# Port Authority Automatic Report Maker (PAARM) by Nicholas Lombardo

The purpose of this project is to assist the AVI Forecasting team in their efforts to predict airport passenger volumes at the Port Authority through customizable report generation. NOTE THESE INSTRUCTIONS ARE FOR PORT AUTHORITY EMPLOYEES ONLY. IF YOU ARE NOT A PORT AUTHORITY EMPLOYEE FOLLOWING THESE INSTRUCTIONS MAY RESULT IN AN ERROR.

**IMPORTANT:** THIS IS ONLY THE PUBLIC VERSION OF THE FULL REPOSITORY AND THEREFORE SOME FILES MAY BE MISSING INCLUDING, "Monthly Pax.csv", "Monthly Seats.csv", "Monthly Economic.csv", "Yearly Pax.csv", "Yearly Economic.csv", "Short Term Data.xslx", "Long Term Data.xslx", and the /Images folder.

## Capabilities

Each report generated creates an ensemble (collection) of several SARIMAX and ARIMA time series models and averages out the results. Seats data is also incorporated to all these models as an exogenous variable. In terms of what kinds of reports could be generated by this project there are two main options: backtest and extrapolate. If you select backtest, the models will be trained on historical data up until the month you select for time and will evaluate the predicted accuracy of the ensemble using mean average percentage error from the month you select until the present. If you select extrapolate the model will be trained on all historical data and will predict until the month you select for time using future predicted seat data. **You must at least one month and you must pick either extrapolate/backtest or the report will not run.** There is a button to "Add Multiple" times which will generate one report for each time value you select on the calendar. If you hit "Add Multiple" by mistake please hit the "Delete" button to get rid of times you do not want, DO NOT just leave the other time values blank as this will cause an error. Selecting too many time values may cause your report to take several minutes to run.

You can decide what you want included in your report through the PAARM Report Settings Webpage.

* Every report will generate an excel file containing all predicted passenger volumes.
* You get to decide whether predictions should be made in the long term (which predicts on a yearly basis but does not use seats as an exogenous variable) or the short term (which predicts on a monthly basis and uses seats as an exogenous variable.)
* You have the option to include optimistic and pessimistic predicted passenger volumes in additions to the average predicted by the ensemble.
* You have the option to export the model parameters to see what is included and the trained coefficients of every model. Selecting this may cause your report to take several minutes to run.
* You have the option to export a set of graphs depicting predicted passenger volumes as a png. (Selecting this will generate one graph for every airport, domestic or international, and optionally airline.)
* You have the option to divide by airline in addition to airport and international/domestic for increased granularity. Selecting this may cause your report to take several minutes to run.
* You have the option to include GDP as an exogenous variable of the models in addition to seat data. Selecting this will cause Percent Change MoM in Real US GDP to be an exogenous variable in all short term models or Percent Change YoY in Real US GDP + Percent Change in Fuel Prices to be exogenous variables in all long term models.
* You have the option to include a Covid dummy variable as an exogenous feature of the model. This dummy variable is 1 within the user defined range of covid and 0 elsewhere.
* You have the option to select what models should be included in the ensemble. (3-5 models are recommended.) **At least one model must be added to the Ensemble.** For short term forecasting SARIMAX models are used while for long term forecasting ARIMAX models are used. To learn about the difference between these SARIMAX/ARIMAX models click here: <https://www.geeksforgeeks.org/complete-guide-to-sarimax-in-python/>
* Summarized: Every SARIMAX model has order = (p, d, q) and seasonal order = (P, D, Q, T). While SARIMAX, has an order and a seasonal order ARIMA only has an order (not a seasonal order) and therefore does not take seasonality into account. SARIMAX also incorporates exogenous variables while ARIMA does not.
  + Here's a simple explanation for each component:
  + Order (p, d, q):
    - p: How many past values are used to predict the future. (# of autoregressive components)
    - d: How many times the data is subtracted to make it stationary. (# of times data is differenced)
    - q: How many past forecast errors are used in the prediction. (# of moving average components)
  + Seasonal Order (P, D, Q, T):
    - P: How many past values from the same season are used. (# of seasonal autoregressive components)
    - D: How many times the seasonal data is adjusted. (# of times seasonal data is differenced)
    - Q: How many past seasonal forecast errors are used. (# of seasonal moving average components)
    - T: How long the seasonal cycle lasts (e.g., 12 for yearly seasonality).
* You also have the option to predict using deep learning rather than (S)ARIMAX Modeling by selecting "Deep Learning" over the default "(S)ARIMAX" tab on the form. **NOTE:** Deep Learning is only available for short term modeling and trying to predict using deep learning in the long term will result in an error.
  + If you select deep learning you have three tunable parameters to customize in the neural network: layers, nodes, and epochs
    - Layers: Determines the number of layers in the neural network (must be between 1 and 10.) A high number of layers can result in a long runtime.
    - Neurons: Determines the number of neurons in each layer of the neural network (must be between 8 and 64.) A high number of neurons per layer can result in a long runtime.
    - Epochs: Determines the number of training cycles (epochs) each model should be trained for (must be between 16 and 2048.) A large number of training epochs per layer can result in a long runtime.
* You also may give every model you add into the ensemble a custom weight by changing the number next to the model once selected on the website. Note that weights must be integers greater than or equal to one.
* You **must** give your report folder a custom name typing it into the Report Name field in the webpage when it is saved into your /Reports directory.
* Lastly, you have the option to change which airports and dom/int types should be included including Subtotals. The default option here is All, to change this please unselect all and select the options you want to be included in your report. NOTE: Subtotal represents the summation over all possible options not only the options you select. (For instance if you select EWR, JFK, and Subtotal for airports Subtotal is still the sum over all four airports not just EWR and JFK.)

