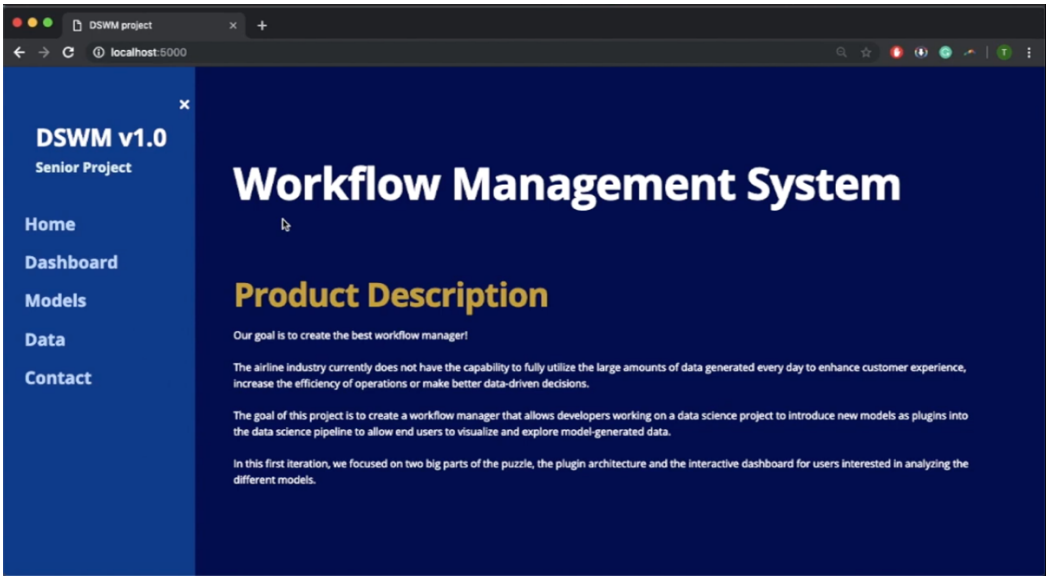
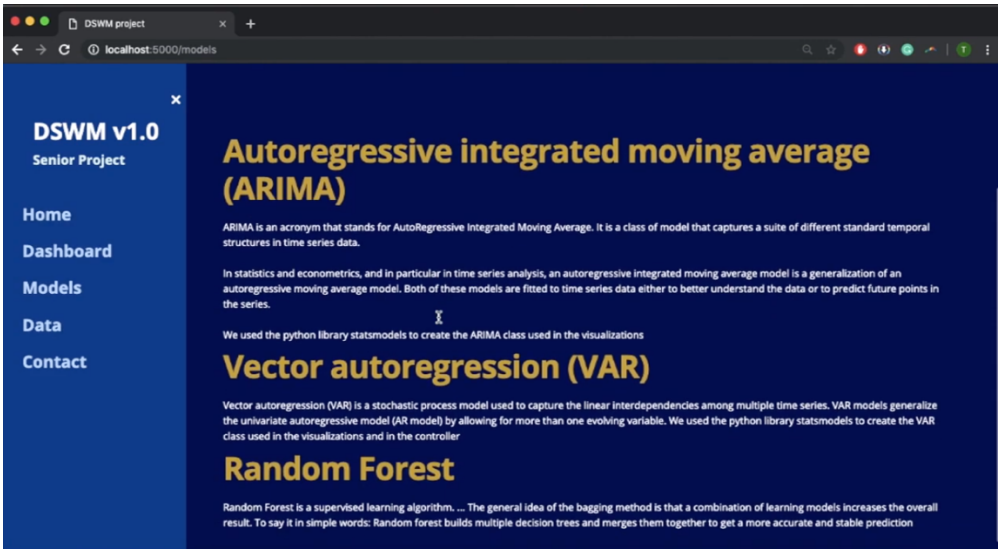
Appendix D - User Manuals, Installation/Maintenance Document, Shortcomings/Wishlist Document and other documents

User Manual for End Users

When the page loads, the user is going to be able to use the side navigation bar to move through the website.



