Data Modeling Automation
System Overview Document

Soo Jung Kim, Manali Patil, Allison Wong
Project Sponsors: Christopher Whittelsey, Michael McLaughlin
(PNC Bank)
Instructor: Olga Karpenko
CS690 Fall 2019

Overview

Building appropriate financial time series models are essential to financial institutions as those models can be used to forecast the firm's risk to changing economic environments. PNC bank, the sponsor of our project, has also been putting a significant amount of work to build the different types of models. While the data set is consistent across nearly every model, the modeling scripts written by different modelers differ drastically as each modeler has his or her own modeling methodologies. Therefore, there hasn't been an easy, simple way for the team to run multiple modeling scripts other than running different modeling scripts each time. Also, the results from each modeling script are stored in various excel files, word documents and png images, which made it harder for the modelers to analyze those results. In short, the legacy process of running models was very time-consuming.

For the master project, USF-PNC team built a web application that effectively automates and simplifies the process of building different models of financial time series. It brute forces through various modeling forms, performs all the data transformation independently, and generates the top N models for consideration subject to user defined constraints. The output of selected models would include the statistical diagnostics of each model and graphical representation of dynamic backtesting and sensitivity testing. The saved outputs would also serve as audit trails for Federal Reserve audits of the financial institution as they will be legitimate evidence of the financial time series modeling procedures. The application significantly reduces the amount of redundancy and provides controls which reduce the likelihood of human error. Furthermore, it may be used to find candidate models more quickly and accurately, which could be very useful for users in benchmarking models against existing candidate models.

Requirements

The project requirements were split into major sections: Data Science related requirements, Database related requirements, Web Application related requirements.

Data Science Related Requirements

Core Functionalities

- Writing data transformation script
- Writing model output to the database
- Saving data points for interactive graphs
- Implementing One Factor Regression model

Extra Functionalities

- Providing options to choose which statistical tests to be run
- Refactoring and parameterizing the model script to make it reusable

Database Related Requirements

Core Functionalities

- Creating database schema based on the sponsors' requests
- Saving all the input values that user puts in when a job is started
- Saving all the output values after each test
- Creating database migration that standardizes the table creation and editing progress

Extra Functionalities

- Creating additional tables to save data points of interactive graphs
- Saving intermediate outputs after the preliminary statistical testing

Web Application Related Requirements

Core Functionalities

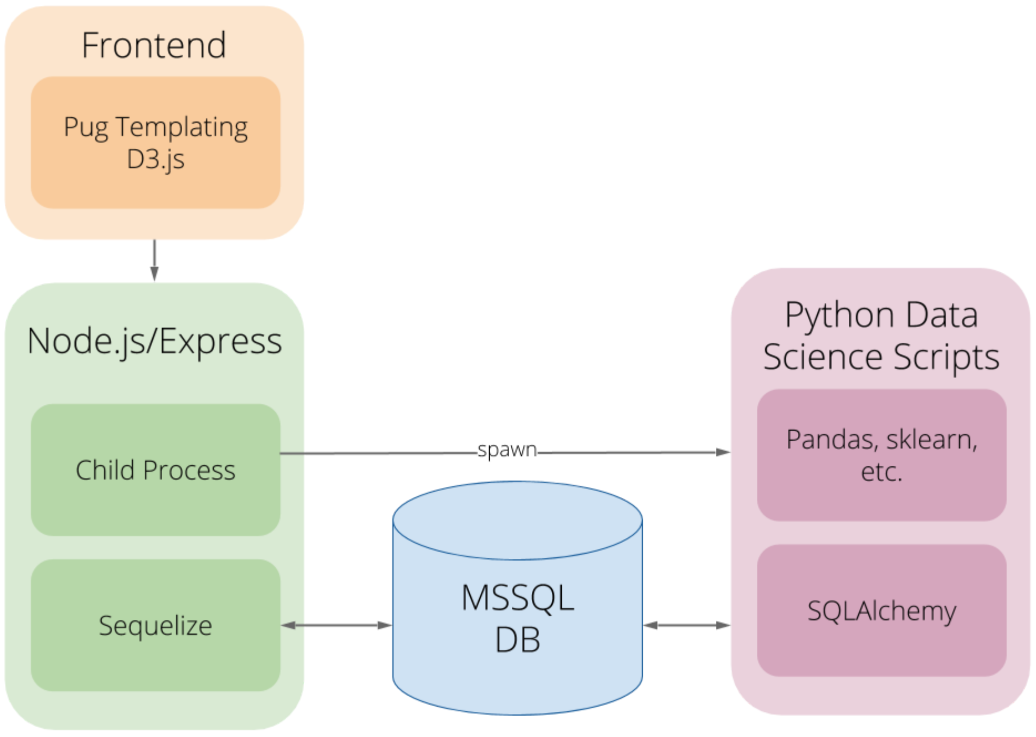
- Backend
    - Connecting the web application with database (MSSQL)
    - Connecting the web application with the data science python script
    - Handle saving and renaming of user uploaded files
- Frontend
    - **GET /** - user can upload the dataset and configure tests
    - **POST /jobs** : handle form submission for new job by spawning the first child process for the preliminary data science script
    - **GET /jobs** - user can view the status of all jobs
    - **GET /jobs/:id** - user can view models that passed/failed the preliminary statistical tests and select them to be included in shortlist
    - **POST /shortlist** - handles form submission for selection of shortlist and spawn the script for additional testing such as backtesting, sensitivity tests, etc.
    - **GET /model/:modelId** - user can view detailed results of each model with graphs

