```
        return 'unknown'
    season, year = parts[0].lower(), int(parts[1])
    if year < 2020:
        return 'pre'
    if year == 2020:
        return 'during'
    if year == 2021:
        if season in ['spring', 'summer']:
            return 'during'
        else:
            return 'post'
    if year > 2021:
        return 'post'
    return 'unknown'

df_merged['covid_period'] = df_merged['Term'].apply(covid_period)
print("\nMerged Data with Time and COVID Period:")
# Replace program name with a random program name for data protection
# (In practice, you would not do this; this is just for the sake of the example.
 ↪)
random_programs = [f'program{i}' for i in range(len(MS_programs))]
df_merged['Program'] = df_merged['Program'].replace(MS_programs,␣
 ↪random_programs)
print(df_merged[['Program', 'Term', 'UsageRate', 'TermDate', 'TimeIndex',␣
 ↪'covid_period']].head(10))
```

```
Merged Data with Time and COVID Period:
       Program         Term  UsageRate   TermDate  TimeIndex covid_period
99   program14  Spring 2017   0.312500 2017-02-01          0          pre
100  program15  Spring 2017   0.280702 2017-02-01          0          pre
111   program1  Spring 2017   0.111111 2017-02-01          0          pre
110   program3  Spring 2017   0.017621 2017-02-01          0          pre
108   program2  Spring 2017   0.188679 2017-02-01          0          pre
107   program4  Spring 2017   0.142857 2017-02-01          0          pre
109  program18  Spring 2017        NaN 2017-02-01          0          pre
105   program9  Spring 2017   0.058824 2017-02-01          0          pre
104  program11  Spring 2017   0.337662 2017-02-01          0          pre
103  program10  Spring 2017   0.342857 2017-02-01          0          pre
```

[34]:
```
# --- Step 5: Run a Regression Analysis using categorical covid_period ---

# Drop rows with missing UsageRate values.
df_reg = df_merged.dropna(subset=['UsageRate'])

# Use Patsy's categorical handling in the formula.
```

```python
# This automatically creates dummies with one category (here, the first, e.g.,␣
↪'pre') as the reference.
formula = 'UsageRate ~ TimeIndex + C(covid_period)'
model = smf.ols(formula=formula, data=df_reg).fit()

print("\nRegression Results:")
print(model.summary())
```

```
Regression Results:
                            OLS Regression Results
==============================================================================
Dep. Variable:              UsageRate   R-squared:                       0.046
Model:                            OLS   Adj. R-squared:                  0.034
Method:                 Least Squares   F-statistic:                     3.701
Date:                Tue, 22 Apr 2025   Prob (F-statistic):             0.0125
Time:                        16:44:41   Log-Likelihood:                -164.66
No. Observations:                 234   AIC:                             337.3
Df Residuals:                     230   BIC:                             351.1
Df Model:                           3
Covariance Type:            nonrobust
================================================================================
===========
                              coef    std err          t      P>|t|      [0.025
0.975]
--------------------------------------------------------------------------------
-----------
Intercept                   0.5968      0.162      3.694      0.000       0.278
0.915
C(covid_period)[T.post]    -0.2619      0.128     -2.040      0.042      -0.515
-0.009
C(covid_period)[T.pre]     -0.2473      0.124     -1.987      0.048      -0.493
-0.002
TimeIndex               -1.129e-05      0.000     -0.101      0.920      -0.000
0.000
==============================================================================
Omnibus:                      255.943   Durbin-Watson:                   1.591
Prob(Omnibus):                  0.000   Jarque-Bera (JB):            10117.188
Skew:                           4.454   Prob(JB):                         0.00
Kurtosis:                      33.956   Cond. No.                     9.79e+03
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly
specified.
[2] The condition number is large, 9.79e+03. This might indicate that there are
strong multicollinearity or other numerical problems.
```

1 Explanation:

2 What the Model is doing

This model estimates the following equation:

$$\text{UsageRate} = \beta_0 + \beta_1 \times \text{TimeIndex} + \beta_2 \times I(\text{covid\_period} = \text{“post”}) + \beta_3 \times I(\text{covid\_period} = \text{“pre”}) + \varepsilon$$

Since we used the formula using categorical notation with C(covid_period), one of the COVID period categories is used as the reference group. From the output, we see coefficients for: - C(covid_period)[T.post] = -0.2619 - C(covid_period)[T.pre] = -0.2473

This implies that the omitted (reference) category is the one not listed—in this case, it is “during” the COVID period. So, the intercept (0.5968) represents the expected usage rate when TimeIndex = 0 for the “during” category.

