Additional Libraries:

This comprehensive guide outlines the additional libraries employed within the Python environment for the Cookbook. These libraries are designed for specific roles by organizing and presenting relevant data. Allowing users to efficiently access and manipulate the required information, enhancing overall productivity.

The Cookbook utilizes the following libraries, which are distinct from the libraries and packages present within ArcPy. These libraries are incorporated into all the notebooks that constitute the Cookbook. Before utilization, several libraries need to be imported as they are not built-in functions: **os**, **sys**, **shap**, **tensorflow**, and **tensorflow_decision_forest**.

- **os** - The **os** library is responsible for file handling and creation, ensuring that data display is efficient when necessary.
- **sys** - The **sys** library enables manipulation of the program's runtime environment through diverse parameters and functions.
- **shap - SHAP** is an interpretability tool that helps evaluate each feature's contribution toward predictions.
- **tensorflow** as ***tf*** - The **TensorFlow** platform enhances workflow by providing automated data management, monitoring performance metrics, facilitating model retraining, and tracking model performance.
- **tensorflow_decision_forests** as ***forest*** - **TensorFlow Decision Forests** serves as a library designed for the training, execution, and interpretation of decision forest models in the TensorFlow framework.

Import:

- **Re** - A regular expression (also known as **re**) is a pattern that defines a set of strings. A given string is considered to match the regular expression if it conforms to the pattern specified by the regular expression.
- **Pandas as pd -** The **Pandas** library is used to manipulate, analyze, and model data sets.
- **Modin.pandas** - **Modin** distributes and computes data faster by using API.
- **Numpy as np** - With the **NumPy** library, data can undergo various transformations, including processing and manipulation, enabling users to handle and visualize large datasets effectively.
- **Datetime** - The **datetime** library is used to control and modify date and time-related data.
- **Fnmatch** - The **fnmatch** module compares filenames against a pattern and returns True or False according to the name.
- **String** - The **string** method allows strings or data to be split when needed.
- **Warnings** - **Warnings** can inform the user of the state of the code or where an error has occurred.
- **Seaborn** - **Seaborn** is a Python library that creates statistical graphics.
- **Matplotlib.pyplot** - Creates data visualization (i.e., graphs)
- **Subprocess** - The Python **Subprocess** module creates new processes and runs new code and applications.
- **Xarray** - **Xarray** is an open-source Python package that makes working with labeled multi-dimensional arrays more efficient and easier to comprehend.
- **Xgboost** - The **XGBoost** Python package can be used to efficiently and effectively implement the gradient-boosting ensemble algorithm for classification and regression problems. XGBoost leverages machine learning techniques to enhance predictive accuracy by aggregating predictions from multiple models.

From __ import __:

- From **numpy** import **mean, std:**
    - Import **mean** - The **NumPy mean** function calculates the average or mean value from all the elements in a given NumPy array.
    - Import **std** - The std() function in **NumPy** computes the **standard deviation** of a given array. The function accepts an array as input and outputs the standard deviation as a numerical value.
- From **sklearn** import **datasets, ensemble**:
    - Import **datasets** - **Sklearn datasets**, generated within the Python-based sklearn library, serve as synthetic datasets for various purposes. These datasets are utilized for machine learning algorithm testing, benchmarking, and developmental processes.
    - Import **ensemble** - The **sklearn.ensemble** module allows for optimizing the machine learning model's arbitrary differentiable loss function.
- From **natsort** import **natsorted** - **natsorted()** sorts strings and numbers separately.
- From **sklearn.metrics** import **accuracy_score, r2_score**:
    - Import **accuracy_score** - The **accuracy_score** function calculates the accuracy of a prediction.
    - Import **r2_score** - The **r2_score**, or regression score function, measures how well a linear regression model fits the data.
- From **sklearn.metrics** import **mean_squared_error** - **MSE** evaluates the closeness between the regression line and the set of data points. A lower MSE indicates that the regression line fits the data better.
- From **sklearn.preprocessing** import **StandardScaler** - The **StandardScaler** removes the mean or average value and scales to the unit variance.
- From **sklearn.model_selection** import **train_test_split** - The sklearn.model_selection package allows arrays or matrices to be split into random subsets for data to be processed and made valid.
- From **contextlib** import **redirect_stdout** - The **redirect_stdout** tool enables the redirection of a program's output to a different file, which is useful in troubleshooting issues. It facilitates debugging and enhances the versatility of existing functions whose output is inherently directly to stdout.
- From **tensorflow.python.client** import **device_lib** - Provides access to the TensorFlow device library.
- From **xgboost** import **XGBRegressor** - **XGBRegressor** offers an interface compatible with scikit-learn, allowing seamless integration in machine learning pipelines. It features a range of hyperparameters specifically tailored to control and optimize the model training process, ensuring precise model tuning and enhanced predictive performance.