## Instructions

### Setup

In order to properly generate reports using this file you will need to have two applications installed **Anaconda** Navigator and **Visual Studio Code**. Both of these applications are approved for non standard use by the Port Authority and can be requested HERE. After you get approval download Anaconda here: <https://www.anaconda.com/> and Visual Studio Code here: <https://code.visualstudio.com/download>.

### Downloading the data

#### Short Term: As of August 2024 there are two updated .csv files in the /datasets folder. However, for future use download the data by following the steps below:

* **For short term modeling you need two csv datasets called "Monthly Seats.csv" and "Monthly Pax.csv" in your /datasets folder.**
* First go to the **Short Term Data** Excel file in the PAARM folder.
* Then, go to the Seats Act page of the excel file.
* Find the file tab in the upper lefthand corner.
* Hit export + change file where you should select the csv option. (Note this may cause an warning about exporting multiple sheets - just ignore this) and save this csv as Monthly Seats.csv into your /datasets folder.
* Repeat this for the passengers now by going to Pax Act page of the excel file and save this as Monthly Pax.csv in the /datasets folder.

#### Long Term:

* **For short term modeling you need only ONE csv datasets called "Yearly Pax.csv" in your /datasets folder.**
* First go to the **Long Term Data** Excel file in the PAARM folder.
* Then, go to the Pax Act page of the excel file.
* Find the file tab in the upper lefthand corner.
* Hit export + change file where you should select the csv option. (Note this may cause an warning about exporting multiple sheets - just ignore this) and save this csv as Yearly Pax.csv into your /datasets folder.
* No need to repeat this for seats as Long term modeling does not use seat data.

### Running a report

To run a report first open Anaconda and navigate to where it says VS Code and hit Launch. (Do not try to Launce VS Code without Anaconda this will cause an error.) Now on VS Code, click **Open Folder** and open the PAARM Folder which could be found in the Port Authority Sharedrive > AVI Forecasting Team - Documents > Forecasting > Long Term Pax > PAARM. Once you open the PAARM Folder, click on PAARM.py and **press the triangular run python file** in the upper righthand corner. An alternative way to run this file is by typing: python PAARM.py into the terminal. **Please make sure you are opening the PAARM Folder using VS Code and not the PAARM.py file.**

**Warning: Make sure all the files in the Sharedrive have a green checkmark (indicating they are downloaded locally) rather than a blue cloud (indicating they are on the cloud) otherwise the project will not be able to access these files.**

Now after a few seconds a website titled **PAARM Report Settings** should pop up which will contain all the ways you can customize your report. Everytime you want to create a new report you must run the Python file again, reloading the page and submitting the form again may cause a "Failed to fetch error." If you want to start over go to VS Code and press **control+C** while in the terminal, this will terminate the server and allow you to run your Python file again. If you accidently close the tab, run your Python file again.