3 Interpreting coefficients

Intercept (0.5968): This is the predicted usage rate for terms in the COVID-during period (the reference category) at the baseline TimeIndex (i.e., the earliest term in your sample).

C(covid_period)[T.post] (-0.2619): This coefficient indicates that, holding TimeIndex constant, terms in the post period (after COVID) have an average usage rate that is approximately 0.262 lower than the COVID-during period.

C(covid_period)[T.pre] (-0.2473): This shows that terms before COVID have an average usage rate about 0.247 lower compared to the COVID-during period.

TimeIndex (-1.129e-05): The TimeIndex coefficient is very small and not statistically significant (p = 0.920), indicating that, after accounting for COVID period differences, there is no significant linear time trend in usage rate.

4 Statistical Significance and Model Fit

P-values: The coefficients for both covid_period dummies are statistically significant (p-values ~0.042 and 0.048), suggesting that the differences in usage rate between the “during” period and both “pre” and “post” are significant at the 5% level. However, the TimeIndex is not significant.

R-squared (0.046): The R-squared of about 4.6% indicates that the model explains only a small portion of the variability in usage rate. This is common in models where there is high variability or many unobserved factors affecting the outcome.

Diagnostic Statistics: The Omnibus test, Jarque-Bera statistic, and the high skewness and kurtosis values indicate that the residuals may be non-normal (possibly due to outliers or other distributional issues). The large condition number (9.79e+03) may hint at some multicollinearity issues, though in this model with just a few predictors it is more likely reflecting that the TimeIndex variable is on a different scale than the dummy variables.

5 What It Means in Context

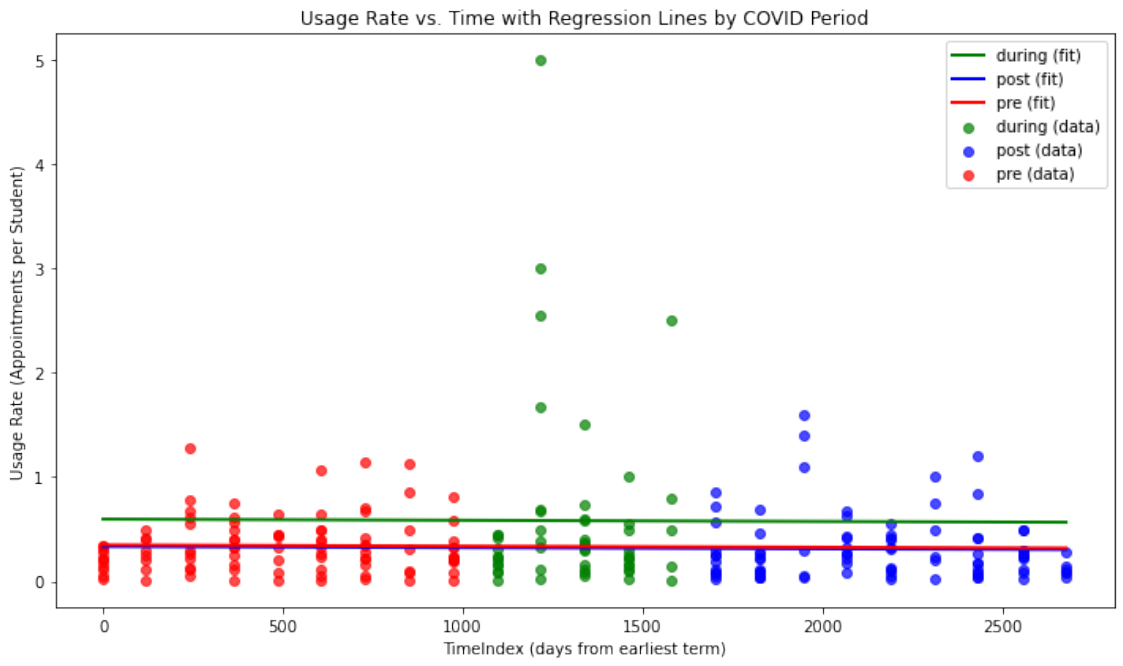
COVID Period Effect: The results suggest that during the COVID "during" period, usage rates (appointments per enrolled student) were higher—by about 0.25 on average—than in both the pre- and post-COVID periods. In other words, there was an elevated usage rate during COVID relative to before or after.