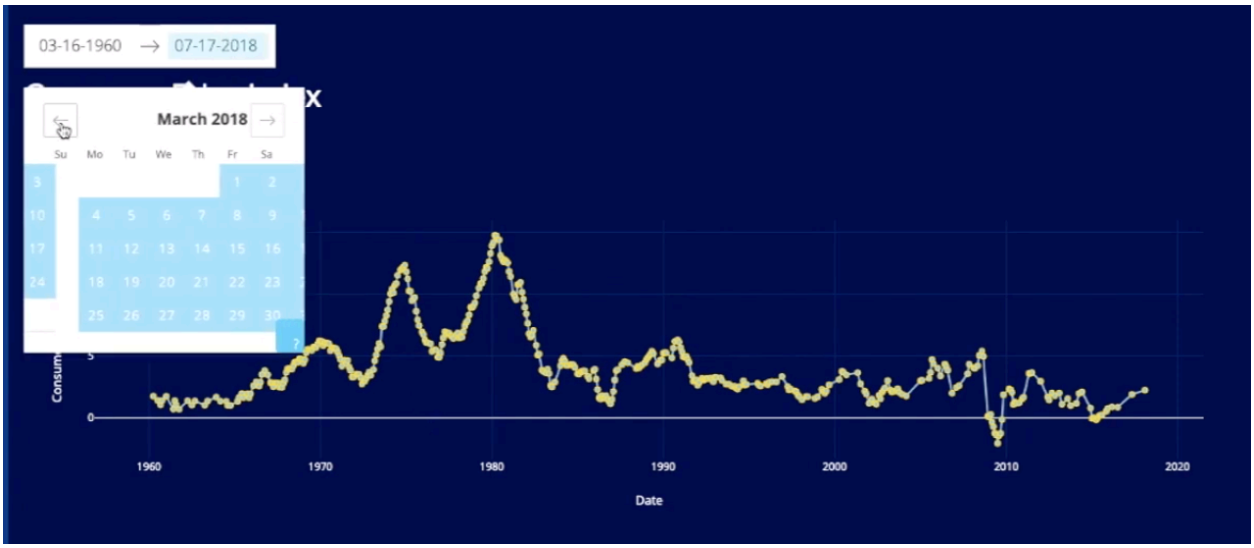
In the main page, the user can read about the goals of the project and the new updates that will be coming soon. In the navigation bar we can see that there are several links that will direct the user to information about the different models and about the data sources that we used for training the model and for the rest of the visualizations.



The user can go to the dashboard and click on the points that are part of the training data to move them around using the slide as shown in the picture below.

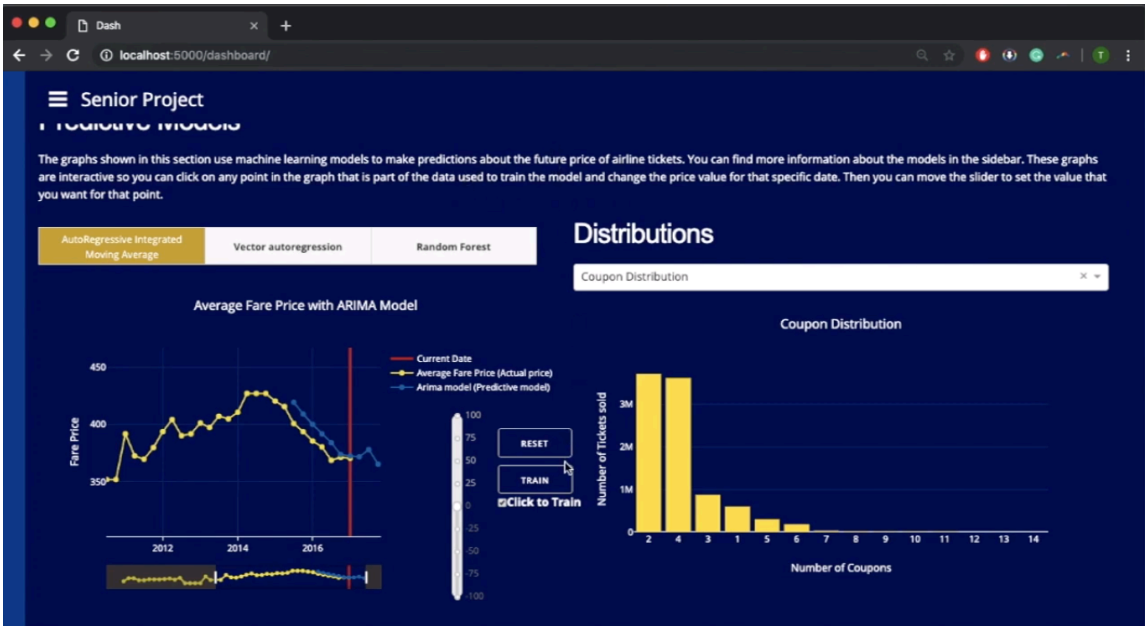
After the user has achieved a configuration that he believes is the one he was looking for, the user can click on the train button to train the machine learning model. If the train button is disabled, then every time the user moves a point up or down, the model will be train with the changes he made.