Extra Functionalities

- Spawning the second child process when the user select the models to be included in the shortlist
- Each backtest date input can take more than one date
- Interactive front end to add/remove fields
- Made the graphs interactive with d3.js

Design

Design Overview



The main components of the project interact with each other as shown in the diagram above. Whenever user uploads a data set and starts a job, the backend of the web application (node.js) will spawn a child process to run the python data science scripts with the preliminary statistical test. The intermediate output from the script will be stored to our MSSQL database, using SQLAlchemy ORM. Then, the web application will grab information from the database, using Sequelize ORM, to display relevant results such as statistical values and interactive graphs on the frontend to user.

Software Blocks

Database

In order for the python script to interface with the database, we utilized SQLAlchemy, we wrote a python class to access the database and handle the connection generated by SQLAlchemy.
The **idb.py** is the script that interacts with the database using SQLAlchemy

Code snippet from idb.py

If a new table is created in the database the following should be included in idb.py for inserting or updating any fields in the table

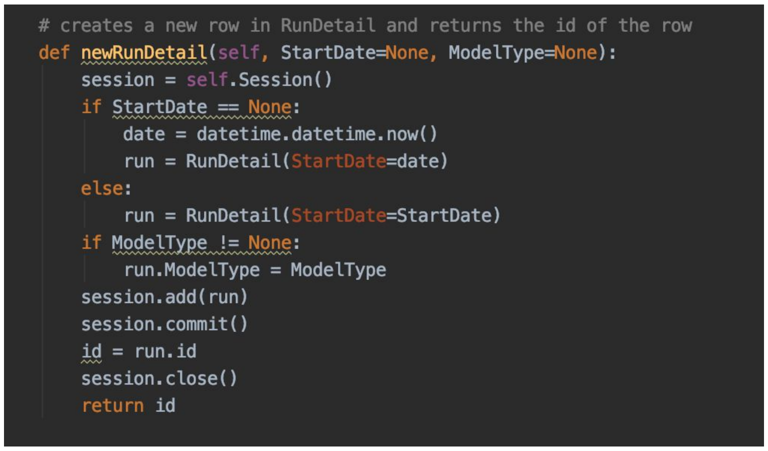
Example of the PACFPlots

```
class PACFPlots(Base):
    __tablename__ = 'PACFPlots'
    id = Column(Integer, primary_key=True)
    ModelId = Column(ForeignKey('ModelRunDetail.id'))
    Name = Column(String)
    PLOT = Column(MSVarBinary)
```