**NOTE ON RUNNING THIS FILE THE FIRST TIME:** The first time you run this file two packages (Tensorflow and arch) will automatically be installed locally onto your device. These packages may take several minutes to run **do not keep hitting the play button as this will cause your progress to start over**. After a few minutes these packages will already be installed and you will not have to repeat this process. From here it should only take a few seconds before hitting the "play" button in the upper righthand corner and the webpage opening up.

Once you are done filling out the report generator hit submit and then screen should change to say "Report Running..." once you are on this page wait patiently as some reports take upwards of 5-10 minutes to run. If however, the screen changes instantly that means there was an error running your report. The error given may have been related to the particular inputs you filled out on the form, read the error message printed to the screen and see if you could fix your inputs so the report could be properly generated. If the error remains and you are unsure why please contact Nicholas Lombardo: [nlombardo@panynj.gov](mailto:nlombardo@panynj.gov).

Once your report is done running, the report should be saved to the /Reports folder under the name you entered in the form.

### Making sure your project is up to date (OPTIONAL)

As this project is still in development and being updated frequently, if you would like to ensure you have the latest version of the project there are two ways you can get the most recent version.

**Download the project locally:** If you would like to download the file locally simply open any folder you would want to save your project INTO using VS Code. Now in the upper righthand corner select View > Terminal and a new command line interface should appear. On this command line copy: git clone https://github.com/nicholasl25/PAARM-Public.git After a few seconds you should see the PAARM folder saved into your directory. From here you now have the latest version simply open this PAARM **folder** using VS Code now and follow the instructions above.

**Download a new version of the Project into the Port Authority Sharedrive:** If the project in the Port Authority Sharedrive is outdated or messed up, to download a fresh version of the project, open VS Code and click **Open Folder**. Now navigate to Port Authority Sharedrive > AVI Forecasting Team - Documents > Forecasting > Long Term Pax and open the **Long Term Pax** folder. From here select View > Terminal in VS Code and a new command line interface should appear. On this command line copy: git clone https://github.com/nicholasl25/PAARM-Public.git After a few seconds you should see a new version of the PAARM Folder saved into Long Term Pax. From here you now have the latest version simply open this PAARM **folder** using VS Code now and follow the instructions above.

## Methodology + File Structure

Here is a brief description of what every file and directory in this PAARM Folder does:

