ERDC/ITL SR-19-18

Information Technology Laboratory

US Army Corps
of Engineers®
Engineer Research and
Development Center

ERDC
ENGINEER RESEARCH & DEVELOPMENT CENTER

Engineered Resilient Systems

TradeAnalysis v1.1 Technical Guide

Joshua Q. Church

September 2019

ERS
ENGINEERED RESILIENT SYSTEMS
DEPARTMENT OF DEFENSE

The U.S. Army Engineer Research and Development Center (ERDC) solves the nation's toughest engineering and environmental challenges. ERDC develops innovative solutions in civil and military engineering, geospatial sciences, water resources, and environmental sciences for the Army, the Department of Defense, civilian agencies, and our nation's public good. Find out more at www.erdc.usace.army.mil.

To search for other technical reports published by ERDC, visit the ERDC online library at http://acwc.sdp.sirsi.net/client/default.

ERDC/ITL SR-19-18
September 2019

TradeAnalysis v1.1 Technical Guide

Joshua Q. Church

Information Technology Laboratory
U.S. Army Engineer Research and Development Center
3909 Halls Ferry Road
Vicksburg, MS 39180-6199

Final Report

Prepared for Headquarters, U.S. Army Corps of Engineers
Washington, DC 20314-1000

Under Engineered Resilient Systems Program, Data Analytics Work Package,
Collaborative Tradespace Analytics Work Unit 92L5D8

Abstract

The Engineered Resilient Systems (ERS) program focuses on the acquisition process for the Department of Defense (DoD). Within ERS, the Tradespace Analysis area has developed a web-based application called TradeAnalyzer, which allows users to apply various analysis techniques across their datasets. However, many customers across the DoD faced issues connecting to external sources.

To mitigate this issue, ERS developed a local Python edition of the application called TradeAnalysis. TradeAnalysis enables the customer to perform high-fidelity tradespace analysis and generate visualizations in a local environment. This application allows users to quickly generate, export, and share offline, standalone, and custom visualizations to further assist the DoD acquisition process.

This report will cover the required dependencies to run the TradeAnalysis application, the recommended software for Python package management, and a breakdown of the underlying code.

Contents

Report Documentation Page

Preface

This report is a deliverable product under the Engineered Resilient Systems (ERS) Program, Data Analytics Work Package, Collaborative Tradespace Analytics Work Unit 92L5D8. Dr. Owen J. Eslinger was the Program Manager; and Dr. Robert M. Wallace was the Technical Director.

The work was performed by the Scientific Software Branch (SSB) and Computational Analysis Branch (CAB) of the Computational Science and Engineering Division (CSED), Engineer Research and Development Center (ERDC), Information Technology Laboratory (ITL), Vicksburg, MS.

At the time of publication, Mr. Timothy Dunaway was Chief, SSB and Dr. Jeffrey L. Hensley was Chief, CAB. Dr. Jerrell R. Ballard was Chief, CSED. The Deputy Director of ITL was Ms. Patti S. Duett and the Director was Dr. David A. Horner.

COL Teresa A. Schlosser was Commander of ERDC, and Dr. David W. Pittman was the Director.

Acronyms and Abbreviations

Acronym	Meaning
CAB	Computational Analysis Branch
CSED	Computational Science and Engineering Division
DoD	Department of Defense
ERS	Engineered Resilient Systems
ERDC	Engineer Research and Development Center
ITL	Information Technology Laboratory
Pip	Pip Installs Packages
SSB	Scientific Software Branch
UI	User Interface
USACE	U.S. Army Corps of Engineers
2D	2-Dimensional
3D	3-Dimensional

1 Introduction

1.1 Background

The Engineered Resilient Systems (ERS) program focuses on the acquisition process for the Department of Defense (DoD). The ERS program has developed a local Python application that enables DoD customers to perform high-fidelity tradespace analysis and generate visualizations in a local environment. This application allows users to quickly generate, export, and share offline, standalone, and custom visualizations to further assist the DoD acquisition process.

1.2 Objectives

The intent of this report is to assist the user with the installation and execution process for the TradeAnalysis application. Additionally, this report covers a detailed breakdown of the application code so that end-users will have the ability to modify the tool where needed and add additional features to assist with their specific requirements.