Requirements.txt:

This guide provides detailed instructions on installing Python packages in Linux to generate a requirements.txt file. The requirements.txt file serves as a list of packages necessary for a project. Utilizing the requirements.txt file allows all project packages to be installed concurrently. Additionally, this file assists users in tracking the models and packages employed in their projects, ensuring clarity and organization.

Before installing the Python packages, the user needs to create a dedicated directory to house all the required packages needed for the text file. This can be done by applying the following command in the terminal: **mkdr *directory_name***

How to install Python packages in a Linux terminal to use for requirements.txt:

- **Note**: Many of the packages have a different name in Linux. As a preliminary step, it is advisable to search for the package's name. This search will provide the specific name used in the Linux operating system.
    - To search a package, use: **sudo apt search *name_of_package***
- There are several options for installing the packages in Linux:
    - To install a package, use: **sudo apt install *name_of_package***
        - ***sudo apt-get install name_of_package*** can also work for installation.
    - To install multiple packages, use: **sudo apt install *package_name1 package_name2 …***
- To list the files installed from a specific package, use: **dpkg -L *name_of_package***
- The following command is used to remove a file: **sudo dpkg -r *name_of_package***

How to create a Python requirements.txt file:

- When working in a virtual environment, use the command below to create the requirements.txt file in the correct format.
    - **pip freeze > requirements.txt**
        - *pip freeze* outputs a list of all installed Python modules with their versions.
    - **pip install -r requirements.txt**
        - The installed packages will appear in the text file.
- A new file will be created in the directory the user is working in when running the code. This file will list the different packages and modules in your virtual environment.

How to maintain a requirements.txt file:

- Creating a requirements.txt file also requires frequent maintenance. For this, any outdated packages will need to be upgraded using the following command:
    - To generate a list of out-of-date packages, use: **pip list - -outdated**
- Users can update an individual package by using: **pip install -U *name_of_package***
- If the user wants to upgrade all items in the file, then employ: **pip install -U -r requirements.txt**
- To update the requirements.txt file, the user can utilize the following command:
  pip freeze > requirements.txt
    - For the following two production branches run:
        - **git commit**
        - **git push**

Creating Python virtual environment:

A Python virtual environment is an isolated working copy of Python that allows the user to work on a specific project without affecting other projects. This enables multiple side-by-side installations of Python, one for each project. To create a Python virtual environment in a Linux terminal, follow these steps:

1. Open your terminal
2. Install Python (if the user hasn't done so already)
3. Use the venv module to create a virtual environment.
4. Activate the virtual environment

How to Install a Virtual Environment:

Virtualenv enables the isolation of Python environments. It aids in the creation of separate workspaces for various Python projects. Each environment resembles a self-contained directory tree, comprising its own Python installation and additional packages required for the project.

- To set up a virtual environment for Python, the user can utilize *virtualenv*, which can be installed by executing the following command in their terminal.
    - **pip install virtualenv**
    - If pip is not present on the user's system, it is advisable to execute the following command:
        - **sudo apt-get install python-pip**
- Create a new directory and launch the terminal. Navigate through the user's computer in the terminal using the *cd* command. The *cd* command allows users to move to a different directory or return to the parent directory. To move through multiple folders and reach a specific location, users can use: **cd ~/path/to/another/folder**
    - To create a new directory, use the command:
        - **mkdir *newdir***
    - For navigation within the terminal, use the command:
        - **cd *~/newdir***
- To create the virtual environment, use the following command:
    - **python3 -m *newdir venv_name***
    - For other versions of Python, use the following command:
        - **virtualenv *direct_name***
- To activate the environment, use the following command:
    - **source *venv_name*/bin/activate**
- Once the virtual environment is activated, the shell will recognize the current virtual environment. To disable the virtual environment and return to the default environment, execute the following command:
    - **deactivate**