How to run our software:

Libraries necessary to install:

- pandastables (pip install pandastables)

Directly run the file input_interface.py
This should call all necessary files

How to use our software:

Our software features 3 main ways for a user to obtain information about the effectiveness of a state's education policies in a tkinter interface, with differently colored frames, each of which contains unique widgets that bring up useful information for a user in new pop-up windows.

Blue Window: Also known as user input Option 1, this first feature allows a user to see the effect a state's or states' policies have on our completed set of educational outcome trends over the years. Simply select the state(s) you are interested in seeing information about, and a new window will pop up with our calculated score of how effective the state's policies are on the wellbeing of its educational outcomes, text stating which policies our data shows to be the most effective and least effective within that state, and a series of scatter plots with best-fit lines in them. These plots show the state's best policy score vs. trend in educational outcomes for all states (including US average and DC), and they visualize how a policy's quality score from the National Council on Teacher Quality correlates to various outcomes.

Purple Window: Also known as user input Option 2, this window allows a user to input a state and an outcome variable of interest and performs similar calculations to the blue window, now with certain outcome variables selected rather than all of them. It similarly produces a window with our calculated score, which policies are the most and least effective at predicting the input(s), and scatter plots organized the same way as in option 1.

Green Window: Also known as user input Option 3, this window allows a user to see the relationship between an outcome of interest and a policy or set of policies. For example, a user might surmise that mathematics preparation for elementary teachers will be predictive of elementary math standardized test scores. They could then plug in "4th Grade Math Scores" on the left drop down window and "Elementary Teacher Preparation in Mathematics" as well as any number of other policies too on the right. A window will then pop up with information about a regression analysis that occurs, including a scatter plot with a trend line.

Orange Window: The predictive value of these policies on outcomes was not as large as our group had hoped. As a helpful tool for users to isolate the most predictive policies on a given outcome, a user is able to input an outcome in the left dropdown window and input an R^2 into the text box and hit submit. In the window at the bottom of the frame, a list of all the policies that have an R^2 value greater than the input R^2 will pop up. We strongly recommend inputting small R^2 values (<0.1).

Yellow Window: This window contains 3 buttons that allow users to see the data underlying our calculations. The first button opens up a window that contains the National Center for Education Statistics dataframe we used. Information on how this data was extracted and manipulated is located in nces.py. The second button pulls up the data we extracted from the National Council on Teacher Quality's State Teacher Policy Database. Information on how this data was extracted is located in nctq.py. The last button brings up information about how NCTQ defines quality policies, bringing up a window with the type of policy on the left and NCTQ' definition of quality on the right.

Documentation of each file/folder:

Data

- csv folder: holds csv and json data from our nces and nctq crawlers, updated when those crawlers are run and stored in this folder. This is so we do not have to crawl every time we run our program, and can instead reference our static data.
    - nctq csv files are stored in separate csvs for each year of data collected. We calculate averaged data to use in our computation.
    - policydesc_dic.json is another return from the nctq crawler, which stores the policy names and the descriptions NCTQ provides. We use this for user reference so they can look up what a policy name means.

    - The nces csv files hold the raw crawler data, and the final data which has been normalized and includes trend information

Data Extraction

- nces.py: contains functions that effectively “crawl” through a series of hand picked links on the National Center for Education Statistics website. It utilizes a pair of specially designed dictionaries and a series of functions to pick out the exact information we want from tables on the nces website, cleans those tables up to make the data usable, joins tables together, and manipulates the data to assist in later analysis. The ultimate output of this file is nces.final, a data frame containing the outcome variables utilized for our analysis.
- Nctq.py- contains functions to crawl through the National Council on Teacher Quality’s State Teacher Policy database, extracting information about each state’s “score” on NCTQ’s evaluation. These scores come in the form of evaluations like “Meets Goal”, “Meets Goal in Part”, and “Meets a Small Part of Goal” (of which there are 6). They were converted to their corresponding integers, and a dataframe was created with the index as states and the columns as scores on each policy for each state. This file also outputs a json file containing a dictionary mapping policy types to NCTQ’s description of what a quality policy in that area looks like.
- Util.py- Contains helper functions utilized in the nctq.py crawler.

