label.
    # print("x values:",xpoints) # for debugging
    # print("y values:",ypoints)
    plt.plot(xpoints, ypoints, linestyle='solid',label=line_label,␣
↪linewidth=2,color=color)
    plt.plot(xpoints[0], ypoints[0],'om', label=point_label_1) # Plots a data␣
↪point on the left of the line. See the 'notes' section within https://
↪matplotlib.org/stable/api/_as_gen/matplotlib.pyplot.plot.html
    plt.plot(xpoints[1], ypoints[1],'og',label=point_label_2) # Plots a data␣
↪point on the right of the line.
```

[14]:

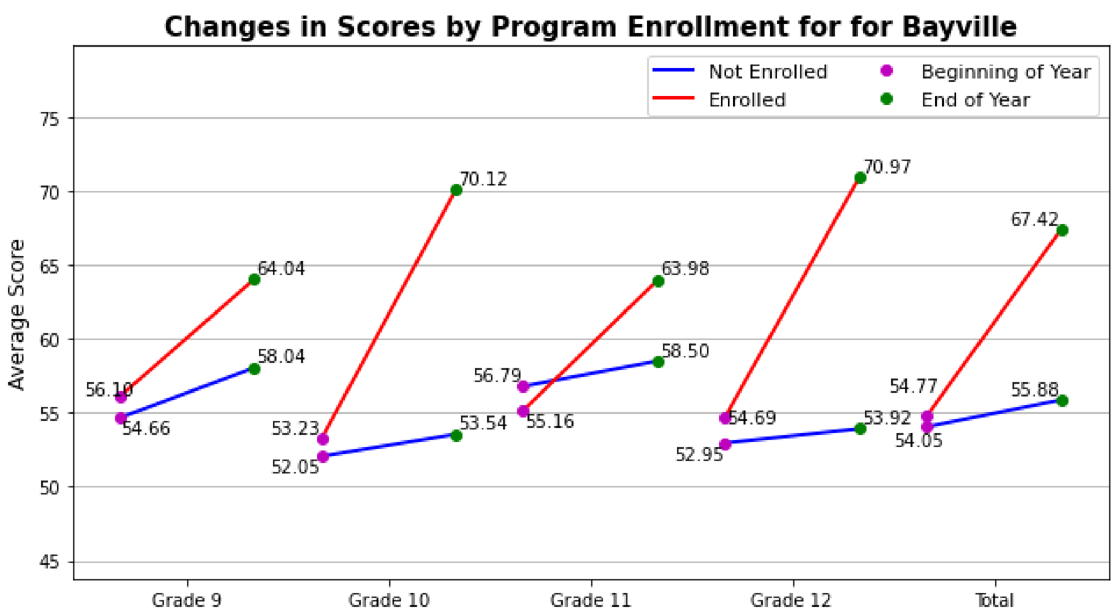
```
for t in range(len(regression_table_list)):
    t_min = min(regression_table_list[t][1].iloc[0:4,0:5].min()) # First finds␣
↪the minimum value for each column, then determines the lowest value out of␣
↪those values
    t_max = max(regression_table_list[t][1].iloc[0:4,0:5].max())
    if t == 0:
        all_schools_min = t_min
        all_schools_max = t_max
    else:
        if t_min < all_schools_min:
            all_schools_min = t_min
        if t_max > all_schools_max:
            all_schools_max = t_max

print(all_schools_min, all_schools_max)
```

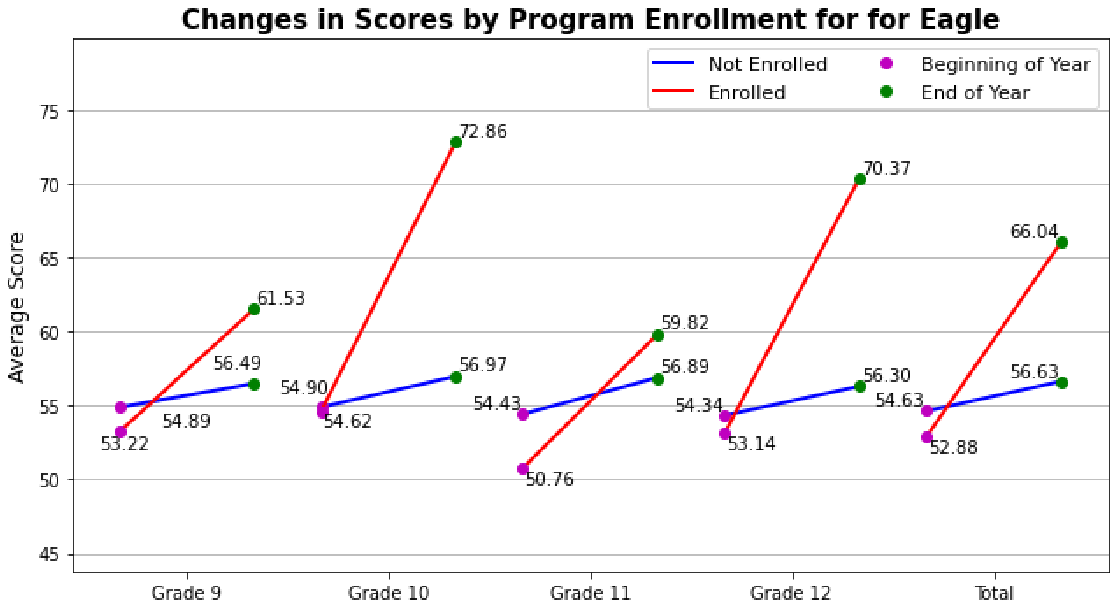
```
48.777777777777786 74.8360655737705
```

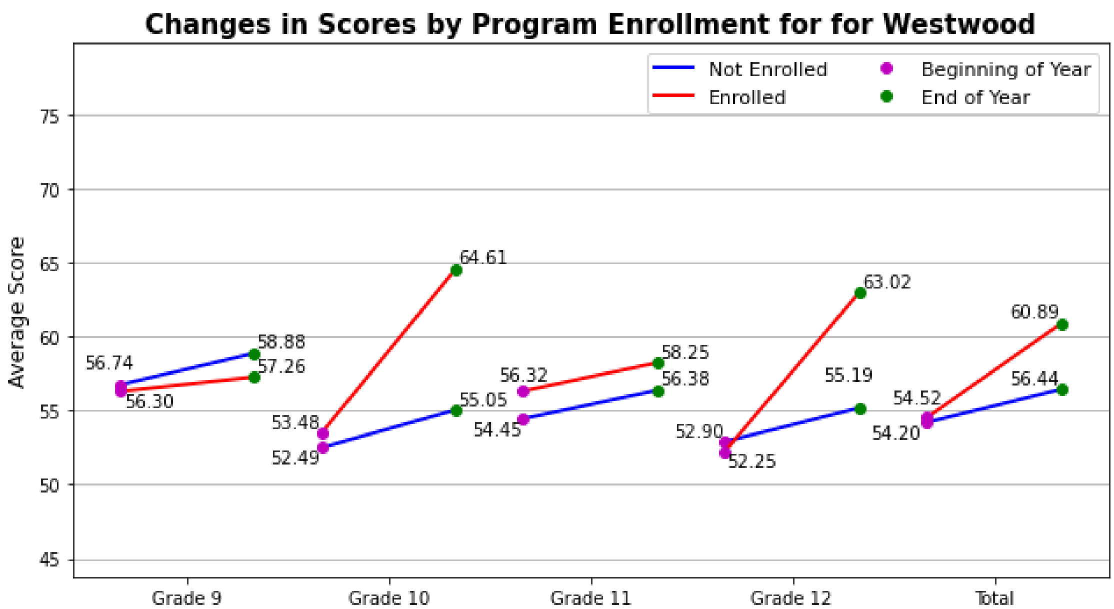
[15]:
```python
for t in range(len(regression_table_list)): # This loop runs through each database stored in the regression_table_list. i is used as an iterator below and I prefer not to use the same letter for two different iterators within a nested for loop, so I instead used t (for 'tuple') here.
    data_source = regression_table_list[t][1] # Giving this data_table a shorter variable name makes the code more readable and, more importantly, makes it easy to replace this DataFrame with a different one.
    # regression_table_list[t][2] returns a dataframe.
    school = regression_table_list[t][0]
    y_min = min(data_source.iloc[0:4,0:5].min())
    y_max = max(data_source.iloc[0:4,0:5].max())