Things to remember

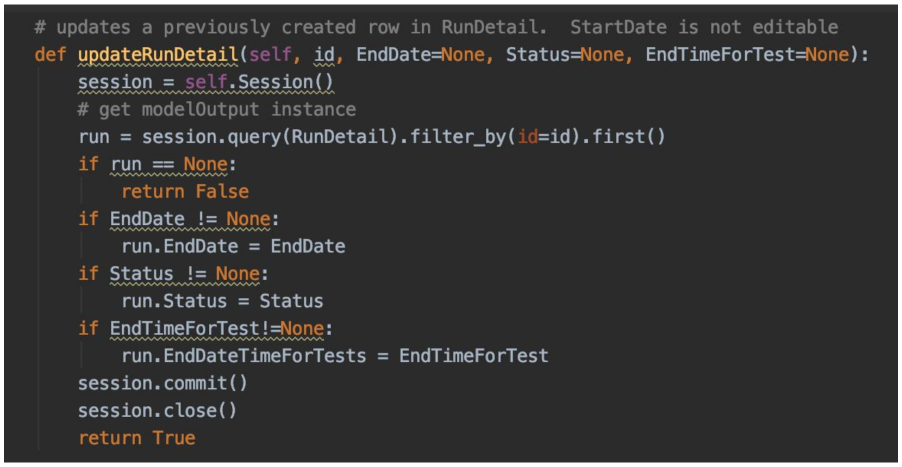
1. Here __tablename__ has to match the table name in the database.
2. There should always be a primary key and this is indicated as above for the field **id**
3. To include relationships between the tables, example each plot here corresponds to a model hence we include **ModelId** and indicate this relationship using the **ForeignKey** keyword and specify the table name where ModelId is primary key. Thus we give **TableName.columnName.**

Inserting a row in the table



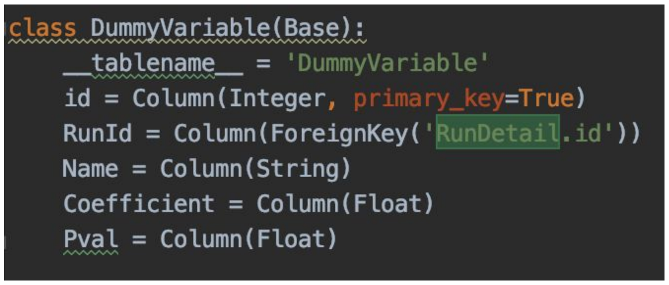
```
# creates a new row in RunDetail and returns the id of the row
def newRunDetail(self, StartDate=None, ModelType=None):
    session = self.Session()
    if StartDate == None:
        date = datetime.datetime.now()
        run = RunDetail(StartDate=date)
    else:
        run = RunDetail(StartDate=StartDate)
    if ModelType != None:
        run.ModelType = ModelType
    session.add(run)
    session.commit()
    id = run.id
    session.close()
    return id
```

Updating a row in the RunDetail table



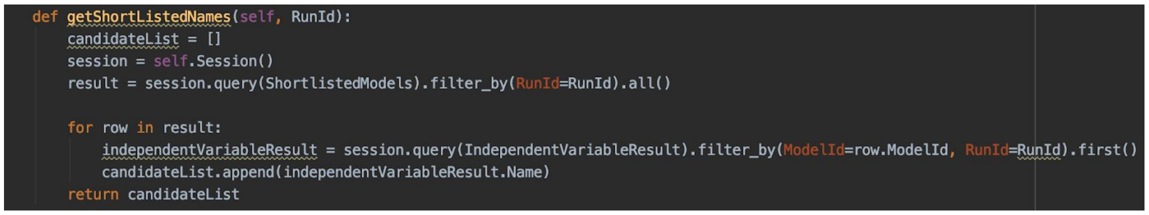
```python
# updates a previously created row in RunDetail.  StartDate is not editable
def updateRunDetail(self, id, EndDate=None, Status=None, EndTimeForTest=None):
    session = self.Session()
    # get modelOutput instance
    run = session.query(RunDetail).filter_by(id=id).first()
    if run == None:
        return False
    if EndDate != None:
        run.EndDate = EndDate
    if Status != None:
        run.Status = Status
    if EndTimeForTest!=None:
        run.EndDateTimeForTests = EndTimeForTest
    session.commit()
    session.close()
    return True
```

Example of RunId being foreign key:



```python
class DummyVariable(Base):
    __tablename__ = 'DummyVariable'
    id = Column(Integer, primary_key=True)
    RunId = Column(ForeignKey('RunDetail.id'))
    Name = Column(String)
    Coefficient = Column(Float)
    Pval = Column(Float)
```

Example of querying to get details for shortlisted models



```python
def getShortListedNames(self, RunId):
    candidateList = []
    session = self.Session()
    result = session.query(ShortlistedModels).filter_by(RunId=RunId).all()

    for row in result:
        independentVariableResult = session.query(IndependentVariableResult).filter_by(ModelId=row.ModelId, RunId=RunId).first()
        candidateList.append(independentVariableResult.Name)
    return candidateList
```

The above query is to obtain

The RunID and ModelID for the shortlisted models are stored in the ShortlistedModels table, where as the details like the independent variable details (for a ModelId) and dependent variable name (for a RunId).