Time Trend: The lack of significance for TimeIndex means that, aside from the abrupt changes associated with the COVID periods, there isn't a clear, gradual trend in the usage rate over time.

Overall Fit and Considerations: While the model finds statistically significant differences among the COVID periods, the overall model fit is relatively weak (low R-squared), indicating that many other factors may also be influencing the usage rate. Additionally, the diagnostics hint at some possible non-normality or multicollinearity concerns, which we might want to investigate further (for example, by checking for outliers or considering transformations).

[35]:
```python
# Retrieve model parameters
params = model.params
beta0 = params['Intercept']
beta_time = params['TimeIndex']

# For our model, the reference category is "during".
# For the other categories, we expect parameters like:
#   C(covid_period)[T.pre] and C(covid_period)[T.post]
# (Make sure you have not manually dropped the original 'covid_period' column␣
↪in df_reg.)

# Create a scatter plot of UsageRate vs. TimeIndex, colored by covid_period
fig, ax = plt.subplots(figsize=(10, 6))

# Get unique COVID periods from df_reg (the original categorical values)
covid_categories = sorted(df_reg['covid_period'].unique())
colors = {'pre': 'red', 'during': 'green', 'post': 'blue'}

# Plot each group as a scatter
for cat in covid_categories:
    subset = df_reg[df_reg['covid_period'] == cat]
    ax.scatter(subset['TimeIndex'], subset['UsageRate'],
               color=colors.get(cat, 'gray'),
               label=f"{cat} (data)", alpha=0.7)

# Determine the range for TimeIndex to create smooth lines
x_min = df_reg['TimeIndex'].min()
x_max = df_reg['TimeIndex'].max()
x_vals = np.linspace(x_min, x_max, 100)

# Overlay the regression lines for each covid_period category.
# For the reference category "during", the predicted value is:
```

```
#   y = beta0 + beta_time * TimeIndex
# For other categories, add the dummy coefficient.
for cat in covid_categories:
    if cat == 'during':
        y_vals = beta0 + beta_time * x_vals
    else:
        dummy_name = f"C(covid_period)[T.{cat}]"
        offset = params.get(dummy_name, 0.0)
        y_vals = beta0 + offset + beta_time * x_vals
    ax.plot(x_vals, y_vals, color=colors.get(cat, 'gray'), linewidth=2,
            label=f"{cat} (fit)")


# Add labels, title, and legend
ax.set_xlabel("TimeIndex (days from earliest term)")
ax.set_ylabel("Usage Rate (Appointments per Student)")
ax.set_title("Usage Rate vs. Time with Regression Lines by COVID Period")
ax.legend()
plt.tight_layout()
plt.show()
```



6 Explanation:

This code performs a regression analysis on the usage rates of various MS programs over time, segmented by COVID periods. Here's a breakdown of the key steps:

1. Scatter Plot:
   - The code loops over each unique value of covid_period (e.g., "pre", "during", "post") and creates a scatter plot of UsageRate versus TimeIndex in different colors.
2. Fitted Lines:
   - We define a range of x values (x_vals) spanning from the minimum to maximum TimeIndex.
   - For the reference category ("during"), the predicted values are given by: $\hat{y} = \beta_0 + \beta_{\text{TimeIndex}} \cdot \text{TimeIndex}$
   - For other categories (e.g., "pre" and "post"), the predicted values include the corresponding dummy variable offset: $\hat{y} = \beta_0 + \beta_{\text{TimeIndex}} \cdot \text{TimeIndex} + \text{DummyOffset}$
3. Plot Customization:
   - Colors are assigned for each category for clarity.
   - Both the scatter points (labeled as data) and the fitted line (labeled as fit) are added to the plot with a legend.