    dl_1 = [0, 1.5, 3, 4.5, 6] # dl = data_labels. These points, along with those in dl_2, help provide correct spacing for the graphs and axis labels. Each line has a length of 1, and there is a gap of 0.5 in between each line.
    # dl_1 stores the x axis start points for each graph, whereas dl_2 stores the x axis end points.
    dl_2 = [1, 2.5, 4, 5.5, 7]

    col_labels = list(data_source.columns.copy()) # Creates a list of all the grades and the 'Total' category in the same order that they appear within the table. The list() operation is necessary in the event that changes to the  column names need to be made.
    fig, ax = plt.subplots(figsize=[9,5])
    fig.set_facecolor('white')
    plt.ylim(all_schools_min-5,all_schools_max+5)
    plt.grid(b=True,which='both',axis = 'y')
    # plt.grid(axis = 'y', which='minor', linewidth=1)

    xtick_list = []

    x_avoid_list = []
    y_avoid_list = [] # These two lists will store a series of coordinates that adjust_text will move data labels away from.
    data_label_list = []

    for i in range(len(data_source.columns)):
        if i == 0: # This if statement, along with the parameter values in the two plot_line statements below, cause only the first set of labels to be graphed, not the subsequent ones (in order to avoid duplicate labels).
            current_non_enrolled_line_label = 'Not Enrolled'
            current_enrolled_line_label = 'Enrolled'
            current_point_label_1 = 'Beginning of Year'
            current_point_label_2 = 'End of Year'
        else:
            current_non_enrolled_line_label = '_'
```

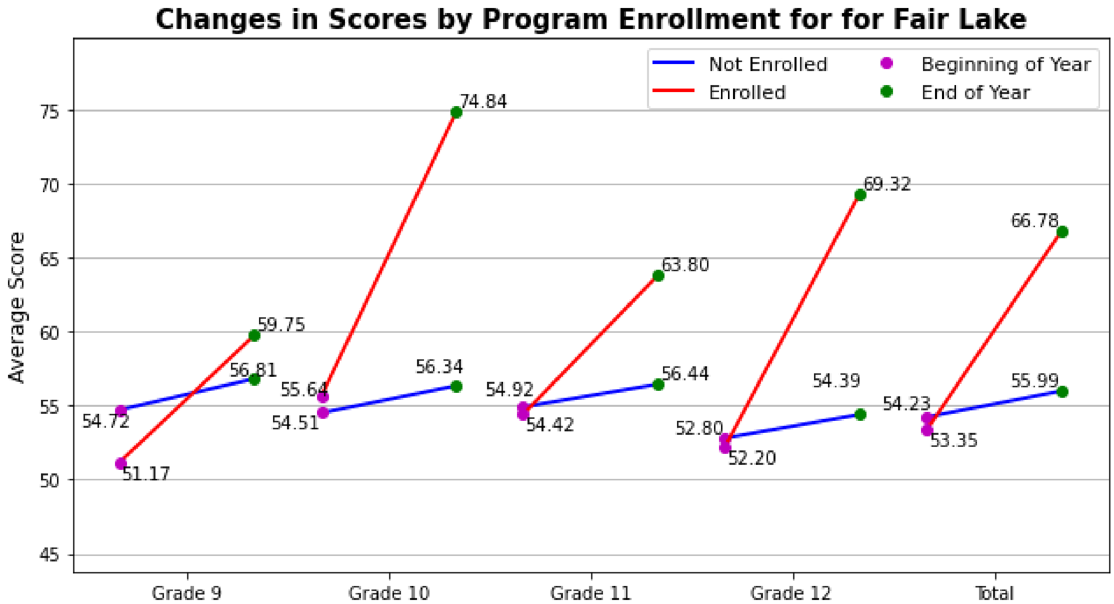
```
            current_enrolled_line_label = '_'
            current_point_label_1 = '_'
            current_point_label_2 = '_'

        non_enrolled_yvals = [data_source.
↪loc['Non_Enrolled_BOY_Mean'][data_source.columns[i]], data_source.
↪loc['Non_Enrolled_EOY_Mean'][data_source.columns[i]]]
        enrolled_yvals = [data_source.loc['Enrolled_BOY_Mean'][data_source.
↪columns[i]], data_source.loc['Enrolled_EOY_Mean'][data_source.columns[i]]]