The details can be fetched for a IndependentVariableResult table first by fetching the ModelId (or Independent Variables) for a RunID (i.e. the particular run).

Then for each ModelId we can obtain the details (in this case, the names of the shortlisted models) by querying the IndependentVariableResult table.

Web Application

Frontend

As the project is focused more on the data science and database aspects, the frontend portion is relatively basic. We envisioned to provide the users a simple user interface which automates most of the manual and time-consuming tasks such as data cleaning and uploading/selecting multiple data sets and comes back with the result. Frontend is built using JavaScript, HTML/CSS, PUG templating, Bulma CSS, and JQuery.

In terms of structuring the packages, we followed `node.js` convention: the pug templates are located in the `/views` folder, while the routes are located in the `/routes` folder. Front end specific javascript files are located in `/public/javascripts`.

Data Visualization

The data visualization code required to generate multiple instances of 7 graph types on the same page, and handle interactive events was written based on encapsulated chart code outlined here: https://bost.ocks.org/mike/chart/.

The seven types of graphs could be separated into two different types of charts with different configurations for each type. Allison wrote encapsulated, configurable and reusable chart code to dynamically generate each type of chart based on the data and configurations. Since the chart code is encapsulated, events on one chart don't interfere with events on other charts, and vice versa.

In more detail, the chart code is written in two different files and are split into "line chart" and "split line chart." In order to generate each chart, the javascript variable holding the json of data points is passed to the front end. The front end generated by pug templating generates the drop down menus and placeholder divs for each graph based on the json of datapoints. When the DOM is loaded, the chart.js script binds the data to each placeholder div, and calls the line chart or split line chart functions with configurations, depending on each type of chart.

Backend

The backend of the web application was written using `node.js` with express to handle routing and handle file uploads with multer. We used the timestamp to generate a unique filename (in case of multiple files with the same name) and will associate that with a new job.

After a file upload and submission of a new job, a user will be redirected to a jobs page. When this happens, the child process will be spawned by `node.js` to run the relevant data science modeling script.

The backend of the web application uses `express` to handle routing, and `body-parser` to parse form elements for user inputs.

The backend queries the database with Sequelize and uses custom functions to parse the data and pass it on for presentation to the front end. The backend uses the `moment.js` library to handle parsing and formatting of dates and timestamps.

Data Science

The data science part provides three main functionalities: data transformation, model creation, and outputting updates to database. All the following steps are carried out in python.

Data Transformations

Once the dataset that is uploaded by the user and the config file created from the frontend become available, the dataset will be transformed depending on the model being run i.e. either ARDL or One-Factor. As the transformations are being done, invalid data and null values will be dropped.

Model Creation & Updates to database

The model code received from the PNC Bank team comes in the form of a set of functions in python. Each model run generated after performing regression is categorized based on whether the model is statistically viable depending on the thresholds set by the user input.

After performing the statistical tests, the reasons of failure of the models is also presented to the user. The user can examine the reasons for the models that are not statistically viable and take decision if the user/modeler wants to make it a candidate.

We have created a separate script ‘afterShortlist.py’ for running all the later tests on candidates and shortlisted models. The other tests (including backtesting, stress-testing and sensitivity) are run after the user/modeler shortlists the models (either from candidate or rejected models).

In order to run the later tests (in ‘afterShortlist.py’), the intermediate data-frames like the regs, dep and base dataframe are required. These data-frames are converted into json and stored in the database during the first script.