Data Manipulation

- basic_regression.py: contains functions to create linear regression models between different policies and outcomes and to set an R2 score cutoff to filter out regression models with low correlation. This file also contains a function to determine the regression model (and corresponding policy) with the highest R2 score.
- default_stat_analysis.py: computes both an overall effectiveness score and individual policy effectiveness scores for each state. This file also contains functions for calculating a normalized overall effectiveness score and finding the best policy (highest effectiveness score) and the worst policy (lowest effectiveness score) for each state.
    - This is the file that does most of the heavy lifting for our state evaluations. First, we use centralized and normalized data in order to make comparisons possible. Second, to minimize the risk of reactive correlations, we take only trend datas for our default computations,

and see how policies influence trends (improvements) in educational outcomes. As is to be expected with educational data, and due to the limited sample size (50 states + DC), our results are not statistically strong. We have employed various methods of normalization, filtering, and more, documented closely in the code of this file, to produce the scores and get as close to adequate outcomes as possible.

- fws.py: contains a function that creates a linear regression model with forward selection when more than three policies are selected or runs a bivariate or univariate linear regression model when two or less policies are selected and outputs a dictionary containing the linear model coefficients for each policy and the model score to assist in calculating policy weights.
- pseudo code project_hq.py: this is a preliminary outline of how our defalt_calc() function and generally the default_stat_analysis.py file would calculate our scores and retrieve other relevant return data. Our functions required much tweaking and restructuring, but this pseudo code is useful for understanding our initial, and many of our final, directions in how we used statistical analysis and created a method to find effectiveness scores, and how we initially planned to handle user input option 3.
- one_policy_one_outcome.py: This was a function that was not ultimately used in the final product. However, it was initially created to assist in the regression calculations that are required for output2.py, in the case where the user only inputs 2 variables total. We were able to combine this and one_pol_one_outcome.py into one more streamline and robust method in output2.py
- Mult_policies_one_outcome.py: This was a function that was not ultimately used in the final product. However, it was initially created to assist in the regression calculations that are required for output2.py in the case where the user inputs one outcome and more than one policy. We were able to combine this and one_pol_one_outcome.py into one more streamline and robust method in output2.py
- program_hq.py: This was a file not ultimately called in our final running of the software. Originally, this file was going to be used as what would become our input_interface file. It was going to call all the calculations and helper functions and call our separate user interface. Tkinter lends itself to being the interface and also calling the calculations, so that is what we ended up doing, but we still often used this function for reference of

retrieving data we needed from other files. It was also used later for testing purposes.

Interface

- input_interface.py: this is the main file of our project, which calls our main calculation files and produces the interactive user interface, with calls to other files that assist the UI as well for further pop-up windows. This is created in an object oriented way using python's default UI package tkinter.
- output1.py: is called by input_interface.py to create a new window that handles both user input Option1 and 2 robustly to produce our main output return screen. Again, this is done in an object oriented manner, where the tkinter window itself is defined as its own object, as well as the miniFrames that it consists of, with one being created per state. This outputs, for each state, our calculated score, its best and worst policies within the state relative to the outcomes considered, and matplotlib scatterplots to visualize some of this data.
- ouput2.py:
  Contains class that creates a tkinter window with corresponding outputs. This page shows the results of a regression between the selected variables. It contains:
  1. A table containing the results of a simple or multilinear regression where the rows are each policy plus the intercept and R2, and the column holds the coefficients of each independent variable and the intercept, R2.
  2. A plot of the most powerful explanatory variable in the selected policies against the outcome variable.
  3. If multiple variables are selected, a correlation matrix of all variables in the analysis.
- ui_util.py: contains essential classes used in building our final tkinter window. This file contains two classes. The first is VerticalScrolledFrame, which is what we use to build our master frame in input_interface, featuring the ability to scroll. The second class NewWindow is utilized in input_interface to allow a new window to be opened up when a button is pressed.
- ui_plot.py: scplot() is called by output2.py to plot the scatterplots with lines of best fit to visualize selected policy and outcome data along with the linear regression model relating the two. The remaining functions in this file were initially meant to plot another scatterplot, output a

matplotlib.Table object to display the data frame data, and abbreviate lengthy policy or outcome names in the display. However, they were ultimately not used due to compatibility issues with the tkinter user interface.