The user can also interact with the other graphs in the dashboard by picking a new range for the visualizations. Using the range selector, the user can go through the months and find exactly the moment he is looking for. Then the graph will update to show only the points that the user wants to see.



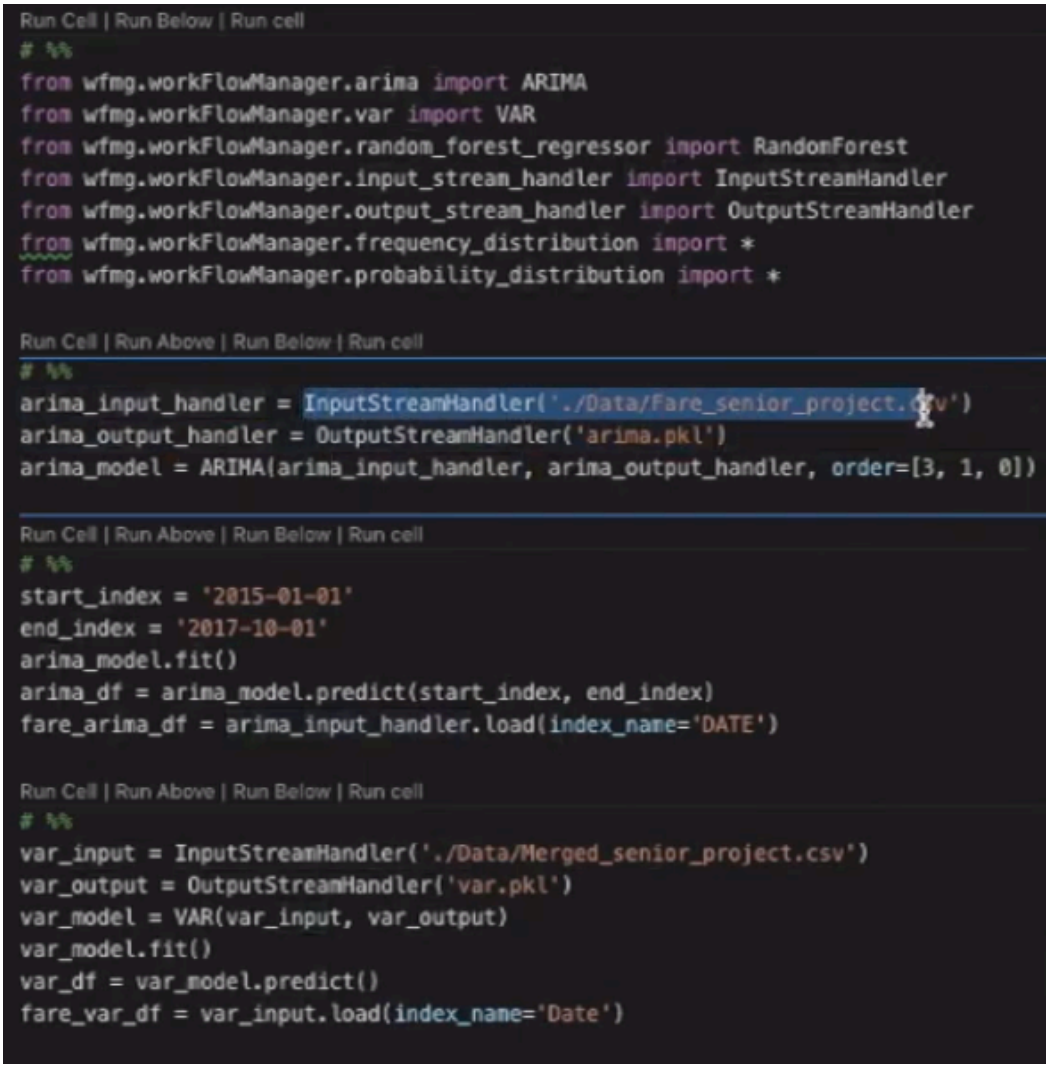
The user can also change the amount of columns that will be visible in the dashboard. The default value for this component is 1, but the user can choose to view up to 5 graphs in a single row. In the picture below we can see the component for changing the number of columns.