* **PAARM.py**: The main python file which is ran by the user. This file, when ran, automatically opens the Report Builder website and calls the appropriate classes and functions from the Classes folder to generate your report. This file interprets the result of the Report Builder website you submit and uses them to create a report.
* **Classes** : Folder containing all custom classes and helper functions
  + **Helper.py**: This file contains all the helper functions needed to actually create the report. In this file you will find minor functions such as get\_title which returns a string of the appropriate graph title given a key and more important functions such as **get\_dict** which divides the data by type/airport/airline and returns the result in the form of a dictionary, and most importantly **get\_everything** which generates the entire report itself. The Pax/Seats/GDP data is also preprocessed here and put into pandas dataframes all\_pax, all\_seats, and GDP respectively.
  + **Predictor.py**: This file contains the template superclass Predictor which is inherited by the Past and Future subclasses. This file defines the recquired methods of any predictor object: .predict(), .save\_numbers(), .save\_numbers2(), .save\_graphs(), .export\_model\_params(), and .get\_date(). The implemetation of these functions is empty blank as no object of class Predictor should be created, rather create classes that inherit from this class such as Past and Future which have their own unique implementations of all these methods.
  + **Past.py** : Class that inherits from Predictor, this class is created by the get\_everything function when "backtest" is selected in Settings.html. This class contains the implementations of all required Predictor methods. Like the name suggests, this class is used to generate a report that is predicting into the past and backtesting the results.
  + **LongPast.py** : This file contains the LongPast class which is a subtype of Past and inherits many of the same methods from it. This class inherits its .save\_numbers(), .save\_numbers2(), .save\_graphs(), and .export\_model\_params() methods from Past but has its own uniquely defined .predict() and .get\_date() methods. LongPast class is used for backtesting real vs predicted passenger volumes in the long term rather than the short. Therefore, it makes predictions on a yearly basis rather than monthly and does not use seat data as an exogenous variable. Additionally class variable monthly is set to False in this class.
  + **DeepLearnerPast.py** : This file contains the DeepLearnerPast class which is a subtype of Past and inherits many of the same methods from it. This class inherits its .save\_numbers(), .save\_numbers2(), .save\_graphs(), .get\_date(), and .export\_model\_params() methods from Past but has its own uniquely defined .predict() method. The DeepLearnerPast class is used for short term passenger forecasting but rather than using SARIMAX ensemble modeling it creates and trains a single neural network. DeepLearnerPast is used when the user selects "Deep Learning" in the Settings.html form and generating a report using deep learning may result in errors/large runtimes.
  + **Future.py** : Class that inherits from Predictor, this class is created by the get\_everything function when "extrapolate" is selected in Settings.html. This class contains the implementations of all required Predictor methods. Like the name suggests, this class is used to generate a report that is predicting into the future (extrapolating.)
  + **LongFuture.py** : This file contains the LongFuture class which is a subtype of Future and inherits many of the same methods from it. This class inherits its .save\_numbers(), .save\_numbers2(), .save\_graphs(), and .export\_model\_params() methods from Future but has its own uniquely defined .predict() and .get\_date() methods. LongFuture class is used for extrapolating passenger volumes in the long term rather than the short. Therefore, it makes predictions on a yearly basis rather than monthly and does not use seat data as an exogenous variable. Additionally class variable monthly is set to False in this class.
  + **DeepLearnerFuture.py** : This file contains the DeepLearnerFuture class which is a subtype of Future and inherits many of the same methods from it. This class inherits its .save\_numbers(), .save\_numbers2(), .save\_graphs(), .get\_date(), and .export\_model\_params() methods from Future but has its own uniquely defined .predict() method. The DeepLearnerFuture class is used for short term passenger forecasting but rather than using SARIMAX ensemble modeling it creates and trains a single neural network. DeepLearnerFuture is used when the user selects "Deep Learning" in the Settings.html form and generating a report using deep learning may result in errors/large runtimes.
  + **Filter.py** : Simple class that does not inherit from Predictor and rather is jused used as an object that defines what keys (types/airports/airlines) should be included in the report and which should not. This class only has one method: .filter\_keys().
  + **Covid.py** : Simple class that represents the user inputted custom Covid dummy variable. Covid objects have two methods .get\_include() which returns a boolean that is True if and only if a covid dummy variable should be included as an exogenous variable of the model and .get\_function() which returns a lambda function that is 1 if the time value passed into it is between the start and end covid dates and 0 otherwise.
* **Templates**: Folder that stores Settings.html.
  + **Settings.html**: This file is the Report Builder website that automatically pops up when PAARM.py is ran. This website allows people to customize the report that is saved into their /Reports folder. Once submit is pressed the data is sent to PAARM.py and the report is generated.
* **static** : Folder that contains styles and javascript for the Settings.html website.
  + **Styles.css** : Cascading style sheet file that formats the html code found in Settings.html and makes the website appear orderly.
  + **Script.js** : Javascript file containing the backend logic of how the Settings.html website should behave when each button is clicked.
  + **Instructions.docx** : Instructions for working with PAARM (identical to README but in the form of a Word document.)
* **Short Term Data**: Excel Pivot table datasets which "Monthly Pax.csv" and "Monthly Seats.csv" can be downloaded from.
* **Long Term Data**: Excel Pivot table datasets which "Yearly Pax.csv" can be downloaded from.
* **Datasets**
  + **Monthly Pax.csv**: Dataset containing all monthly passenger data. Instructions on how to properly download this data are found in the "Downloading the Data" portion of the Readme. The data in this csv should begin no later than Jan 2016.
  + **Monthly Seats.csv**: Dataset containing all seat data. Instructions on how to properly download this data are found in the "Downloading the Data" portion of the Readme. The data in this csv should begin no later than Jan 2016.
  + **Monthly Economic.csv**: Dataset that contains all monthly economic data (historic and predicted values) to be used as optional exogenous variables on a monthly timeframe for short term modeling. Currently only % Change Real GDP is used as en exogenous economic variable in short term modeling. The data in this csv should begin no later than Jan 2016.
  + **Yearly Pax.csv** : Dataset containing all yearly passenger data to be used for Long Term modeling. This data is used in the long term passenger forecasts when seats are not available. The data in this csv should begin no later than 2002.
  + **Yearly Economic.csv** : Dataset containing all economic data (historic + predicted) to be included as exogenous variables in the long term passenger forecasting. Currently two economic factors are incorperated as exogenous variables: % Change Real GDP and % Change Fuel price. The data in this csv should begin no later than 2002.
* **Old Code**
  + **Old Code.py**: Old Python File that contains outdated helper functions saved just in case they are ever needed again. Some code is a remnant from the Google Colab .ipynb file
  + **ACF.py** : This Python file is simply to create and export graphs of the autocorrelation function of the data (to see what degree AR/MA components are statistically significant and should be included in the model.) This file is **not** meant to be used anymore and running it will likely cause an error.

