**Introduction to R and RStudio**

R is a statistical programming language. It is a powerful tool for data processing and manipulation, statistical inference, data analysis, and machine learning algorithm. Based on 2017 analysis, it was found that R is used most by academics, healthcare, and the government. R supports importing of data from different sources like flat files, databases, web, and statistical software such as SPSS and STATA. R is a preferred language for some data scientists because R functions are easy to use. It is also known for producing great visualizations and contains packages to handle data analysis without the need to install additional libraries. A popular integrated development environment for developing and running the R language source code and programs is RStudio. It improves and increases productivity with the R language. R studio includes: a syntax-highlighting editor that supports direct code execution and a place where you can keep a record of your work, a Console for typing R commands, a workspace and History tab that shows the list of R objects you created during your R session and the history of all previous commands, and finally, Files, Plots, Packages, and Help tabs. The Files tab shows files in your working directory. The Plots tab displays the history of plots you have created. You can also export plots to PDF or image files. The Packages tab displays external R packages available on your local computer. And, the Help tab provides help on R resources, R studio support, packages, and many more. If R is your tool choice for data science, here are some popular R libraries available in the Data Science community: dplyr for manipulating data, stringr for manipulating strings, ggplot for visualizing data, and caret for machine learning. To get you up and learning quickly, we have provided you with an R Studio virtual environment as part of the Skills Network Labs. This virtual lab environment is designed to assist you to easily practice what you learn in the course and skip the need to create an account or download or install anything.

In this video, you learned the capabilities of R and its uses in Data Science, the RStudio interface for running R codes, and popular R packages for Data Science.

**Plotting in RStudio**

With the influx of data, one of your many jobs as data scientists is to produce insights using visualizations. R has different packages for data visualization that you can use based on your requirement. To install these packages in your R environment, use the install.packages and the package name command. Examples of R packages include the following. ggplot is used for data visualizations such as histograms, bar charts, scatterplots, and so on. It allows adding layers and components to a single visualization. Plotly is used for web-based data visualizations that can be displayed or saved as individual HTML files. Lattice is used to implement complex, multi-variable data sets. It is a high-level data visualization library that can handle graphics without customizations. And, Leaflet is used for creating interactive plots. R has inbuilt functions to create plots and visualization. For example, you can create a plot using the definition shown here. The plot function returns a scatterplot of the values vs. the index. You can also add lines to the function and a title to make the visualization easier to read and understand. To add a line, you specify the type and to add a title, you select the title function. In the plot, you have added a line and a title. You can create informative visualizations using the ggplot library of R. It can handle complex requests by adding layers to plots using different functions and arguments. For example, to create a scatter plot, let’s use the inbuilt dataset Mtcars. You will first read the ggplot library into the memory using the library function. Next, use the ggplot function on the data frame MTcars, specify the X axis as miles per gallon and the Y axis as weight. Then add the geom point function to specify a scatter plot; otherwise, it will return an empty plot. The output will be an easier-to-read plot. In addition, you can add titles and change the axis name by using the Ggtitle argument and the lab’s argument to specify appropriate names for both axes. The result will be a graph with meaningful titles. In the lab, you will recreate the graphics with ggplot and the extension library called GGally. GGally extends ggplot by adding several functions to reduce the complexity of combining geometric objects with transformed data. In this video, you learned about: Popular data visualization packages in R, Plotting with the inbuilt R plot function, Plotting with ggplot, Adding titles and changing the axis names using the ggtitle and lab’s function.

**Overview of Git and GitHub**

A version control system allows you to keep track of changes to your documents. This makes it easy for you to recover older versions of your document if you make a mistake, and it makes collaboration with others much easier. Here is an example to illustrate how version control works. Let’s say you’ve got a shopping list and you want your roommates to confirm the things you need and add additional items. Without version control, you’ve got a big mess to clean up before you can go shopping. With version control, you know exactly what you need after everyone has contributed their ideas.

Git is free and open source software distributed under the GNU General Public License. Git is a distributed version control system, which means that users anywhere in the world can have a copy of your project on their own computer. When they’ve made changes, they can sync their version to a remote server to share it with you. Git isn’t the only version control system out there, but the distributed aspect is one of the main reasons it’s become one of the most common version control systems available. Version control systems are widely used for things involving code, but you can also version control images, documents, and any number of file types. You can use Git without a web interface by using your command line interface, but GitHub is one of the most popular web-hosted services for Git repositories. Others include GitLab, BitBucket, and Beanstalk. There are a few basic terms that you will need to know before you can get started. The SSH protocol is a method for secure remote login from one computer to another. A repository contains your project folders that are set up for version control. A fork is a copy of a repository. A pull request is the way you request that someone reviews and approves your changes before they become final. A working directory contains the files and subdirectories on your computer that are associated with a Git repository. There are a few basic Git commands that you will always use. When starting out with a new repository, you only need create it once: either locally, and then push to GitHub, or by cloning an existing repository by using the command "git init".