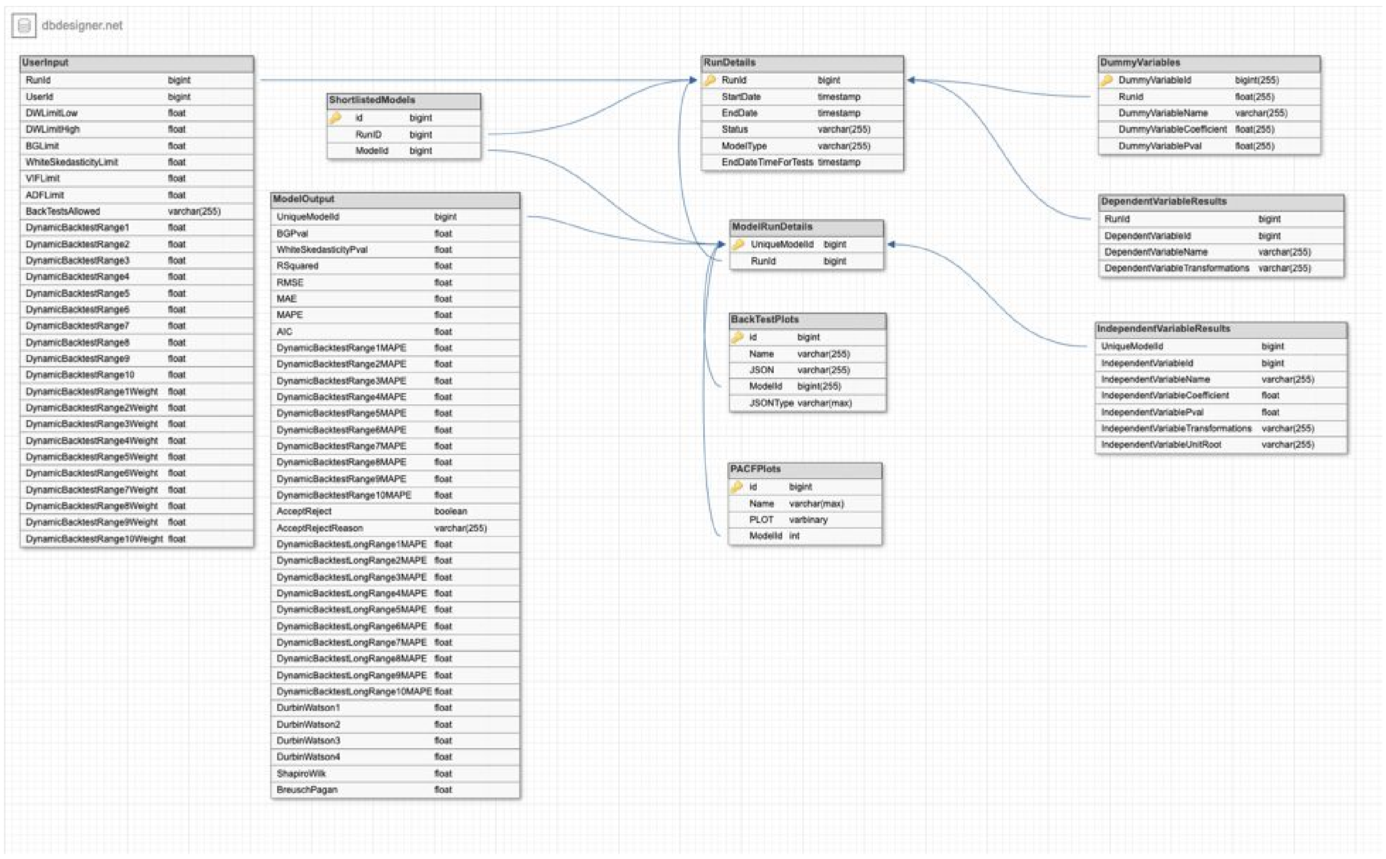
When the afterShortlist.py runs, these intermediate outputs are fetched from the database along with the candidate and shortlisted models and then the backtesting process is started.

The data-points for all the graphs being generated (Except PACF) are stored in the database in the form of json.

The PACF plot is stored in the form of a stream of bytes.

Data Flow

Database



UserInput

RunId	bigint
UserId	bigint
DWLimitLow	float
DWLimitHigh	float
BGLimit	float
WhiteSkedasticityLimit	float
VIFLimit	float
ADFLimit	float
BackTestsAllowed	varchar(255)
DynamicBacktestRange1	float
DynamicBacktestRange2	float
DynamicBacktestRange3	float
DynamicBacktestRange4	float
DynamicBacktestRange5	float
DynamicBacktestRange6	float
DynamicBacktestRange7	float
DynamicBacktestRange8	float
DynamicBacktestRange9	float
DynamicBacktestRange10	float
DynamicBacktestRange1Weight	float
DynamicBacktestRange2Weight	float
DynamicBacktestRange3Weight	float
DynamicBacktestRange4Weight	float
DynamicBacktestRange5Weight	float
DynamicBacktestRange6Weight	float
DynamicBacktestRange7Weight	float
DynamicBacktestRange8Weight	float
DynamicBacktestRange9Weight	float
DynamicBacktestRange10Weight	float