      ␣
↪plot_line([dl_1[i],dl_2[i]],non_enrolled_yvals,color=('blue'),line_label=␣
↪current_non_enrolled_line_label)
        plot_line([dl_1[i],dl_2[i]],enrolled_yvals,color=('red'), line_label =␣
↪current_enrolled_line_label, point_label_1 = current_point_label_1,␣
↪point_label_2 = current_point_label_2)
        # Creating data labels for both sets of lines
        for j in range (2): # Loop will start with the non-enrolled students␣
↪graph and then conclude with the enrolled students graph
            if j == 0:
                yval_source = non_enrolled_yvals # Contains points for the␣
↪leftmost part of each graph
            if j == 1:
                yval_source = enrolled_yvals # Contanins points for the␣
↪rightmost part of each graph
            left_data_label = plt.text(dl_1[i],yval_source[0],'{:.2f}'.
↪format(yval_source[0]),ha='center') # Plots a percentage label next to the␣
↪left end of the line using the points in dl_1 for x values and the␣
↪percentage values from the DataFrame for y values.
            data_label_list.append(left_data_label)
            right_data_label = plt.text(dl_2[i],yval_source[1],'{:.2f}'.
↪format(yval_source[1]),ha='center')
            data_label_list.append(right_data_label)
            x_avoid = np.linspace(0,1,21) # Creates 20 points between (and␣
↪including) 0 and 1, as each line has an x length of 1
            y_avoid = yval_source[0] + x_avoid*(yval_source[1] -␣
↪yval_source[0]) # Creates a series of y values that line up with the lines␣
↪created earlier in this for loop. Each value equals the left y␣
↪value(yval_source[0]) plus the x value (from 0 to 1) * the total rise of the␣
↪line from left to right.
            x_avoid += dl_1[i] # Shifts the points in x_avoid so that they line␣
↪up with their corresponding line. (Only the first line actually starts at 0;␣
↪the other lines are offset by dl_1[i].
            x_avoid_list.extend(x_avoid)
            y_avoid_list.extend(y_avoid)
    adjust_text(data_label_list, x=x_avoid_list, y=y_avoid_list)
```

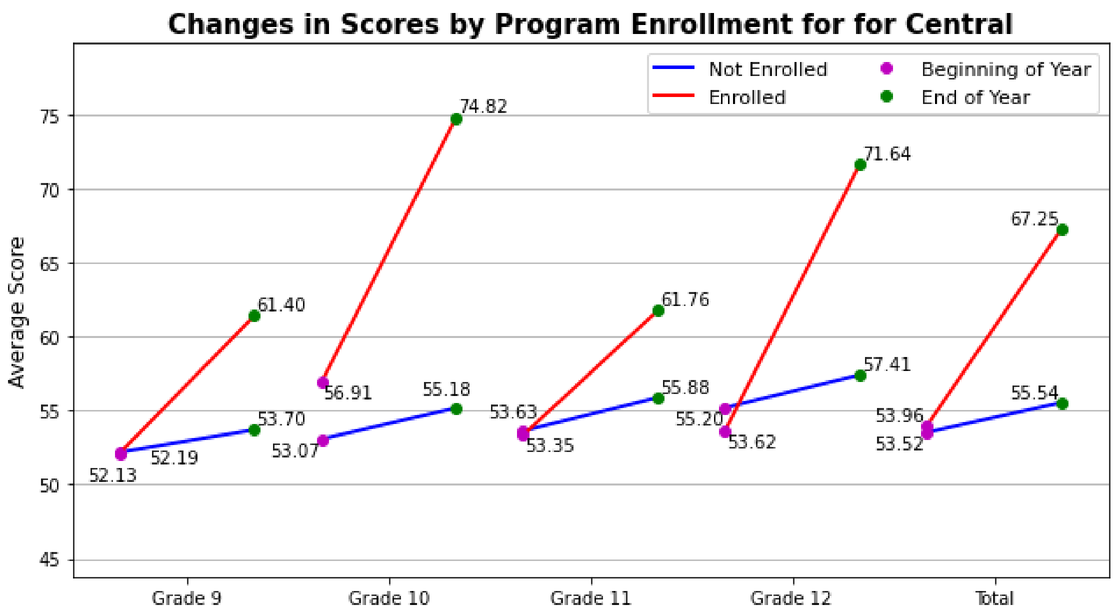
```python
    if school == 'Total':
        plt.title("Changes in Scores by Program Enrollment for All Schools",fontweight='bold',fontsize=15)
    else:
        plt.title("Changes in Scores by Program Enrollment for for "+school,fontweight='bold',fontsize=15)
    for m in range(len(dl_1)):
        xtick_list.append(np.mean([dl_1[m], dl_2[m]])) # Calculates the midpoint between each pair of dl_1 points and dl_2 points.
    ax.set_xticks(xtick_list) # These ticks are placed in the middle of each graph in order to make the graph labels align correctly.
    ax.set_xticklabels(col_labels)
    ax.set_ylabel('Average Score',fontsize=12)
    plt.tight_layout()
    plt.legend(ncol=2,fontsize=11)
    file_string = "score_graphs\\"+school +'_score_changes.png'