"git add" moves changes from the working directory to the staging area. "git status" allows you to see the state of your working directory and the staged snapshot of your changes. "git commit" takes your staged snapshot of changes and commits them to the project. "git reset" undoes changes that you’ve made to the files in your working directory. "git log" enables you to browse previous changes to a project. "git branch" lets you create an isolated environment within your repository to make changes. "git checkout" lets you see and change existing branches. "git merge" lets you put everything back together again. To learn how to use Git effectively and begin collaborating with data scientists around the world, you will need to learn the essential commands. Luckily for us, GitHub has amazing resources available to help you get started. Go to try.github.io to download the cheat sheets and run through the tutorials. In the following modules, we'll give you a crash course on setting up your local environment and getting started on a project.

**Introduction to GitHub**

Linux development in the early 2000’s was managed under a free-to-use system known as BitKeeper. In 2005, BitKeeper changed to a for-fee system which was problematic for Linux developers for many reasons. Linus Torvalds led a team to develop a replacement source-version control system. The project ran in a short a timeframe and the key characteristics were defined by a small group. These include: Strong support for non-linear development. (Linux patches were then arriving at a rate of 6.7 patches per second) Distributed development. Each developer can have a local copy of the full development history. Compatibility with existing systems and protocols. This was necessary to acknowledge the diversity of the Linux community. Efficient handling of large projects. Cryptographic authentication of history. This makes certain that distributed systems all have identical code updates. Pluggable merge strategies. Many pathways of development can lead to complex integration decisions that might require explicit integration strategies. What is special about the Git Repository model? Git is designed as a distributed version-control system. Primarily focused on tracking source code during development. Contains elements to coordinate among programmers, track changes, and support non-linear workflows. Created in 2005 by Linus Torvalds for distribution of Linux kernels. Git is a distributed version-control system that is used to track changes to content. It serves as a central point for collaboration with a particular focus on agile development methodologies. In a central version control system, every developer needs to check out code from the central system and commit back into it. As Git is a distributed version control, each developer has a local copy of the full development history, and changes are copied from one such repository to another. Each developer can act as a hub. When Git is used correctly, there is a main branch that corresponds to the deployable code. Teams can continuously integrate changes that are ready to be released and can simultaneously work on separate branches in between releases. Git also allows centralized administration of tasks with access-level controls for each team. Git can co-exist locally such as through the GitHub Desktop client or it can be used directly through a browser connected to the GitHub web interface. IBM Cloud is based on sound and established open-source tools including Git repositories, often called repos. GitHub is an online hosting service for Git repositories. GitHub hosted by a subsidiary of Microsoft. GitHub offers free, professional and enterprise accounts. As of August 2019, GitHub had over 100M repositories. A Repository is: A data structure for storing documents including application source code. A repository can track and maintain version-control. GitLab is a complete DevOps platform, delivered as a single application. GitLab provides access to Git repositories, controlled by source code management. With GitLab, developers can: Collaborate, reviewing code, making comments and helping to improve each other’s code. Work from their own local copy of the code. Branch and merge code when required. Streamline testing and delivery with Built-in Continuous Integration (CI) and Continuous Delivery (CD). In this video, you learned: GitHub is the online hosting service for Git repositories. Repositories store documents including application source code and enable contributors to track and maintain version-control. What is special about the Git Repository model? Git is designed as a distributed version-control system. Primarily focused on tracking source code during development. Contains elements to coordinate among programmers, track changes, and support non-linear workflows.

**GitHub Repositories**

Signing up for a free, personal account on GitHub is quick and easy. Start at the GitHub site, https://github.com You’ll need to choose a username, enter your email address and select a password, then click Sign up for GitHub. Next, you’ll have a short test to prove that you’re a person. Click Verify and solve the puzzle presented. When you’re done, click join a free plan and then you’ll be taken to a screen where you can select the type of account – most likely a free, personal account is all that you’ll want. Choose to set up a personal, free account, which is the default. GitHub asks some questions about your work, programming experience and interests. You can skip these if you want. Finally, you’ll have to respond to an email that you receive which proves that you linked to GitHub from an account that you access GitHub provides you with some starting points. You can choose to create a repository or an organization, or you can take the Introduction to GitHub course. Remember, a repository is a data structure for storing documents including application source code which tracks and maintains version-control. An organization is a collection of user accounts that owns repositories. Organizations have one or more owners, who have administrative privileges for the organization. Or you can skip this for now and get straight to work. GitHub provides many resources to help you work effectively. When you have time, read the GitHub guide. The heart of a Git-based project is the repository. This contains all your code and the related artifacts, including things like: A README file to describe the purpose of the project. A license to express the ways in which people can use your code, Etc. You can also make your repository private (only available to people with accounts that have permission to see it) or public (searchable and seen by everyone). When you create your repository, you’ll notice that it has a number of tabs, and is opened to the Code tab.