When you run this code, you should see a scatter plot of your usage rates (from df_reg) along with regression lines showing the trend for each COVID period category. This visual presentation can help you examine how usage behavior changes across time and among the different COVID periods.

[36]:
```python
# --- Step 1: Filter for the 'during' COVID period ---
df_during = df_reg[df_reg['covid_period'] == 'during'].copy()

# --- Step 2: Compute predicted UsageRate for the "during" group ---
# Since for the reference group ("during") the model is:
#   Predicted UsageRate = beta0 + beta_time * TimeIndex
df_during['Predicted'] = beta0 + beta_time * df_during['TimeIndex']

# --- Step 3: Compute residuals and standardized residuals ---
df_during['Residual'] = df_during['UsageRate'] - df_during['Predicted']

# We compute an approximate standardized residual (z-score) for the subset.
# (For more robust values one might use model.get_influence() on the full model,
# but here we approximate using the mean and std of the residuals in the subset.
 ↪)
mean_resid = df_during['Residual'].mean()
std_resid = df_during['Residual'].std()
df_during['StandardizedRes'] = (df_during['Residual'] - mean_resid) / std_resid

# --- Step 4: Flag observations with |StandardizedRes| > 2 as potential␣
 ↪outliers ---
threshold = 2
outliers_during = df_during[np.abs(df_during['StandardizedRes']) > threshold]

# --- Step 5: Report which programs have outlier observations ---
outlier_programs = outliers_during['Program'].unique()
print("Programs with outliers (|standardized residual| > 2) in the 'during'␣
 ↪COVID period:")
```

```
print(outlier_programs)

# Optionally, you can further inspect the outliers:
print("\nOutlier observations during COVID period:")
print(outliers_during[['Program', 'Term', 'TimeIndex', 'UsageRate',␣
 ↪'Predicted', 'Residual', 'StandardizedRes']])
```

```
Programs with outliers (|standardized residual| > 2) in the 'during' COVID
period:
['program2' 'program6' 'program4']

Outlier observations during COVID period:
      Program         Term  TimeIndex  UsageRate  Predicted  Residual  \
245  program2  Summer 2020       1216   2.545455   0.583068  1.962386
243  program6  Summer 2020       1216   5.000000   0.583068  4.416932
244  program4  Summer 2020       1216   3.000000   0.583068  2.416932
251  program6  Summer 2021       1581   2.500000   0.578949  1.921051

     StandardizedRes
245         2.215260
243         4.986100
244         2.728379
251         2.168599
```

7 Explanation:

1. Filtering for the "during" Group:

   We limit our data to only those observations whose covid_period equals "during".

2. Computing Predictions & Residuals:

   For the "during" group the model predicted value is computed as:

   $\hat{y} = \beta_0 + \beta_{\text{TimeIndex}} \times \text{TimeIndex}$

   The residual is the difference between the actual UsageRate and this predicted value.

3. Standardizing Residuals:

   The standardized residual is computed by subtracting the mean of the residuals and dividing by the standard deviation. Observations with an absolute standardized residual greater than 2 are flagged as potential outliers.