1.3 Approach

Within ERS, the Tradespace Analysis area has developed a web-based application called TradeAnalyzer, which allows users to apply various analysis techniques across their datasets. However, many customers across the DoD faced issues connecting to external sources.

To mitigate this issue, ERS developed a local Python edition of the application called TradeAnalysis. TradeAnalysis enables the customer to perform high-fidelity tradespace analysis and generate visualizations in a local environment.

TradeAnalysis v1.1 was designed specifically to enable users to perform their analytics needs via their local desktop environment.

1.4 Scope

The scope of this report will focus on instructing the user how to install TradeAnalysis v1.1, start the program, and understand the granular details of the underlying codebase.

2 Dependencies

The following sections outline the dependencies required to run the TradeAnalysis application, along with the recommended software for Python package management.

2.1 Programming language

Python is a high-level, interpreted programming language used commonly for general-purpose programming. Python is widely used in scientific computing, and has a core philosophy that focuses on readability, simplicity, and explicitness. Python is the primary language behind the TradeAnalysis application.

Python ≥ 3.X.X | https://www.python.org/downloads/

2.2 Python libraries

Multi-language library for data visualization creation. For this application, the Plotly Python library is used.

- Plotly | https://plot.ly/python/getting-started/

Python library for high-level mathematical functions.

- NumPy | http://numpy.org

Python library for data manipulation and analysis.

- Pandas | https://pandas.pydata.org/

A micro web framework written in Python and based on the Werkzeug toolkit and Jinja2 template engine.

- Flask | http://flask.pocoo.org/

Python library for symbolic mathematics.

- SymPy | http://www.sympy.org/en/index.html

2.3 Version control software (optional)

A version control system for tracking changes in computer files and coordinating work on those files among multiple people.

- Git | https://git-scm.com/

2.4 Browsers

Supported Browsers for the User Interface (UI).

- Google Chrome: https://www.google.com/chrome/browser/
- Apple Safari: https://support.apple.com/downloads/#safari
- Mozilla Firefox: https://www.mozilla.org/en-US/firefox/new/

2.5 Recommended software

This is the preferred method for Python package and dependency management. Most of the required dependencies for this application are included in the installation of Anaconda.

- Anaconda | https://www.anaconda.com/download/

3 Installing Dependencies

It is assumed that the user has met the following requirements before proceeding:

- The user has installed Python ≥ 3.X.X.
- The user has basic knowledge using macOS or Linux Terminal or Windows Command Prompt.
- The user has basic understanding with installing packages using either the Pip Installs Packages (Pip) package manager or the Anaconda package manager (Conda).

3.1 Verify Python is installed

For macOS or Linux, open Terminal. For Windows, open the Command Prompt.

Type the following command: `$ python -V`

If the following error message is received: *"python is not recognized as an internal or external command, operable program or batch file*," then Python was not installed in the path. Add Python to the path to use this program.

3.2 For macOS or Linux

- Open the file .bash_profile with your preferred text editor.

This file can be found at /Users/<user-name>/.bash_profile or ~/.bash_profile.

- Enter the following for standard Python:

export PATH="/path/to/python/package/location/:${path}"

- Enter the following for Anaconda Python:

export PATH="/anaconda/bin:$PATH"

- Save the file and start a new Terminal shell.

3.3 For Windows

- Access *System* from the Control Panel.
- Click on the *Advanced System Settings* tab.
- Click on the *Environmental Variables* button at the bottom of the screen.
- Click the *New* button under the *System Variables* section.
- Enter `PYTHONPATH` in the *Variable* field.
- Type the path for the Python modules in the value field.
- Click *OK* when you are finished setting the `PYTHONPATH` environment variable.

3.4 Anaconda package management

In order to run the commands below, open macOS or Linux Terminal or Windows Command Prompt.

The following command should be used to install packages using the following *Anaconda* package management system:

```
$ conda install <package>
```

Anaconda installation commands:

- Plotly: `$ conda install plotly`
- NumPy: `$ conda install numpy`
- Pandas: `$ conda install pandas`
- Flask: `$ conda install flask`
- SymPy: `$ conda install sympy`

For additional information about Anaconda package management, navigate to the following link: https://conda.io/docs/user-guide/install/index.html.