    plt.savefig(file_string,dpi=400)
    plt.show()
    #plt.close()
```

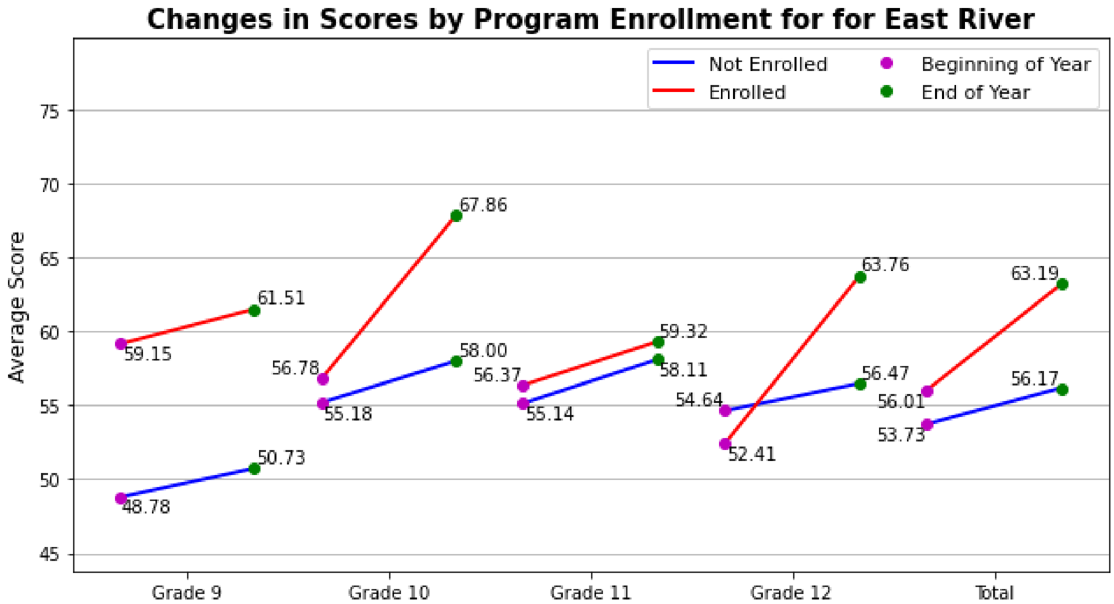



Changes in Scores by Program Enrollment for for Eagle
Not Enrolled
Enrolled
Beginning of Year
End of Year
Average Score
75
70
65
60
55
50
45
Grade 9
Grade 10
Grade 11
Grade 12
Total
61.53
56.49
54.89
53.22
72.86
56.97
54.90
54.62
59.82
56.89
54.43
50.76
70.37
56.30
54.34
53.14
66.04
56.63
54.63
52.88

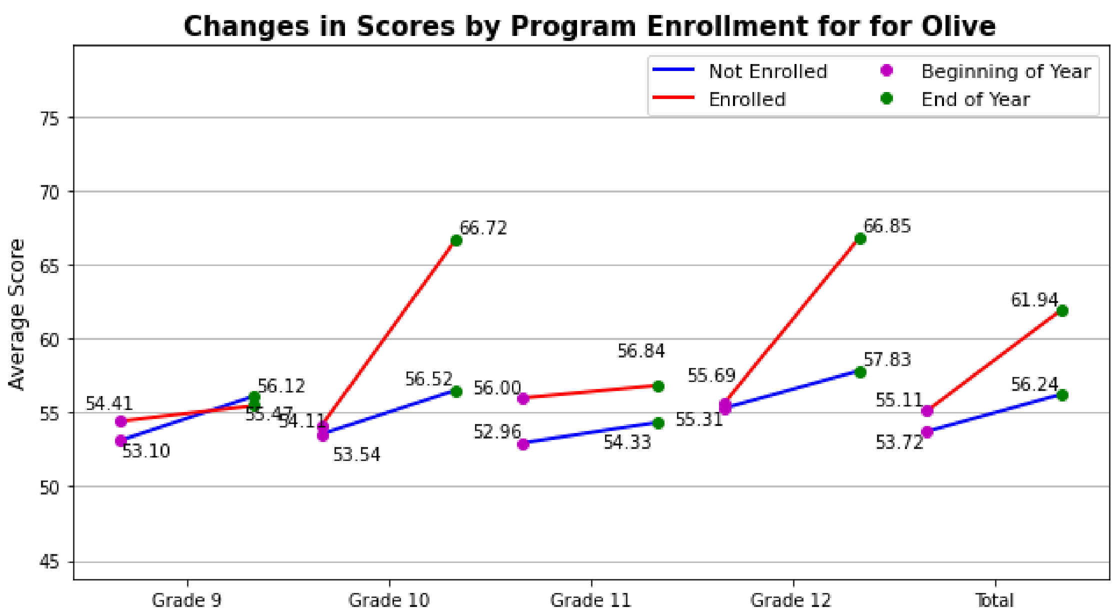

Changes in Scores by Program Enrollment for for Westwood
Not Enrolled
Enrolled
Beginning of Year
End of Year
Average Score
75
70
65
60
55
50
45
Grade 9
Grade 10
Grade 11
Grade 12
Total
56.74
58.88
57.26
56.30
64.61
55.05
53.48
52.49
56.32
58.25
56.38
54.45
63.02
55.19
52.90
52.25
60.89
56.44
54.52
54.20


Changes in Scores by Program Enrollment for for Fair Lake
Not Enrolled
Enrolled
Beginning of Year
End of Year
Average Score
75
70
65
60
55
50
45
74.84
69.32
66.78
63.80
59.75
56.81
55.64
56.34
54.92
56.44
54.39
55.99
54.72
54.51
54.42
52.80
54.23
53.35
51.17
52.20
Grade 9
Grade 10
Grade 11
Grade 12
Total

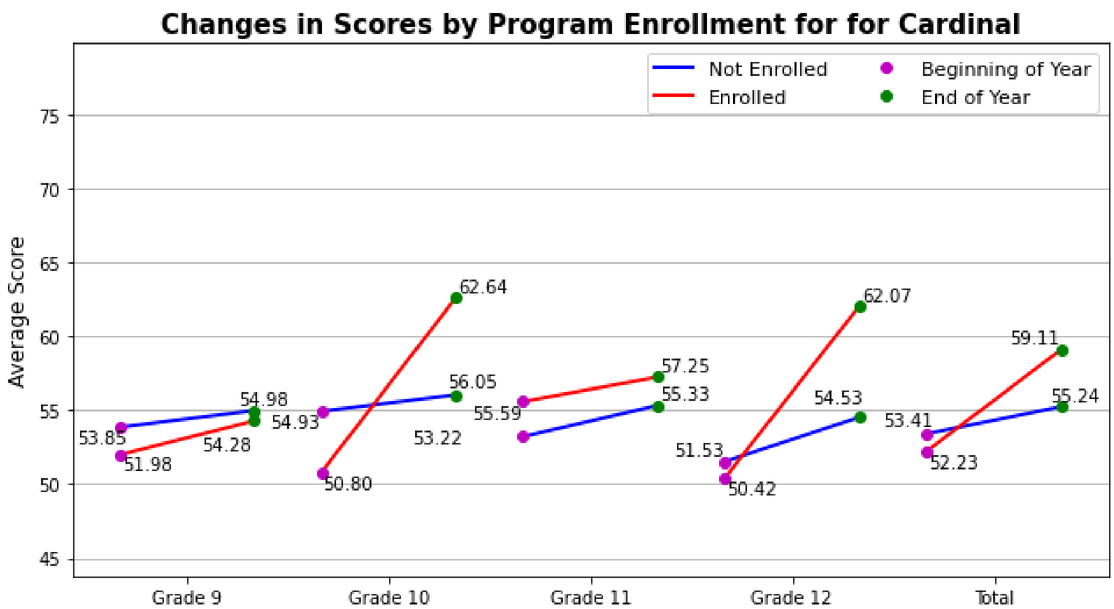

Changes in Scores by Program Enrollment for for Central
Not Enrolled
Enrolled
Beginning of Year
End of Year
Average Score
75
70
65
60
55
50
45
74.82
71.64
67.25
61.40
61.76
57.41
56.91
55.18
55.88
55.54
53.70
53.63
55.20
53.96
52.19
53.07
53.35
53.62
53.52
52.13
Grade 9
Grade 10
Grade 11
Grade 12
Total

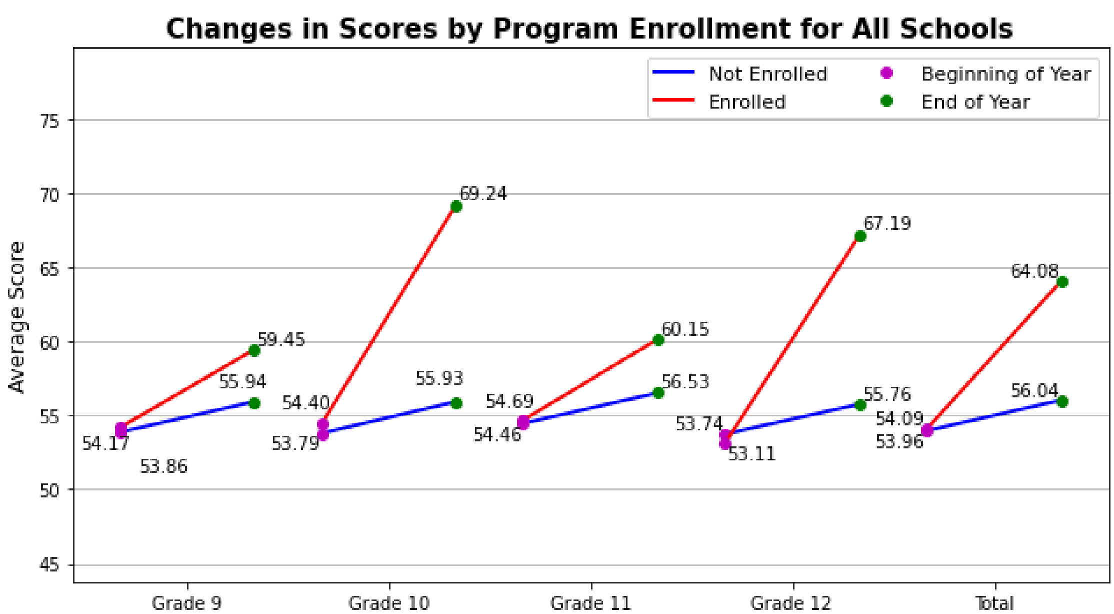


Changes in Scores by Program Enrollment for for East River
Not Enrolled
Enrolled
Beginning of Year
End of Year
Average Score
75
70
65
60
55
50
45
Grade 9
Grade 10
Grade 11
Grade 12
Total
59.15
61.51
48.78
50.73
56.78
67.86
55.18
58.00
56.37
59.32
55.14
58.11
54.64
52.41
63.76
56.47
56.01
63.19
53.73
56.17



Changes in Scores by Program Enrollment for for Olive
Not Enrolled
Enrolled
Beginning of Year
End of Year
Average Score
75
70
65
60
55
50
45
Grade 9
Grade 10
Grade 11
Grade 12
Total
54.41
56.12
53.10
55.47
54.11
66.72
53.54
56.52
56.00
56.84
52.96
54.33
55.69
66.85
55.31
57.83
55.11
61.94
53.72
56.24





A virtually identical set of graphs was created in an earlier program. Although these graphs were based off regression outputs, whereas the graphs in the earlier program were based off pivot tables, the numbers shown in both graphs are identical.

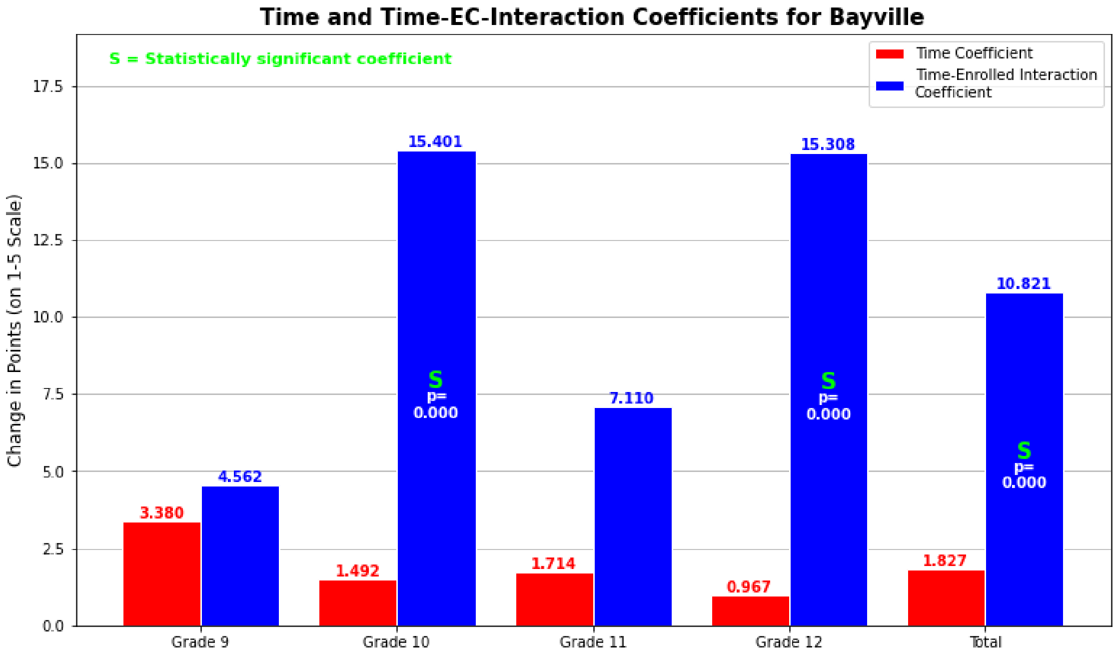
The following block of code creates charts that show the relative magnitute of the different coefficients within the DataFrames regression_table_list.

[17]:
```python
for t in range(len(regression_table_list)): # This loop runs through each database stored in the regression_table_list. i is used as an iterator below and I prefer not to use the same letter for two different iterators within a nested for loop, so I instead used t (for 'tuple') here.