4. Reporting Outliers by Program:

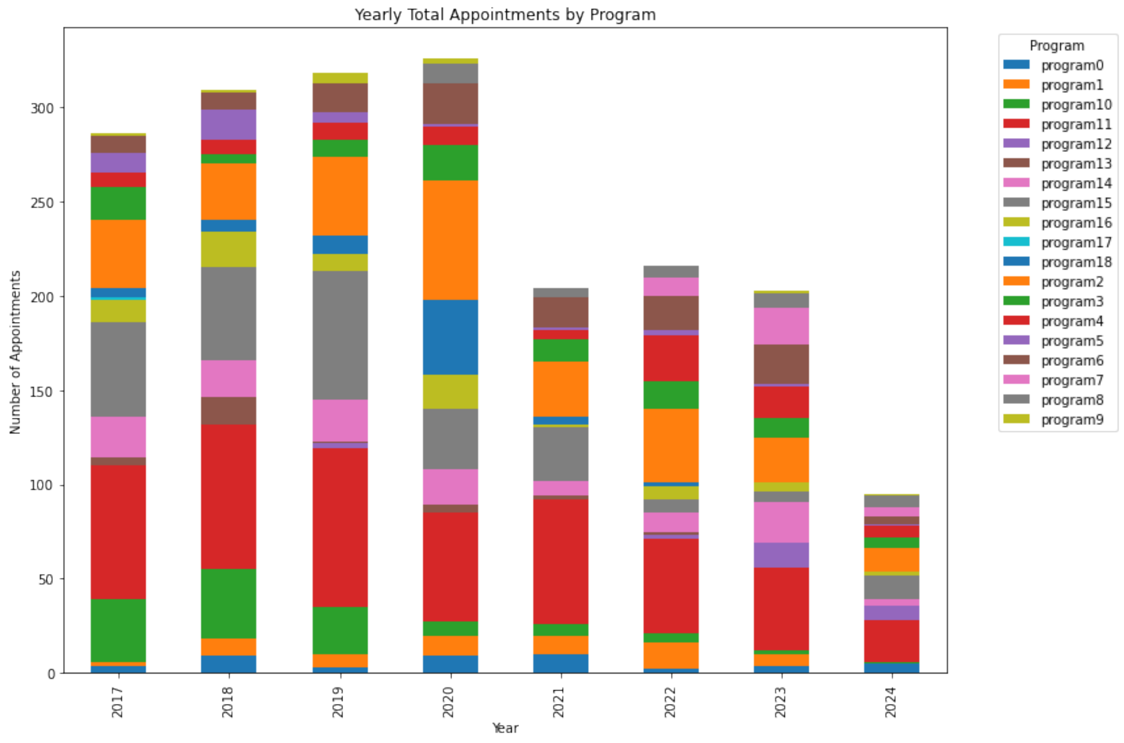
   Finally, we extract the unique program names from the flagged outlier observations.

Running this snippet will print which programs have outlying observations during the COVID "during" period and then list the details for further inspection.

[37]:
```
from scipy.stats import chi2_contingency

# --- Step 1: Create a 'year' column from the Appointment Date ---
# (Assumes df_appointments exists and that 'Appointment Date' is of type␣
↪datetime.)
df_appointments['year'] = df_appointments['Appointment Date'].dt.year
df_appointments['Program'] = df_appointments['Program'].replace(MS_programs,␣
↪random_programs)

# --- Step 2: Create a pivot table for the yearly breakdown by Program ---
# Each row is a year, columns are programs, and the values are the count of␣
↪appointments.
yearly_breakdown = df_appointments.groupby(['year', 'Program']).size().
↪unstack(fill_value=0)
print("Yearly Breakdown by Program:")
print(yearly_breakdown)

# --- Plot a stacked bar chart of the yearly appointments by program ---
fig, ax = plt.subplots(figsize=(12, 8))
yearly_breakdown.plot(kind='bar', stacked=True, ax=ax)
ax.set_title('Yearly Total Appointments by Program')
ax.set_xlabel('Year')
ax.set_ylabel('Number of Appointments')
ax.legend(title='Program', bbox_to_anchor=(1.05, 1), loc='upper left')
plt.tight_layout()
plt.show()


# --- Step 3: Create a contingency table by COVID period for appointments by␣
↪program ---
covid_breakdown = df_appointments.groupby(['covid_period', 'Program']).size().
↪unstack(fill_value=0)
print("\nBreakdown by COVID Period (Appointments by Program):")
print(covid_breakdown)

# --- Step 4: Run a Chi-Square Test on the COVID Breakdown ---
chi2, p_val, dof, expected = chi2_contingency(covid_breakdown)
print("\nChi-square Test Results:")
print(f"Chi2 Statistic: {chi2:.3f}")
print(f"Degrees of Freedom: {dof}")
print(f"P-value: {p_val:.5f}")

# Interpretation:
if p_val < 0.05:
    print("The distribution of appointments by program differs significantly␣
↪across COVID periods (p < 0.05).")
```

```
else:
    print("There is no significant difference in the distribution of␣
 ↪appointments by program across COVID periods (p >= 0.05).")
```

```
Yearly Breakdown by Program:
Program  program0  program1  program10  program11  program12  program13  \
year
2017            4         2         33         71          0          4
2018            9         9         37         77          0         14
2019            3         7         25         84          3          1
2020            9        11          7         58          0          4
2021           10        10          6         66          0          2
2022            2        14          5         50          2          2
2023            4         6          2         44         13          0
2024            5         0          1         22          8          0

Program  program14  program15  program16  program17  program18  program2  \
year
2017            22         50         12          1          5        36
2018            20         49         19          0          6        30
2019            22         68          9          0         10        42
2020            19         32         18          0         40        63
2021             8         28          2          0          4        29
2022            10          7          7          0          2        39
2023            22          5          5          0          0        24
2024             3         13          2          0          0        12

Program  program3  program4  program5  program6  program7  program8  program9
year
2017           18         7        11         9         0         0         1
2018            5         8        16         9         0         0         1
2019            9         9         5        16         0         0         5
2020           19        10         1        22         0        10         3
2021           12         5         1        16         0         5         0
2022           15        24         3        18        10         6         0
2023           10        17         1        21        20         7         2
2024            6         6         1         4         5         6         1
```



```
Breakdown by COVID Period (Appointments by Program):
Program       program0  program1  program10  program11  program12  program13  \
covid_period
during              14        17         10         79          0          6
post                16        24         11        161         23          2
pre                 16        18         95        232          3         19

Program       program14  program15  program16  program17  program18  program2  \
covid_period
during               22         49         18          0         44        79
post                 40         36         16          0          2        88
pre                  64        167         40          1         21       108

Program       program3  program4  program5  program6  program7  program8  \
covid_period
during              26        14         1        27         0        13
post                36        48         6        54        35        21
pre                 32        24        32        34         0         0

Program       program9
covid_period
during               3
post                 3
```

```
pre                 7

Chi-square Test Results:
Chi2 Statistic: 434.588
Degrees of Freedom: 36
P-value: 0.00000
The distribution of appointments by program differs significantly across COVID
periods (p < 0.05).
```