3.5 Pip package management

The following command should be used to install packages using the Pip package management system:

```
$ pip install <package>
```

Pip installation commands:

- Plotly: `$ pip install plotly`
- NumPy: `$ pip install numpy`
- Pandas: `$ pip install pandas`
- Flask: `$ pip install flask`
- SymPy: `$ pip install sympy`

For additional information about Pip package management, navigate to the following link: https://docs.python.org/3/installing/index.html.

4 Starting the Application

It is assumed that the user has completed the Installing the Dependencies before proceeding.

4.1 Download or clone the repository to your device

To download the files directly, navigate to the following link:
https://public.git.erdc.dren.mil/ers/TradeAnalysis/tree/master.

To clone the repository, perform the following:

For Windows, open Command Prompt. For macOS or Linux, open Terminal.

Enter the following command:

```
$ git clone
git@public.git.erdc.dren.mil:ers/TradeAnalysis.git
```

4.2 Start the flask server

For Windows, open Command Prompt.

Enter the following commands:

```
$ cd <path\to\TradeAnalysis\application\code>

$ python app.py
```

For macOS or Linux, open Terminal.

Enter the following command:

```
$ cd <path/to/TradeAnalysis/application/code>

$ python app.py
```

The Flask server should now be actively listening for a connection.

Open one of the supported browsers (i.e., Google Chrome, Firefox, or Safari) and navigate to the following URL: http://localhost:5000.

The port number can be changed by opening the file *app.py* and modifying the line `port = 5000` at the bottom of the file.

Once connected, the TradeAnalysis application should be ready to use.

5 Application Overview

This section covers the TradeAnalysis visualization types, code, and application structure. An overview of the underlying functions within the code are also provided.

5.1 Available visualization types

- Histogram | https://plot.ly/python/histograms/
- 2-Dimensional (2D) Scatter Plot | https://plot.ly/python/line-and-scatter/
- 3-Dimensional (3D) Scatter Plot | https://plot.ly/python/3d-scatter-plots/
- 3D Clustering Scatter Plot | https://plot.ly/python/3d-point-clustering/
- 3D Surface Plot | https://plot.ly/python/3d-surface-plots/
- Contour Plot | https://plot.ly/python/contour-plots/
- Heatmap | https://plot.ly/python/heatmaps/
- Heatmap Correlation Matrix | https://plot.ly/python/heatmaps/
- Parallel Coordinates | https://plot.ly/python/parallel-coordinates-plot/
- 2D Scatter Plot Pareto Front | https://plot.ly/python/line-and-scatter/
- Box Plots | https://plot.ly/python/box-plots/

5.2 Application folder structure

This section shows a high-level overview of the structure of the code. Each of the following will be discussed in detail in the following sections.

- TradeAnalysis/ (root)
  - graphs/
  - exports/
  - plot_modules/
  - static/
    - css/
    - images/
    - js/
    - vendor/
  - templates/
    - index.html
  - uploads/
  - app.py
  - .gitignore

5.3 TradeAnalysis/graphs/

The graphs folder contains all of the previously generated visualizations in Hypertext Markup Language (HTML) format. When a visualization is generated, a subfolder is created inside the graphs folder. The subfolder stores all HTML files for the specified visualization type.

```
TradeAnalysis/graphs/<visualization-
type>/html/<visualization.html>
```

For example, assume the generated visualization was a histogram and the dataset used to generate the visualization was `test.csv`. The output would be as follows:

```
TradeAnalysis/graphs/histogram/html/test.html
```

5.4 TradeAnalysis/plot_modules/

This folder stores the Python visualization code as separate modules. Each module is separated into its own file. This module structure was chosen for simplicity purposes.

The following three types of files are located within this folder: `__init__.py`, `<name_of_visualization.py>`, and `helper.py`.

- `__init__.py`
  - This file denotes that this folder is a Python package and allows the program to import these modules when necessary. These files are generally empty.
- `<name_of_visualization.py>`
  - This is the naming structure chosen for simplicity. For example, the Box Plots module is handled within the file `box_plot.py`.
  - Each module contains a function that requires a Tradespace object. The Tradespace object is outlined in the `TradeAnalysis/app.py` section.
- `helper.py`
  - This file contains functions frequently used by the various plot modules.
  - `def plot()`
    - Creates the HTML file for the path supplied.

  - `def default_layout()`
    - Sets the default layout of the plot figure.
  - `def html_directory()`
    - Creates the subfolder within graphs to store the HTML file.