Code – this is where all the source files reside. Git was initially created as a source code repository and now all sorts of files end up in here. If you created a README and/or license, that’s all that’s here right now. Issues – as you can imagine, you can track and plan with tools such as “Issues” that lists all open items against your project base. Pull Requests – this is part of the mechanism for collaborating with other users. Pull requests define changes that are committed and ready for review before being merged into the main branch. Projects – all the tools for managing, sorting, planning, etc. your various projects. This is the core of the collaborative power of GitHub. Wiki, Security, and Insights – often left for more advanced users, these tools provide a communication base to the external user community. Settings – GitHub allows for a lot of personalization, including changing the name of your repository and controlling access. In this video, you learned: How to create and verify a GitHub account. Repositories are storage structures that can hold Code, track Issues, and enable you to collaborate with others.

**GitHub – Getting Started**

Let’s start by creating a new repository. Click + then click New Repository. To create a new repository, you need to provide these details: give your new repository a name; optionally, add a description of your repository; choose the repository visibility - whether you want it to be public or private; and choose the option to Initialize this repository with readme file. Then click Create Repository.

You will now be redirected to the repository you have created. The root folder of your repository is listed by default and it has just one file ReadMe.md.

Now, it’s time to edit the readme. You can do this in your browser. Just click the pencil to open the online editor and you can change the text of the readme. To save your changes to the repository, you must commit them. After you have made your changes, scroll down to the Commit changes section. Add a commit message and optionally add a description, then click Commit changes. The "commit changes" is used to save your changes to the repository. Go back to the home screen by clicking the repository name link. Note that the readme file is updated and verify your changes.

Let’s learn how to create a new file using the built-in web editor provided by GitHub which runs in the browser. Click Add File, the click Create New File to create the new file.

To create a python file called firstpython.py. First, provide the file name. Next, add a comment that describes your code, then add the code.

Once finished, commit the change to the repository. You can see that your file is now added to the repository and the repository listing shows when the file was added or changed. When you need to change the file, you can edit it again. Click the file name, and then click the pencil icon, make your edits and commit the changes.

You can also upload a file from your local system into the repository. From the home screen of the repository, click Add File and choose the Upload files option.

Click Choose Your Files and select the files you want to upload from your local system.

The file upload process may take a short time, depending on what you are uploading. Once the files finish uploading, click Commit Changes. The repository now reflects the files that were uploaded. In this video, you learned how to create a repository, edit files, and commit changes using the web interface.