8 Explanation

1. *Year Extraction:*

   We add a new column, year, using df_appointments['Appointment Date'].dt.year so that we can aggregate by year.

2. *Yearly Pivot Table & Plot:*

   The pivot (or contingency) table, yearly_breakdown, contains appointment counts for each program per year. We then plot this as a stacked bar chart so you can see the distribution visually.

3. *Contingency Table by COVID Period:*

   Next, we group by covid_period and Program to see how many appointments fell into each combination; this table is stored as covid_breakdown.

4. *Chi-Square Test:*

   The chi-square test (chi2_contingency) checks whether the distribution (i.e., the proportion of appointments across programs) is independent of the COVID period. A significant p-value (commonly $p < 0.05$) indicates that the distributions differ significantly.

9 Test Results

```
The chi-square test output shows:
•   Chi-square Statistic: 434.588
•   Degrees of Freedom: 36
•   P-value: effectively 0 (p < 0.00001)
```

9.1 What This Means

1. **Null Hypothesis (H):**

   Under the null hypothesis, the distribution of appointments by program would be independent of the COVID period (i.e., pre-, during-, and post-COVID periods have the same distribution of appointments across programs).

2. **Test Result:**

   The chi-square statistic is very large (434.588) relative to 36 degrees of freedom, and the p-value is extremely small. This means that we have strong statistical evidence to reject the null hypothesis.

3. **Conclusion:**

   The significant result ($p < 0.05$) indicates that the distribution of appointments by program differs significantly across the COVID periods. In other words, the proportions with which appointments occur in each program are not the same before COVID, during COVID, and after COVID. There is a statistically significant shift in the pattern of appointments by program across these periods.

9.2 Implications

- **Program Prioritization:** Some programs may have seen increases or decreases in appointment counts relative to their overall distribution—for example, if one program has a comparatively higher number of appointments during COVID compared to pre-COVID or post-COVID.
- **Resource Allocation:** The significant differences in appointment distributions might suggest that resource usage (or demand) shifted during the different periods. This could be useful for planning or adjusting support or outreach based on which programs showed marked changes.
- **Further Analysis:** To understand which specific programs contribute most to this overall difference, we might: • Look at standardized residuals in your contingency table. • Identify which cells (program and COVID period combinations) show the greatest deviation between observed and expected counts.