5.5 TradeAnalysis/static/

This folder contains the JavaScript (JS), Cascading Style Sheets (CSS), and image files that assist with the creation of the UI.

5.6 TradeAnalysis/static/css/

- `bootstrap.min.css`
  - Twitter's Bootstrap handles element styling, such as table formatting, dropdowns, navigation, and other various stylistic features.
- `bootstrap.min.css.map`
  - This maps the Bootstrap source code to additional dependencies.
- `styles.css`
  - This is the custom CSS that handles additional stylistic choices, such as media screen minimum and maximum viewpoints, font choices, UI positioning, and the loading wheel.

5.7 TradeAnalysis/static/js/

- `vendor/`
  - The following files are open-source files used to handle various functionality within the application:
    - bootstrap.min.js
    - html5shiv.min.js
    - jquery.min.js
    - popper.min.js
    - respond.min.js
- `get.js`
  - This uses Asynchronous JavaScript and Extensible Markup Language (XML) (AJAX) to get specific properties and elements from the Flask server to populate the UI.
  - `function getObjectives(callback)`
    - This gets the list of available objectives (the column headers) and returns them to the supplied callback function.
  - `function histogram(objectives)`

        - This function creates a modal for the histogram option and populates the specified containers with the objectives. This creates the dropdowns and other features shown in the modal.
    - `function <visualization-type>(objectives)`
        - The remaining functions follow the pattern above, in which a modal is created for the visualization type. Each modal is populated with the supplied objectives. This creates the dropdowns and other features shown in the modal.
    - The following is a sample use case:
        - User clicks button for *Histogram*.
        - `getObjectives(histogram)` is called.
        - Passed in as `getObjectives(callback)` where `callback == histogram()`.
        - AJAX request sent to `/objectives`.
        - The Flask server returns back a list of the objectives to the function. This is now handled by `callback`, which is a function passed in as a parameter.
        - `callback(data)` is called where `callback(data) == histogram(data)`.
        - `histogram(objectives)` is called, which triggers `createModal("histogram", "Histogram")`.
        - The histogram modal is now created and the objectives are appended to the x-axis dropdown selection.
- `globals.js`
    - This file hosts all of the global variables, which are variables accessible by all other scripts within the same domain. When `var` is used, that variable is declared as a global variable. For local variable declaration, `let` is used.
        - Local: `let myVariable;`
        - Global: `var myVariable;`
- `helper.js`
    - This file contains helper functions that are used interchangeably between the other JS files.
    - `function validFile(file)`
        - This verifies that the file extension is a supported type.
    - `function hideModal(modal=null)`
        - This hides the supplied modal argument. If no modal is supplied, the currently displayed modal is hidden.
    - function `showModal(modal)`
        - This displays the supplied modal identity (ID).

- `function plot(type)`
  - This routes the plot type to the Flask app. For example, `plot("histogram")` will plot a visualization of type histogram.
- `function checkbox(id)`
  - If a checkbox is selected, it sets the value as True; otherwise, it sets the value as False.
- `function selectAll(checkbox)`
  - This allows the user to check or uncheck all of the checkboxes in the list with one click.
- `function isNumber(e)`
  - This prevents the user from entering non-numerical values.
- `function checkUploadStatus()`
  - If the user uploads a new file, this function will clear out any data not found within the newly selected file. This prevents data from mixing between different files.
- `function createModal(id, title)`
  - This serves as a modal template creator for each of the visualization types, allowing dynamic creation of modals.
- `function resetModal(btn)`
  - This reverts the modal back to its original state.
- `function removeObjective(btn)`
  - This allows the user to remove the selected objective.

- `listeners.js`
  - Each function within this file is tied to an event listener delegated by the document.
  - `$(document).ready()`
    - This triggers once the page has loaded. This is designed to force the tradespace data to reset after the user refreshes the page.
  - `$(document).on("change", <id>, function())`
    - This triggers once the state of the supplied ID changes. For example, when the user chooses the *Upload File* button, the page "listens" for change in the file.
  - `$(document).on("click", <id>, function())`
    - This triggers once the user clicks on the element of the supplied ID.
- `select-previous.js`
  - This file uses AJAX to get the list of available files from the *uploads* folder. If files are available, they are returned as a list type to the user.