    data_source = regression_table_list[t][1] # Giving this data_table a shorter variable name makes the code more readable and, more importantly, makes it easy to replace this DataFrame with a different one.
    # regression_table_list[t][2] returns a dataframe.
    school = regression_table_list[t][0]


    fig, ax = plt.subplots(figsize=[12.8,7.5]) # Widening the graph gives the x axis labels more space
    col_labels = list(data_source.columns.copy())
    print(type(col_labels))
    fig.set_facecolor('white')

    x_values = np.linspace(0,len(col_labels)-1,len(col_labels)) # Creates x values for plotting bars and other features. The list of x values iterates by 1 and is equal to the length of the number of column labels.
    # It's useful to set these manually so that other points on the graph (such as significance labels) can be plotted at the right positions.

    bar_width = 0.4
    ax.set_xticks(x_values)
    plt.grid(axis = 'y') # https://matplotlib.org/stable/api/_as_gen/matplotlib.artist.Artist.set_zorder.html#matplotlib.artist.Artist.set_zorder
    ax.set_xticklabels(col_labels, rotation = 0)
    ax.set_axisbelow(True) # https://matplotlib.org/stable/api/_as_gen/matplotlib.axes.Axes.set_axisbelow.html#matplotlib.axes.Axes.set_axisbelow
    time_coeff_bars = plt.bar(x_values - bar_width/2, data_source.loc['Time_Coeff'][:], bar_width, label='Time Coefficient',color='red',edgecolor='white',linewidth=1) # [:] is used because all columns are being included; however, this could be adjusted to only include a range of columns if needed. Converting the output to a list avoids an error message.
    time_enrolled_interaction_coeff_bars = plt.bar(x_values + bar_width/2, data_source.loc['EOY_and_Enrolled_Coeff'][:], bar_width, label='Time-Enrolled Interaction\nCoefficient',color='blue',edgecolor='white',linewidth=1)
    ax.bar_label(time_coeff_bars,label_type = 'edge',color='red',fontweight='bold', fmt='%.3f')
    ax.bar_label(time_enrolled_interaction_coeff_bars,label_type = 'edge',color='blue',fontweight='bold', fmt='%.3f')
```

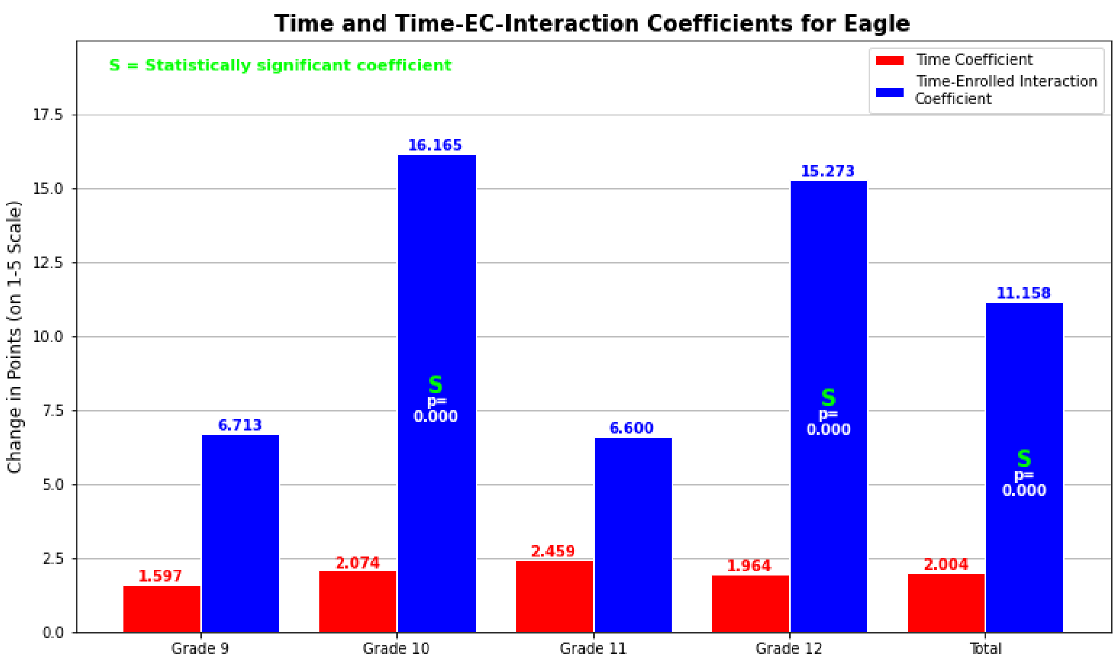
```
    ax.set_ylim(min(ax.get_ylim()[0], 0), max(ax.get_ylim()[1]+3,0))
    ax_height = ax.get_ylim()[1] - ax.get_ylim()[0]
    ax_width = ax.get_xlim()[1] - ax.get_xlim()[0]