If the user decides that he prefers to view the dashboard 2 graphs at a time, the application will look like this.



User Manual for Data Science Developers

The workflow manager application allows developers to implement new models using the classes provided by the wfmg module to quickly and painlessly visualize the results in the web application.



```
Run Cell | Run Below | Run cell
# %%
from wfmg.workFlowManager.arima import ARIMA
from wfmg.workFlowManager.var import VAR
from wfmg.workFlowManager.random_forest_regressor import RandomForest
from wfmg.workFlowManager.input_stream_handler import InputStreamHandler
from wfmg.workFlowManager.output_stream_handler import OutputStreamHandler
from wfmg.workFlowManager.frequency_distribution import *
from wfmg.workFlowManager.probability_distribution import *

Run Cell | Run Above | Run Below | Run cell
# %%
arima_input_handler = InputStreamHandler('./Data/Fare_senior_project.csv')
arima_output_handler = OutputStreamHandler('arima.pkl')
arima_model = ARIMA(arima_input_handler, arima_output_handler, order=[3, 1, 0])

Run Cell | Run Above | Run Below | Run cell
# %%
start_index = '2015-01-01'
end_index = '2017-10-01'
arima_model.fit()
arima_df = arima_model.predict(start_index, end_index)
fare_arima_df = arima_input_handler.load(index_name='DATE')

Run Cell | Run Above | Run Below | Run cell
# %%
var_input = InputStreamHandler('./Data/Merged_senior_project.csv')
var_output = OutputStreamHandler('var.pkl')
var_model = VAR(var_input, var_output)
var_model.fit()
var_df = var_model.predict()
fare_var_df = var_input.load(index_name='Date')
```

In this picture we can see that we must import the libraries for the code to work. The user then must initialize objects for the input stream handler and output stream handler for each model object that he wants to create so that the model can get access to the training data. Notice that we need to create a range for the dates that the model will predict. If we run this code, we are going to get a data frame with the predictions from the ARIMA model from the start date that we defined to the end date.

In the picture we also see that the VAR model is also available to developers. The VAR model is interesting because it uses more than one Time Series to predict the future prices.
The regular result for a univariate model such as ARIMA is shown below.

```
FARE
2015-07-01  419.367185
2015-10-01  409.078142
2016-01-01  400.031327
2016-04-01  391.730816
2016-07-01  384.049721
```

The VAR model on the other hand is capable of using several data frames to make it's predictions like we can see in this picture.

```
0           1          2
2015-07-01  407.522875  62.089991  8.168354
2015-10-01  403.175606  67.270818  8.482643
2016-01-01  401.536517  71.874392  8.759450
2016-04-01  401.765288  75.756540  9.009956
2016-07-01  403.206604  78.894046  9.243163
            AVG_FARE         oil       cpi
2006-01-01    339.01   63.266667  1.493333
2006-04-01    364.66   70.410000  1.250000
2006-07-01    364.66   70.416667  1.246667
2006-10-01    347.62   59.976667  1.006667
2007-01-01    347.80   58.076667  1.106667
```

The developers also have access to several scripts that take care of all the preprocessing of the csv files that we used such as changing the index to a dateindex, changing the name of the columns etc. Using the wfgm module, developers can also extend the functionality of our models by creating classes that inherit from the WFMG_model class or the WFMG_proc class.

In the project repository there are 3 exameples, the ARIMA model, the VAR model and the Random Forest Model. Here we decided to show part of the code of the ARIMA model so that you can get an Idea of how to create subclasses that can be used by the visualization module.

```
class ARIMA(WFMG_Model):
    def __init__(self, input_stream, output_stream, **kwargs):
        super().__init__('ARIMA', input_stream, output_stream, **kwargs)
        self.input_data = self.input_stream.load(index_name='DATE')

    def predict(self, start_index, end_index):
        """This function trains the ARIMA model to predict the fare
        Keyword arguments:
            start_date(string): start_date string with the format of yy-mm-dd.

            end_date(string): end_date string with the format of yy-mm-dd.
        """
        if self.ml_model is not None:
```

Installation and Maintenance.