  - `function getPreviousFiles()`
    - A post request is made to the Flask server. If the user has previously generated visualization(s), a list of references to the files will be returned to the function.
    - A dynamically created table is populated with the list of values and injected into the HTML page for the viewer to see.
  - `function selectPreviousFile(file)`
    - If a file has been placed in the *uploads* folder, the user can select the file from the Select Previous File section. The selected file is passed to the Flask server, where the tradespace object is created in memory.
    - The file variables are updated, and `checkUploadStatus()` is called to reset all previous modals to their original states if the file changed. Clearing the modals prevents mismatched data between two files.
- `set.js`
  - `function setHistogram()`
    - This function extracts the values held within the histogram modal. The stored data within the elements is extracted, which is then passed to the Flask server to route to the Python histogram handler.
    - Once the information is set and there are no issues, the `plot("histogram")` function is executed to generate a histogram based on the set values.
  - `function setBoxPlot()`
    - This function extracts the values held within the box plot configuration. Since there may be more than one objective in the box plot that needs to be plotted, a loop must be used to obtain all the elements.
    - The stored data within the elements is extracted, which is then passed to the Flask server to route to the Python box plot handler. Once the information is set, `plot("box_plot")` is called to generate the box plot based on the set values.
  - `function setScatter2d()`
    - This function extracts the values stored within the elements of the 2D scatter plot modal. The *x* and *y* axes are extracted, along with the optional title of the visualization.
    - The stored data within the elements is extracted, which is then passed to the Flask server to route the information to the Python scatter plot handler.

- Once the information is set, `plot("scatter_plot_2d")` is called to generate the scatter plot based on the set values.

o `function setScatter3d()`
- This function extracts the values stored within the elements of the 3D scatter plot modal. The *x*, *y*, and *z* axes are extracted, along with the optional title of the visualization.
- The stored data within the elements is extracted, which is then passed to the Flask server to route the information to the Python scatter plot handler.
- Once the information is set, `plot("scatter_plot_3d")` is called to generate the scatter plot based on the set values.

o `function setClustering3d()`
- This function extracts the values stored within the elements of the 3D clustering plot modal. The *x*, *y*, and *z* axes are extracted, along with the optional title of the visualization.
- The stored data within the elements is extracted, which is then passed to the Flask server to route the information to the Python clustering plot handler.
- Once the information is set, `plot("clustering_3d")` is called to generate the clustering scatter plot based on the set values.

o `function setSurface3d()`
- This function extracts the values stored within the elements of the shared surface plot 3d, contour, and heatmap modal.
- Since each of the following takes the data in the same format, they share the same modal and elements. A radio button with element ID type is used to distinguish them.
- The stored data within the elements is extracted, which is then passed to the Flask server to route the information to the Python handler.
- The handler is chosen based on the type (i.e., `surface_plot_3d`, `contour`, or `heatmap`).
- Once the information is set, `plot(<type>)` is called to generate the visualization.

o `function setParallelCoordinates()`
- This function extracts the values stored within the elements of the parallel coordinates modal.
- The stored data within the elements is extracted, which is then passed to the Flask server to route the information to the Python parallel coordinates handler.

        - Once the information is set, `plot("parallel_coordinates")` is called to generate the parallel coordinates plot based on the set values.
    - `function setParetoFront()`
        - This function extracts the values stored within the elements of the pareto front modal.
        - The stored data within the elements is extracted, which is then passed to the Flask server to route the information to the Python pareto front handler.
        - Once the information is set, `plot("pareto_front")` is called to generate the pareto front plot based on the set values.
    - `function setCorrelationMatrix()`
        - This function extracts the values stored within the elements of the correlation matrix modal.
        - The stored data within the elements is extracted, which is then passed to the Flask server to route the information to the Python correlation matrix handler.
        - Once the information is set, `plot("correlation_matrix")` is called to generate the correlation matrix based on the set values.
- `upload.js`
    - `function uploadFile()`
        - This function handles the upload process for the application. During a normal form upload, the page refreshes. This would be problematic, as it would reset the data in memory on the Flask server side.
        - To prevent this, `event.preventDefault()` is used to halt the normal upload procedure.
        - Instead, AJAX is used to asynchronously upload the file. The file reference is passed to the Flask server. The file is copied into the uploads folder.
        - If an unsupported file is uploaded, an error message will be returned. The file selected is now set into memory to be used for the available visualizations.
- `view-export.js`
    - `function getVisualizations()`
        - This function creates a modal with a populated table of all previously generated visualizations.
        - The modal is emptied beforehand to prevent duplicate data.