    add_legend_flag = 0
    # Adding labels to indicate which coefficients are statistically significant
    for s in range (len(data_source.columns)):
        if data_source.loc['Time_Significant'][s] == 1:
            plt.text(x_values[s] - bar_width/2, data_source.
↪loc['Time_Coeff'][s]/2, 'S', color = (0, 1, 0), fontsize=15,␣
↪fontweight='bold', ha = 'center') # using [s]/2 places the point in the␣
↪middle of the bar. Because the y limits of the axis were set so that 0 woudl␣
↪always be included, 'S' should never be hidden from view.
            plt.text(x_values[s] - bar_width/2, data_source.
↪loc['Time_Coeff'][s]/2-ax_height/20, 'p=\n'+'{:.3f}'.format(data_source.
↪loc['Time_Pval'][s]), color = 'white', fontsize=10, fontweight='bold', ha =␣
↪'center')
            add_legend_flag = 1
            # -ax_height/[20] offsets the label by a value that scales with the␣
↪y axis dimensions so that the p value doesn't overlap with the 'S' metric.
        if data_source.loc['EOY_and_Enrolled_Significant'][s] == 1:
            plt.text(x_values[s] + bar_width/2, data_source.
↪loc['EOY_and_Enrolled_Coeff'][s]/2, 'S', color = (0, 1, 0), fontsize=15,␣
↪fontweight='bold', ha = 'center')
            plt.text(x_values[s] + bar_width/2, data_source.
↪loc['EOY_and_Enrolled_Coeff'][s]/2-ax_height/20, 'p=\n'+'{:.3f}'.
↪format(data_source.loc['EOY_and_Enrolled_Pval'][s]), color = 'white',␣
↪fontsize=10, fontweight='bold', ha = 'center')
            add_legend_flag = 1
    if add_legend_flag == 1:
        plt.text(ax.get_xlim()[0]+ax_width/30, ax.get_ylim()[1]-ax_height/20,␣
↪'S = Statistically significant coefficient', color = (0, 1, 0), fontweight =␣
↪'bold', fontsize = 11)


    plt.legend(loc = 'upper right')

    ax.set_ylabel('Change in Points (on 1-5 Scale)',fontsize=12)
    if school == 'Total':
        title_string = "Time and Time-Enrolled-Interaction Coefficients for All␣
↪Schools"
    else:
        title_string = "Time and Time-EC-Interaction Coefficients for "+school
    plt.title(title_string,fontweight='bold',fontsize=15)

    file_string = 'coeff_graphs\\'+school+'_coeffs.png'
```

```
    plt.savefig(file_string,dpi=400)
    plt.show()
```

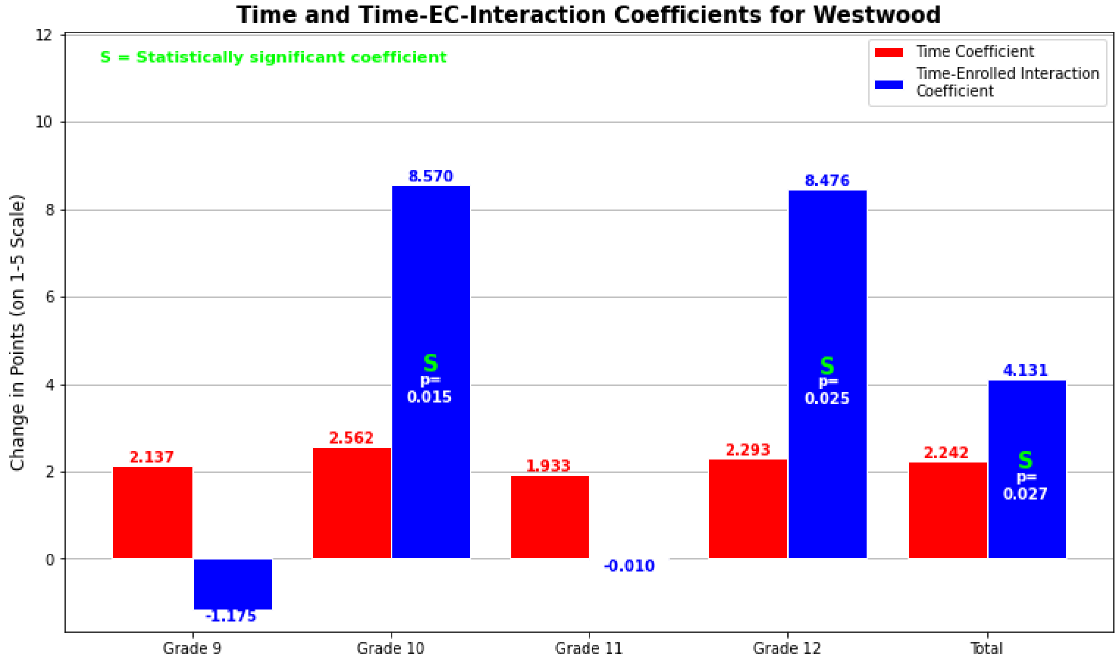
```
<class 'list'>
```



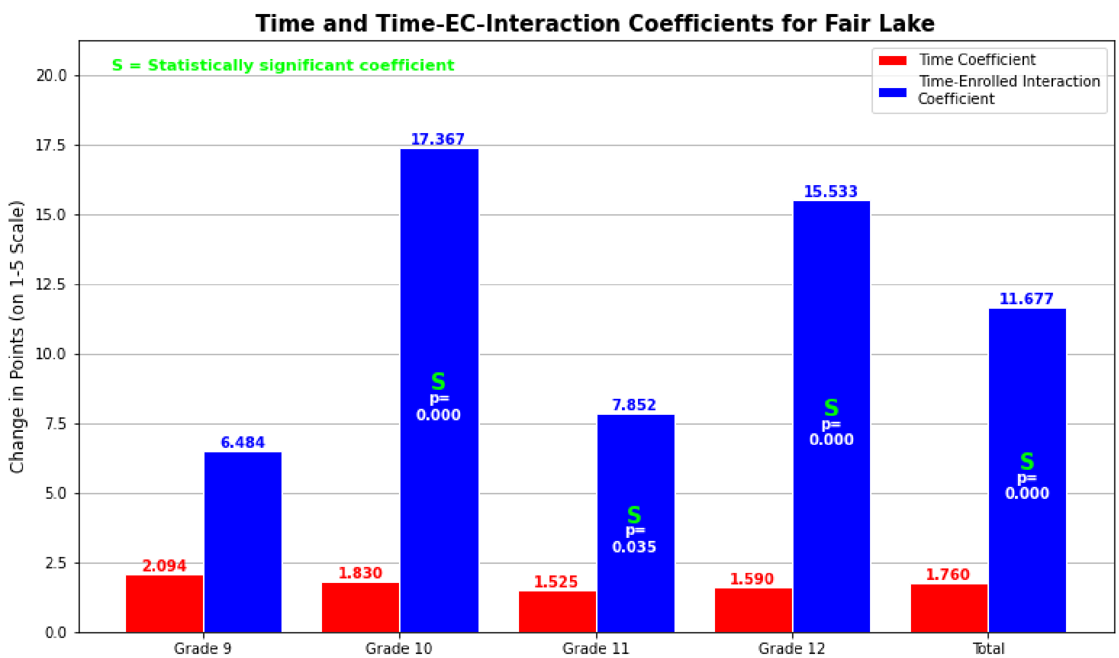