First, you must clone the repo from the bitbucket repository or from the AirLab github repo. Once the repository is stored locally, you can start to download the dependencies. All dependencies are stored in the the requrements.txt file. To install all of them at the same time you must use pip. I suggest that you use pip3 just in case you still have the old version of pip laying around somewhere. For the development of this project we used python version 3.6.

Before you install the requirements, I would suggest that you create a virtual environment so that any new dependency that gets added to the project can be stored in the file requiremets.txt.
It is also good practice to keep the python interpreter that you are using constant throughout the development process.

You can create a virtual environment by typing this command.

python3 –m venv venv

The second venv is just the name of the folder that the utility venv is going to create in our current directory. You can change it to anything you want.

After the folder is created, you can activate the virtual environment by typing this command.

source venv/bin/activate

Notice that if you have a folder with a different name, then you should replace the venv in that last command.

Now, you are ready to install the dependencies and start coding. You can install the requirements by typing the following command in any bash terminal.

pip3 install -r requirements.txt

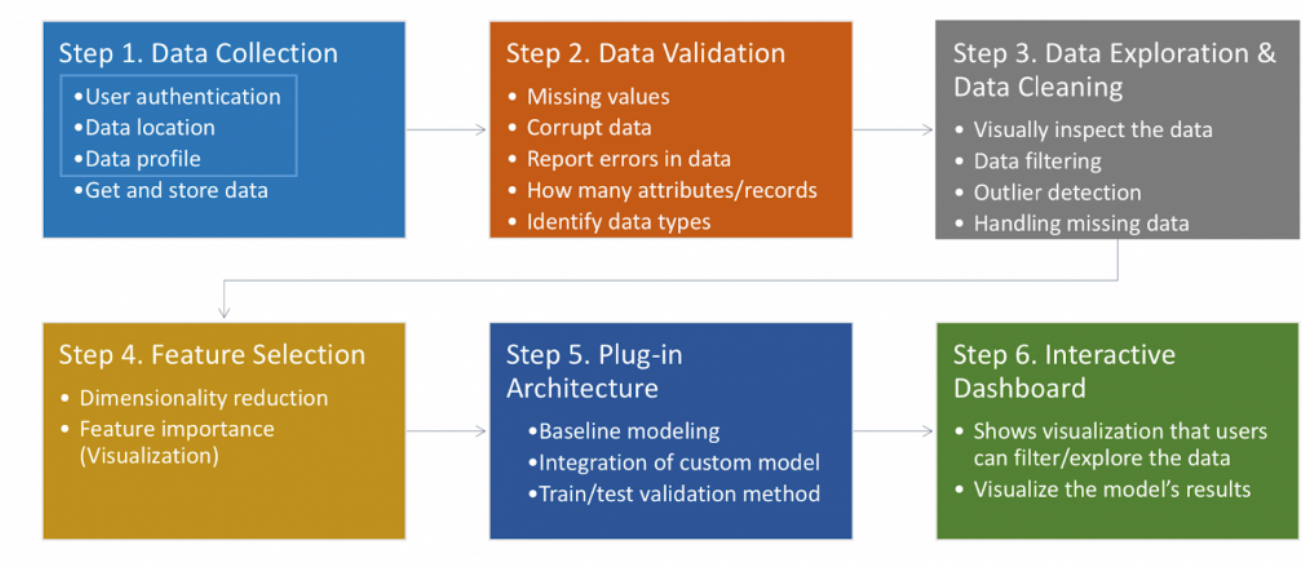
Now you are ready to start coding!

For more information on how to maintain the code and how to install everything that you will need, please refer to the video “FIU SCIS 2019Spring WFMG InstallMaintenanceGuide”

Shortcomings and Wishlist

The main shortcoming in this first iteration was that we did not get to create a Database to store the data and models, in order to process the data even more efficiently and also to process new data automatically.

The main goal of the project was to implement the last few steps of the data science pipeline as shown below and on the FIU AIRlab website. We mainly focused on the plug-in architecture and the
interactive dashboard. Future iterations should implement the remaining steps of the pipeline in reverse order ending with Data collection.



REFERENCES

- The Plotly Reference Manual: https://plot.ly/python/reference/
- The Dash User Guide with tutorials and many examples: https://dash.plot.ly/
- Flask documentation: http://flask.pocoo.org/docs/1.0/