- A get request to get the list of visualizations must be made. If a list of visualizations is sent back, a dynamically created table is made with the list of visualizations.
- Once the table is made, it is appended to the page for the user to see.

o `function viewVisualizations()`
- This function displays the user-selected visualization.
- In order to keep up with the individual files and their respective directories, the following naming structure was chosen for the element IDs in the visualization:
  ~ `column-<visualization type>-row-<row number>.`
  ~ For example, if the user wants to select the 3rd visualization in the histogram dropdown, the element ID would be as follows:
  - column-histogram-row-2 (counting from 0).
- Once this information is placed in the correct format, this is sent to the Flask server, so that the correct file can be sent back to the page.
- Once the file is sent back, the visualization is appended to the page for the user to see.

o `function appendToDiv(visualization)`
- This function appends the provided element to the page.
- The div containing the visualizations is cleared to prevent overlapping visualizations.
- Afterwards, the provided element is appended to the page for the user to view.

o `function deleteVisualizations(choice)`
- This function deletes all of the checked visualizations within the table list. A confirmation message is displayed to verify the user's action with the deletion process.
- If confirmed, a deletion request is sent to the Flask server, where the server-side deletion process is handled.
- After deletion, if the table entry is empty (i.e., there are no more visualizations in the list), then the entire directory is deleted.

o `function exportVisualizations()`
- This function sends a request through the Flask server to compress the folder with all the previously generated visualizations.
- The user can select either a zip or tar file.

- Once completed, a new folder called exports is created and the visualizations are stored within it.

5.8 TradeAnalysis/static/images/

This folder stores all of the images used to make aspects of the UI. The images are referenced within the custom CSS and HTML files.

5.9 TradeAnalysis/static/templates/

In order to use index.html as a traditional HTML file, it must be placed in a folder called templates. This is mandated by Flask in its overall structure.

- index.html
  - This is the linker file that brings together all of the UI elements within the application. In the <head> tags at the top of the file, the CSS files are imported.
  - In the bottom of the <body> tags, the JS sources are imported.
  - The remaining portion of the HTML is broken down as follows:
    - Navigation bar
      - ~ The navigation bar allows the user to have clickable links to upload a file and view and/or export previous visualizations.
    - Background / Icons
      - ~ The main UI options reside in this section. These options allow the user to upload or select a file, view previous visualizations, or create new visualizations.
    - Upload File Modal
      - ~ The UI elements for uploading a file are handled here.
    - Select Previous File Modal
      - ~ The list of previously selected files is contained here.
    - View / Export Visualizations Modal
      - ~ This is the container for the previously generated files.
    - Select Visualization Modal
      - ~ This is the list of available visualization types for the user to select.

5.10 TradeAnalysis/uploads/

This folder stores each file the user has previously uploaded.

5.11 TradeAnalysis/.gitignore

This file helps structure files in GitLab. In most cases, this file can be ignored.

Additional information about this file can be found here:

- https://git-scm.com/docs/gitignore
- https://github.com/github/gitignore

5.12 TradeAnalysis/app.py

This file serves as the Flask local server, which routes interactions between the UI and Python application code.

- `app = Flask(__name__)`
  - The object app now serves as the Flask server object. This object works as the handler for all of the routing, configuration, etc.
- `@app.route(<place-to-route>, methods=["POST", "GET"])`
  - The @ symbol is a decorator in Python. `@app.route` is a Flask decorator that listens for a request to be made. The function immediately following the route decorator is called.
- `@app.errorhandler(<value>)`
  - This returns the user back to *index.html* after an error.
- `return jsonify(<value>)`
  - When a request is made to the server, a value must be returned. If not, the application will error out.
- `def allowed_file(filename)`
  - This verifies that the supplied file is a supported file type in the application.
- `def index()`
  - Once the application starts, Flask routes the index.html page to initiate the UI. This is the main page of the application.
- `def refresh()`
  - After the page refreshes, the tradespace object resets. This prevents the user from accidently mixing data between different datasets.
- `def upload()`
  - This function handles the file uploading process.
  - `secure_filename()` handles the pathing to the file. The path to uploads must be linked before moving the file there.