A note on the difficulty of education research:

Our group entered this project with an idea in mind that experts' evaluations (NCTQ's) of what makes good education policy would have sizable predictive value on real world education outcomes. Our final product certainly outputs interesting results. For example, our algorithm strongly believes in the predictive power of strong policies around teacher's pay scales on various relevant education outcomes. However, overall, we found that finding meaningful relationships between policy and outcomes is incredibly difficult, and the task of controlling for the myriad of confounders (demographics, time between policy enactment and effects, random variation) was beyond the scope of this project. You can visualize these difficulties directly using the user interface's orange frame options, where the R2 values are fairly low, with few above 0.3. Still, we hope that you find our program as interesting as we do.

Policies that had the most influence:

Taken from the policy_weight_dic computed in the default_stat_analysis file's default_calc() function.
Below are results for when filtering by R2 = 0.05 and R2 = 0.1

```
states_overall_effectiveness_score, state_to_policy_effectiveness_score, policy_weight_dic =
d.default_calc(avg_nctq, centered_avg_nctq, nces_trends, 0.05, True):

{'Compensation for Prior Work Experience (Retaining Effective Teachers Policy)':
0.8039621953795837,
 'Frequency of Evaluation and Observation (Teacher and Principal Evaluation Policy)':
0.757658664065781,
 'Licensure Deficiencies (Special Education Teacher Preparation Policy)': 0.2648357262741035,
 'Extended Emergency Licenses (Exiting Ineffective Teachers Policy)': 0.6831807064203643,
 'Measures of Professional Practice (Teacher and Principal Evaluation Policy)':
0.6550843228367323,
 'Adolescent Literacy (Secondary Teacher Preparation Policy)': 0.1458786794179821,
 'New Teacher Induction (Retaining Effective Teachers Policy)': 0.3074144457492294,
 'High-Need Schools and Subjects (Teacher Compensation Policy)': 0.4369148915818726,
 'Pay Scales (Retaining Effective Teachers Policy)': 1.9106369322237866,
 'Provisional and Emergency Licensure (Hiring Policy)': 0.16961552713625885,
 'Reductions in Force (Exiting Ineffective Teachers Policy)': 0.3661483325615727,
 'Admission into Preparation Programs (Delivering Well Prepared Teachers Policy)':
0.6251585264358706,
 'Principal Evaluation and Observation (Teacher and Principal Evaluation Policy)':
0.38154323874865403,
 'Pension Neutrality (Pensions Policy)': 0.5729625605686729,
 'Tenure (Identifying Effective Teachers Policy)': 0.5296213193951326,
 'Part Time Teaching Licenses (Expanding the Pool of Teachers Policy)': 0.0699319637775183}
```

```
states_overall_effectiveness_score, state_to_policy_effectiveness_score, policy_weight_dic =
d.default_calc(avg_nctq, centered_avg_nctq, nces_trends, 0.1, True)
```

```
{'Pension Sustainability (Retaining Effective Teachers Policy)': 0.8624753572733922,
 'Licensure Deficiencies (Special Education Teacher Preparation Policy)': 0.2648357262741035,
 'Extended Emergency Licenses (Exiting Ineffective Teachers Policy)': 0.7522329462569142,
 'Induction (Retaining Effective Teachers Policy)': -0.0625240691434646,
 'New Teacher Induction (Retaining Effective Teachers Policy)': 0.2965920737492752,
 'Pay Scales (Retaining Effective Teachers Policy)': 1.1902795048636499,
 'Early Childhood (Delivering Well Prepared Teachers Policy)': 1.2785310972750918}
```

Some conclusive statements: Overwhelmingly, the best policy we found per state in terms of its level of impact for all outcomes in general for that state, were policies related to Pay Scales, Pension Sustainability, and Early Childhood Teacher Prep. Very often, the least important policy was Induction pertaining to retaining effective teachers. These patterns are to be expected, and are explanatory in and of themselves. Our data suggests what matters most for improving education outcomes is less pedagogical policy goals for children's success and more policies that reflect retaining and upkeeping teacher environments themselves. While our tool does substantial analysis, much further analysis can be done by a user who has access to our tool as well.