```
<class 'list'>
```



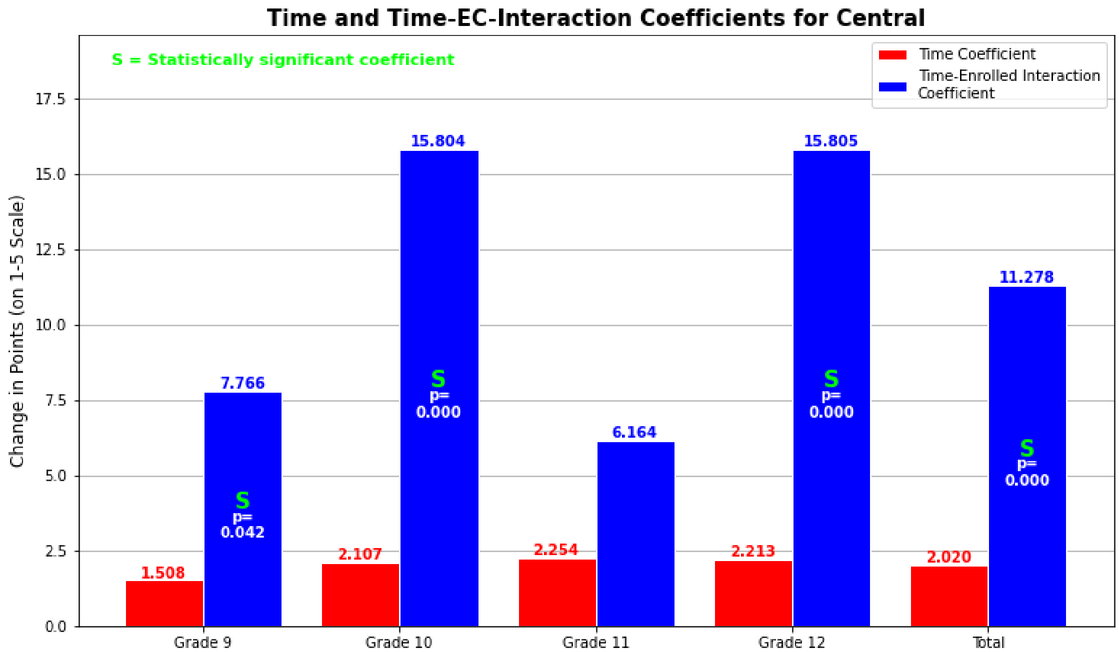
<class 'list'>



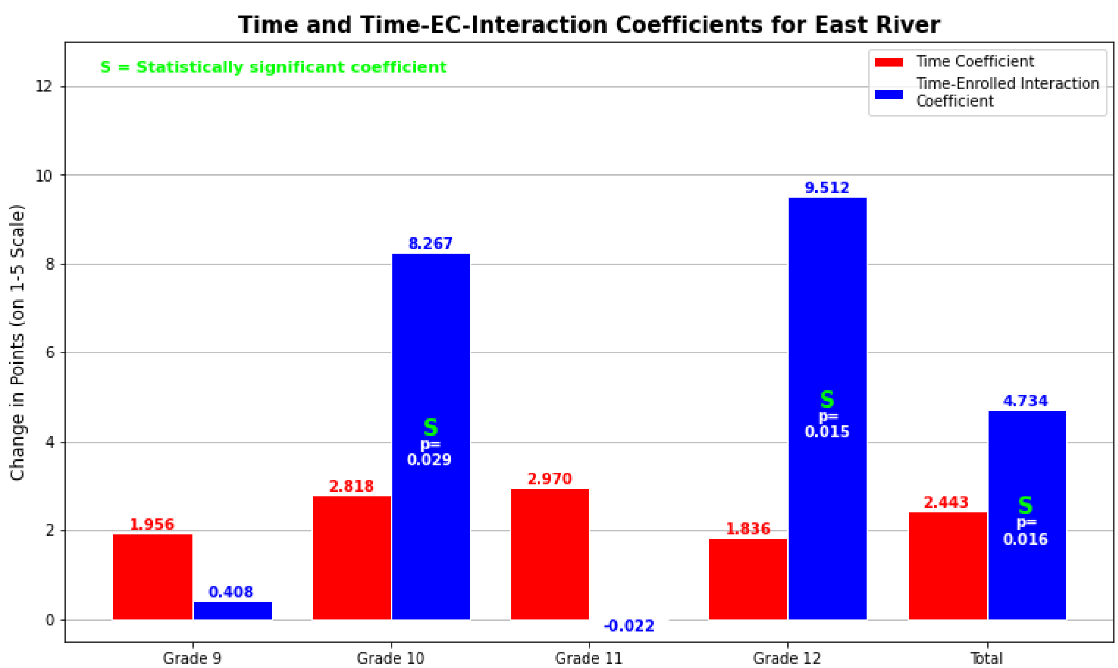
<class 'list'>



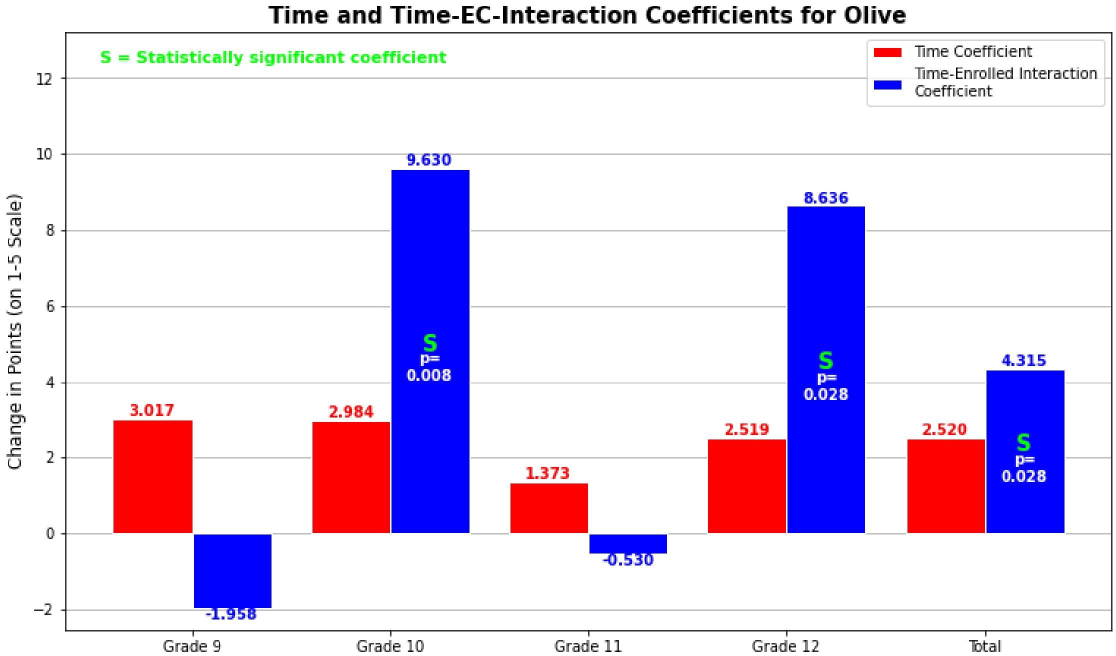
<class 'list'>



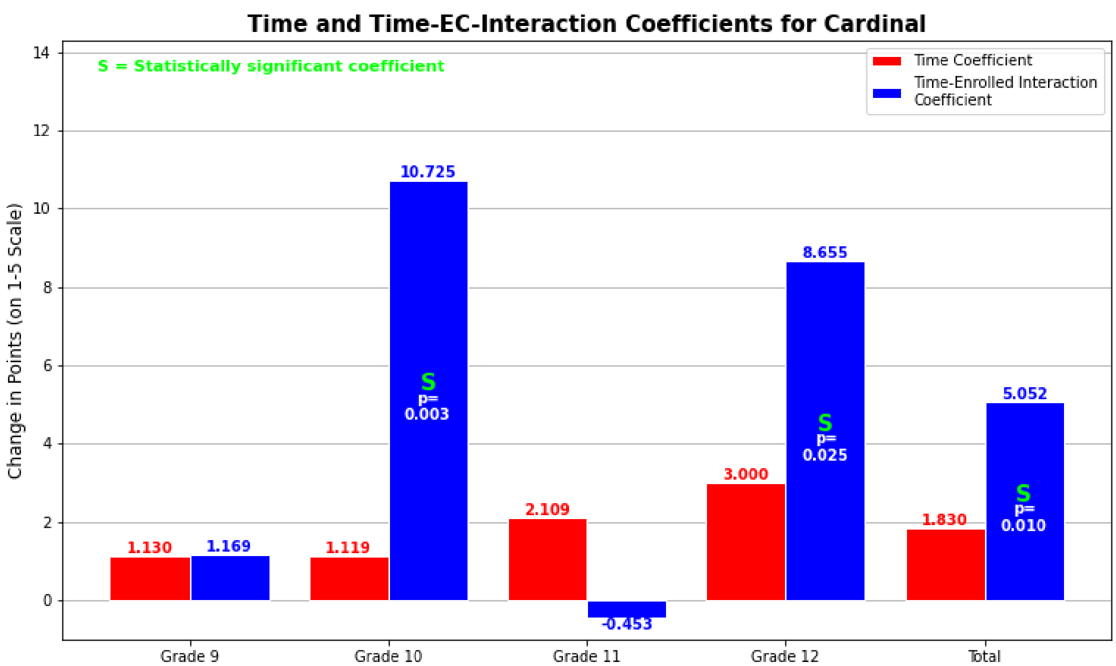
<class 'list'>



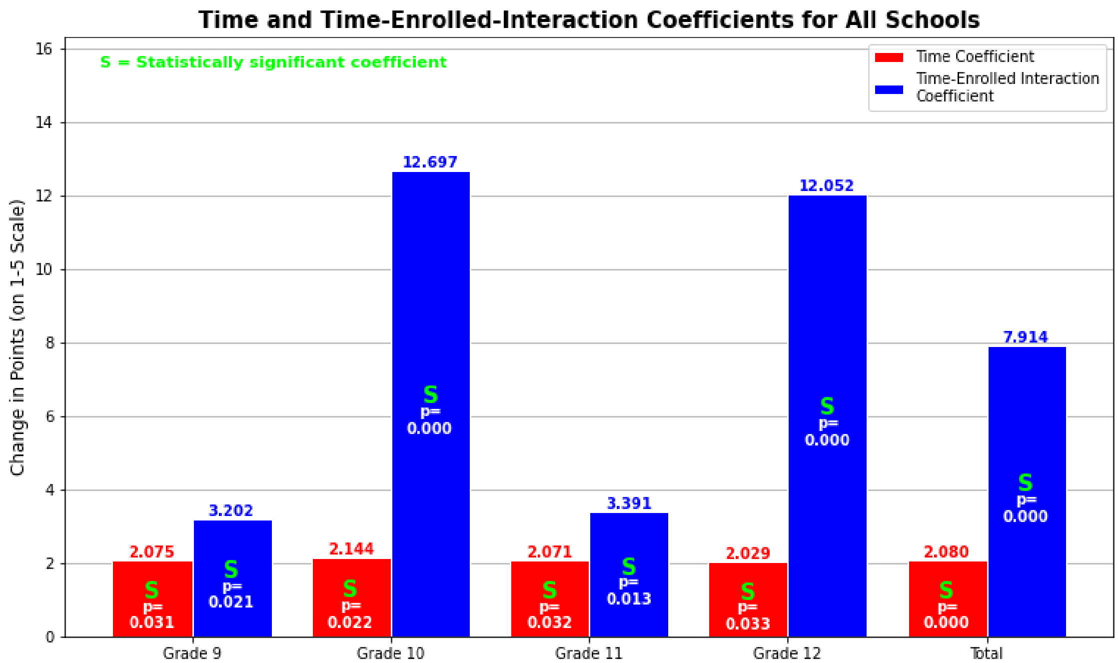
<class 'list'>



<class 'list'>



<class 'list'>



[18]:
```
with pd.ExcelWriter('regression_tables.xlsx') as writer:
    for i in range(len(regression_table_list)):
        regression_table_list[i][1].to_excel(writer, sheet_name =␣
↪regression_table_list[i][0])
```

[19]:
```
end_time = time.time()
run_time = end_time - start_time
run_minutes = run_time // 60
run_seconds = run_time % 60
print("Completed run at",time.ctime(end_time),"(local time)")
print("Total run time:",'{:.2f}'.format(run_time),"second(s)␣
↪("+str(run_minutes),"minute(s) and",'{:.2f}'.
↪format(run_seconds),"second(s))") # Only valid when the program is run␣
↪nonstop from start to finish
```

```
Completed run at Mon Jul  5 23:44:46 2021 (local time)
Total run time: 36.82 second(s) (0.0 minute(s) and 36.82 second(s))
```