    - `pathlib.Path()` will handle the creation of the directory. If it does not exist, this will create it; otherwise, it skips this process. Afterwards, the file is saved to the uploads folder for later use.
- `def create_tradespace_object(file)`
    - This function handles the creation of a global tradespace object. Each tradespace object has a variety of attributes. These attributes change depending on the file selected and the visualization chosen.
    - `tradespace.backup()` is called after creation to back up the objectives. Since objectives change from visualization to visualization, they must revert back to the original values before modifying them again.
- `def get_objectives()`
    - This function extracts and returns the list of the objectives from the tradespace object.
- `def verify()`
    - This function verifies that the user has selected a file and that the tradespace object is not empty. If it is not empty, a success message is returned. Otherwise, a 204 error code is sent back to denote that the user must create the object.
- `def invalid_dtype(case, objectives)`
    - This function determines whether the currently selected data is compatible with the desired visualization. The tradespace data is handled with Pandas, which has a built-in type checker. There are currently two cases for the data types:
        - Case 0
            - ~ Visualizations do not have constraints on the data because the visualizations support all available types.
        - Case 1
            - ~ Visualizations do not support strings or any non-numerical data. If types *string* or *object* are found within the data of case 1, an error message is returned to the user.
- `def set_histogram()`
    - This sets the user-supplied histogram data. It verifies that the data supplied is valid.
    - If valid, the *x* axis and title data are set within the tradespace object.
- `def set_box_plot()`
    - This sets the user-supplied box plot data. It verifies that the data supplied is valid. If valid, the list of supplied objectives and the title data are set within the tradespace object.
- `def set_correlation_matrix()`

  - This sets the user supplied correlation matrix data. It verifies that the data is valid. If valid, the list of supplied objectives and the title data are set within the tradespace object.
- `def set_scatter2d()`
  - This sets the user supplied scatter plot data. It verifies that the data supplied is valid. If valid, the *x* and *y* axes, along with the title data are set within the tradespace object.
- `def set_scatter3d()`
  - This sets the user-supplied scatter plot data. It verifies that the data supplied is valid. If valid, the *x*, *y*, and *z* axes, along with the title data are set within the tradespace object.
- `def set_clustering()`
  - This sets the user-supplied clustering plot data. It verifies that the data supplied is valid. If valid, the *x*, *y*, and *z* axes, along with the title data are set within the tradespace object.
- `def set_surface3d()`
  - This sets the user-supplied surface plot, contour, or heatmap data. It verifies that the data supplied is valid. If valid, the *x*, *y*, and *z* axes, along with the title data are set within the tradespace object.
- `def set_parallel_coordinates()`
  - This sets the user-supplied parallel coordinates plot data. If valid, the list of objectives and other optional settings are set within the tradespace object.
- `def set_pareto_front()`
  - This sets the user-supplied pareto front plot data. If valid, the list of objectives and the other optional settings are set within the tradespace object.
- `def plot()`
  - This routes the plot type to the correct plot module.
  - After completion of a plot, the tradespace is reset to its original state so that it can be modified by another visualization type if needed.
  - The HTML file produced by the visualization is returned to the application page and rendered for the user to see.
  - The Flask module `send_file(path)` is used to send the file to the page.
- `def export()`
  - This archives the list of previous visualizations using a module named `shutil.make_archive()`.

    - The path to the directory must be passed, and the module will recursively handle archiving the data.
    - It presents two options: zip files or tar files.
    - This selection is chosen by the user, and after the archive process is complete, the file is placed in the exports folder.
- `def get_visualizations()`
    - The following function returns back a list of previously generated visualizations. This is done by concatenating the path to the graphs folder and obtaining the HTML files associated with each visualization type.
- `def view_previous()`
    - The following function accepts a file selection from the user and returns the chosen file to be rendered on the application page.
    - This is done by navigating to the location of the file and using the built-in Flask `send_file()` module.
- `def delete_previous()`
    - The following function accepts a file argument from the user and deletes the selected file(s).
    - In order to remove the file, the Python library operating system (os) must be used. Before deleting a file, the file permission must be set to 777. In Python 3, this is done by `os.chmod(file, 0o777)` and `os.remove(file)`.
    - After the deletion of the file(s), there is a check to see if the folder is empty. If it is empty, `shutil.rmtree(path)` is used to recursively remove the folder.
- `def get_previous_uploads()`
    - The following function returns a list of previously uploaded files. This allows the user to select a file from the uploads folder instead of having to re-upload each time.
- `def select_previous()`
    - The following function sets the user-supplied file as the current tradespace object. If the file is valid, the `create_tradespace_object()` function is executed.
- `class TradeSpace`
    - The tradespace object stores all the attributes from the data files, user-chosen specifications, etc.
    - `def __init__(self, filename)`
        * This is the constructor for the tradespace object. This accepts the file from the user and sets default attributes. These attributes can later be modified.