**GitHub: Working with Branches**

A branch is a snapshot of your repository to which you can make changes. It is a copy of the master branch and can be used to develop and test workflow changes before merging it into the master branch. In Git and GitHub, there is a main branch called master. It has the deployable code and is the official working version of your project. It is meant to be stable, thus, it is advisable not to push any code that has not been tested in the master. If you want to change the code and the workflow in the master branch, you can create a copy of the master branch. This can be the child branch that will be a copy of the workflow. In the child branch, changes and experiments are done. You can build, make edits, test the changes, and when you are satisfied with them, you can merge them back to the master branch, where you can prepare the model for deployment. You can see that all of this is done outside the main branch and until you merge, changes will not be made to the workflow before you branched. To ensure that changes done by one member, do not impede or affect the workflow of other members, multiple branches can be created and merged appropriately with the master after the workflow is properly tested and approved. To create branches in GitHub, let’s look at this repository. There is currently one branch in the repository. You want to make some changes but don’t want to alter the master in case something goes wrong, so you will create a branch. To do that, you will click the drop-down arrow and create a new branch. Name the new branch ‘child branch’ and then click enter. The repository now has two branches, the master and child branches. You can check this by selecting the child branch in the Branch selector drop-down list. All the content in the master branch is copied to the child branch. However, you can add files in the child branch without adding any to the master branch. To add a file, ensure the child branch is selected in the branch selector drop-down list. Then click Create new file. In the space provided, name the file ‘test child dot py’ and then add a few lines of code. You can print the statement inside the child branch. At the bottom of the screen, you will see a section, ‘Commit new file.’ Commit messages are important as they help to keep track of the changes made. Add a descriptive commit message for the convenience of the team. Here you can add ‘Create test child dot py.’ Then click Commit new file. The file is added to the child branch. You can verify by going to the master branch by clicking ‘master’ from the Branch selector menu, and you can see that the new file is not added to the master branch. After you have created the new file, test and ensure it is working. You can merge the changes in the child branch to reflect in the master branch by creating a Pull Request (PR). Pull requests show the differences in the content from both branches. It can notify other team members of the changes and edits to the main branch. Ideally, another team member reviews the changes and approves them to be merged with the Master branch. Pull requests are a means of collaboration on GitHub. When you open a pull request, you propose your changes. You can assign team members to review and approve your contribution and merge in the target branch. To open a pull request to see the differences between the branches, click Compare and pull request. If you scroll down to the bottom of the screen, you will see the comparison between both branches. It shows that one file has changed, and the file has two additions, the two lines you added to the file with zero deletions. You will now create the pull request. Add the title and an optional comment. Click Create pull request. The next screen will show the details of the pull request. If you are okay with the changes, click Merge pull request and then click Confirm. You will get a confirmation that the pull request has been successfully merged. You can delete the branch if you no longer need to edit or add new information. Now, the child branch has completely merged with the Master branch. You can check the Master branch and verify it contains the test child dot py file. In this video, you learned: A branch is a snapshot of your repository to which you can make changes. In the child branch, you can build, make edits, and test the changes, and then you can merge them with the Master branch. To ensure that changes done by one member do not impede or affect the workflow of other members, multiple branches can be created and merged with the master. And, a pull request is a way to notify other team members of the changes and edits to the main branch.

**GitHub Branches**

All files in GitHub are stored on a branch. The master branch is definitive. It stores the deployable version of your code. The master branch is created by default, however, you can use any branch as the main, finished, deployable version of the code. When you plan to change things, you create a new branch and give it a descriptive name. The new branch starts as an exact copy of the original branch. As you make changes, the branch that you created holds the changed code. To create a new branch, click drop-down branch: master Add new branch name into new branch text and select Create branch. GitHub branches can be very complex for large software projects. For a simple project, such as the ones we are exploring, consider the following: Start with a common base, the initial source for this project. At one point, the code is branched while new features are developed. In this example, both branches are undergoing changes. When the two streams of work are ready to merge, each branch’s code is identified as a tip. and the two tips are merged into a third, combined branch. Developers work on source files in a branch. Since some projects take a while, the source doesn’t make sense right away. To change the contents of a file: Select the file. Click the pencil icon. Make the changes. Commit the changes. When the developer has completed their assigned work, to save their changes, they commit the code. Commit indicates that the developer is convinced that the code represents a stable platform for the feature or set of features being developed. When a developer commits changed source to their path, they are required to write a comment that describes the changes. The comment should be meaningful and descriptive. The developer can choose to commit to the current branch or create a new branch. Some best practices : Don’t end the commit message with a period. Keep commit messages under 50 characters – use the extended window for the details. Always write in an “active” voice. Pull is used to initiate the merging of branches in a way to capture changes. A pull request makes the proposed (committed) changes available for others to review and use. A pull can follow any commits, even if code is unfinished. A pull requires a user to approve the changes. This can be the author of the change or it can be assigned within the team. Note that GitHub automatically makes a pull request on your behalf if you make a change on a branch that you do not own. Since the log files are immutable, it is always possible to find the person who approved the merge of the change. To open a new pull request: Click Pull request and select New pull request. Select the new branch from the compare box. Scroll down to view the changes. Confirm that the changes are what you want to assess. Add a title and description to the request. Click Create pull request. The intent of Git repositories is for the master branch to be the only deployed code. Developers can change source files in a branch but the changes are not released until. They are committed. A pull command is issued. The code is reviewed and approved. The approved code is merged back into the master code. To merge a committed code change into your master code: Click Merge pull request. Click Confirm merge. When all changes for a branch are complete, that branch is considered obsolete and it should be deleted. In this video, you learned: All files in GitHub exist on a branch. The Master Branch contains the finished, deployable version of the code. Create new branches when you need to change the code. The new branch starts as an exact copy of the original branch. As you make changes, the branch that you created holds the changed code. More than one branch can be undergoing changes at the same time. Saved changes are called commits. Pull requests enables other users to review and use the proposed changes (committed). When you are ready to merge the changed code into the master branch, you merge the committed code changes into your master code.