### How does it work? -- High level overview of code

After you hit "Submit" on the Report Generator website the data is processed to create two objects by PAARM.py. The first is of class Filter and is called My\_Filter in the code, this object is responsible for determining which datasets (dom/int + airports + airlines) should and should not be included in the report. You can see the definition of a filter object if you go to **Classes/Filter.py**. (Optionally: an object of class Covid may be included which will add a covid dummy variable as an exogenous variable to the model.) The next object that is created is an object of super class Predictor titled My\_ensemble in the code which contains all models, weights, and features that should be considered in the report.

**Case #1: Short Term (S)ARIMAX:**

* If you are backtesting in the short term the object will be of type Past which inherits from Predictor class and if you are extrapolating in the short term the object will be of type Future which inherits from Predictor. You can see the definition of a Past object if you go to **Classes/Past.py** and you can see the definitions of a Future object in **Classes/Future.py**.

**Case #2: Long Term (S)ARIMAX:**

* If however, long term is selected rather than short term the Predictor object will either be of type LongPast if backtesting or LongFuture if extrapolating. The definitions of these Predictor subclasses can be found in **Classes/LongPast.py** and **Classes/LongFuture.py** respectively.

**Case #3: Short Term Deep Learning:**

* If however, "Deep Learning" is selected over the default "(S)ARIMAX" tab in the model section of the form then if you are backtesting an object of type DeepLearnerPast will be created while if you are extrapolating an object of type DeepLearnerFuture will be created. The definitions of these Predictor subclasses can be found in **Classes/DeepLearnerPast.py** and **Classes/DeepLearnerFuture.py** respectively.

**Case #4: Long Term Deep Learning:**

* As of August 2024 long term deep learning is not available.

After these two objects are created the **get\_everything function** (defined in Classes/Helper.py) is called on them in addition to several other user inputted variables.

This get\_everything function is the function that actually generates everything in the report. From here get\_everything either calls **.predict()** method of My\_Ensemble (which may be of type Past of Future but must be of parent class Predictor). **It is here in these functions that call the following other methods:**

* **.save\_numbers():** These functions generate the Data.xslx file in the report and **INCLUDE** optimistic/pessimistic predicted values. Their definitions could be found in Classes/Past.py and Classes/Future.py as well as in Classes/LongPast.py and Classes/LongFuture.py.
* **.save\_numbers2():** These functions generate the Data.xslx file in the report and **does not INCLUDE** optimistic/pessimistic predicted values. Their definitions could be found in Classes/Past.py and Classes/Future.py as well as in Classes/LongPast.py and Classes/LongFuture.py.
* **.save\_graph():** These functions are optionally called and are responsible for creating + exporting the matplotlib graphs in the report. Their definitions could be found in Classes/Past.py and Classes/Future.py as well as in Classes/LongPast.py and Classes/LongFuture.py.
* **.export\_model\_params():** These functions are optionally called and responsible for exporting the model parameters as a text file. Their definitions could be found in Classes/Past.py and Classes/Future.py
* **.get\_date():** These functions contain the logic deciding how "time" variables should be converted into a date. These functions are different between classes and predictions are made on a yearly basis in the long term and on a monthly basis in the short.
* To see the specific implementation of the Predictor methods see the documentation in the code of Classes/Past.py and Classes/Future.py as well as in Classes/LongPast.py, Classes/LongFuture.py, Classes/DeepLearnerPast.py and Classes/DeepLearnerFuture.py.