    - `def config_init(self, filename)`
        * The following module separates the filename from the extension. The extension determines the method in which to read the file in as a dataframe with Pandas.
        * If it is a CSV file, then `pandas.read_csv()` must be used. Otherwise, `pandas.read_excel()` is used.
        * The file is stored in memory as a Pandas dataframe for data manipulation.
        * The column headers are used as the objectives and stored as `self.objectives` within the object. These are later used to select which columns the user would like to visualize.
- `def backup(self)`
    - The following module backs up the list of objectives and original title. To prevent data loss the objectives are modified throughout each visualization, they must return back to the original state between each visualization type.
- `def revert(self)`
    - The following module reverts to the backup of the objectives and title. This is done to allow the user to select from all of the original values.

6 Conclusion

The ERS Tradespace Analysis area developed a web-based application called TradeAnalyzer, which allows users to apply various analysis techniques across their datasets. However, many customers across the DoD faced issues while connecting to external sources.

To mitigate this issue, ERS developed a local Python edition of the application called TradeAnalysis. TradeAnalysis enables the customer to perform high-fidelity tradespace analysis and generate visualizations in a local environment. This application allows users to quickly generate, export, and share offline, standalone, and custom visualizations to further assist the DoD acquisition process.

This report covered the dependencies required to run the application, along with supplying external links for additional information about those dependencies. Python package management software was recommended to the user to make the installation process easier. Finally, a breakdown of the application code was covered from a high-level perspective, giving the user insight to the underlying code functionality.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. REPORT DATE *(DD-MM-YYYY)*	2. REPORT TYPE	3. DATES COVERED *(From - To)*
September 2019	Final	

4. TITLE AND SUBTITLE

TradeAnalysis v1.1 Technical Guide

5a. CONTRACT NUMBER

5b. GRANT NUMBER

5c. PROGRAM ELEMENT NUMBER

6. AUTHOR(S)

Joshua Q. Church

5d. PROJECT NUMBER

5e. TASK NUMBER

5f. WORK UNIT NUMBER
92L5D8

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

U.S. Army Engineer Research and Development Center
Information Technology Laboratory
3909 Halls Ferry Road
Vicksburg, MS 39180-6199

8. PERFORMING ORGANIZATION REPORT NUMBER

ERDC/ITL SR-19-18

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)

Headquarters, U.S. Army Corps of Engineers
Washington, DC 20314-1000

10. SPONSOR/MONITOR'S ACRONYM(S)

11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION / AVAILABILITY STATEMENT
Approved for public release; distribution is unlimited.

13. SUPPLEMENTARY NOTES

14. ABSTRACT

The Engineered Resilient Systems (ERS) program focuses on the acquisition process for the Department of Defense (DoD). Within ERS, the Tradespace Analysis area has developed a web-based application called TradeAnalyzer, which allows users to apply various analysis techniques across their datasets. However, many customers across the DoD faced issues connecting to external sources.

To mitigate this issue, ERS developed a local Python edition of the application called TradeAnalysis. TradeAnalysis enables the customer to perform high-fidelity tradespace analysis and generate visualizations in a local environment. This application allows users to quickly generate, export, and share offline, standalone, and custom visualizations to further assist the DoD acquisition process.

This report will cover the required dependencies to run the TradeAnalysis application, the recommended software for Python package management, and a breakdown of the underlying code.

15. SUBJECT TERMS

Systems engineering--Decision making
Big data
Quantitative research
Computer programs

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			19b. TELEPHONE NUMBER *(include area code)*
Unclassified	Unclassified	Unclassified	SAR	33	