ShortlistedModels

id	bigint
RunID	bigint
ModelId	bigint

ModelOutput

UniqueModelId	bigint
BGPval	float
WhiteSkedasticityPval	float
RSquared	float
RMSE	float
MAE	float
MAPE	float
AIC	float
DynamicBacktestRange1MAPE	float
DynamicBacktestRange2MAPE	float
DynamicBacktestRange3MAPE	float
DynamicBacktestRange4MAPE	float
DynamicBacktestRange5MAPE	float
DynamicBacktestRange6MAPE	float
DynamicBacktestRange7MAPE	float
DynamicBacktestRange8MAPE	float
DynamicBacktestRange9MAPE	float
DynamicBacktestRange10MAPE	float
AcceptReject	boolean
AcceptRejectReason	varchar(255)
DynamicBacktestLongRange1MAPE	float
DynamicBacktestLongRange2MAPE	float
DynamicBacktestLongRange3MAPE	float
DynamicBacktestLongRange4MAPE	float
DynamicBacktestLongRange5MAPE	float
DynamicBacktestLongRange6MAPE	float
DynamicBacktestLongRange7MAPE	float
DynamicBacktestLongRange8MAPE	float
DynamicBacktestLongRange9MAPE	float
DynamicBacktestLongRange10MAPE	float
DurbinWatson1	float
DurbinWatson2	float
DurbinWatson3	float
DurbinWatson4	float
ShapiroWilk	float
BreuschPagan	float

RunDetails

RunId	bigint
StartDate	timestamp
EndDate	timestamp
Status	varchar(255)
ModelType	varchar(255)
EndDateTimeForTests	timestamp

ModelRunDetails

UniqueModelId	bigint
RunId	bigint

BackTestPlots

id	bigint
Name	varchar(255)
JSON	varchar(255)
ModelId	bigint(255)
JSONType	varchar(max)

PACFPlots

id	bigint
Name	varchar(max)
PLOT	varbinary
ModelId	int

DummyVariables

DummyVariableId	bigint(255)
RunId	float(255)
DummyVariableName	varchar(255)
DummyVariableCoefficient	float(255)
DummyVariablePval	float(255)

DependentVariableResults

RunId	bigint
DependentVariableId	bigint
DependentVariableName	varchar(255)
DependentVariableTransformations	varchar(255)

IndependentVariableResults

UniqueModelId	bigint
IndependentVariableId	bigint
IndependentVariableName	varchar(255)
IndependentVariableCoefficient	float
IndependentVariablePval	float
IndependentVariableTransformations	varchar(255)
IndependentVariableUnitRoot	varchar(255)

The database needs to keep tracks of the inputs by the user and the output produced by each model in the run.

Basically the run is tied to a single user inputted file of a certain format. Each run has multiple model runs (a model determined by which independent variable(s)). A model run has outputs which are calculated and in the end is accepted or rejected based on the limits specified in the user input for different values. The output includes the coefficients and p-values of the independent variables, dummy variables and lag variables associated with a run of the model, the r-squared result and the result of various backtests.

The schema design is tied closely to not only how end result data is stored from each model, but also how the web application will programmatically keep track of user input(s).

To handle migrations, we are using built in sequelize migrations for node.js, and writing Sequelize models to match the migrations.

The database has tables to store the model output.

1) **RunDetail** - Storing details for one run
2) **ModelRunDetail** - Storing details for model in a run
3) **UserInput** - Storing the user input for a particular run
4) **IndependentVariableResult** - Details of Independent variable
5) **DependentVariableResult** - Details of Dependent variable
6) **DummyVariable** - Details of Dummy variable
7) **IntermediateOutput** - Stores the intermediate output generated after statistical test and before shortlist. Stores 3 jsons for regs, dep and base dataframes.
8) **ModelOutput** - Stores all the output values for a run
9) **ShortlistedModels** - Stores the details of the shortlisted models for a run
10) **PACFPlots** - Stores the PACF plots for each model
11) **BackTestPlots** - Stores all the plots except for PACF.

Useful database queries :

To display output from table DependentVariableResult

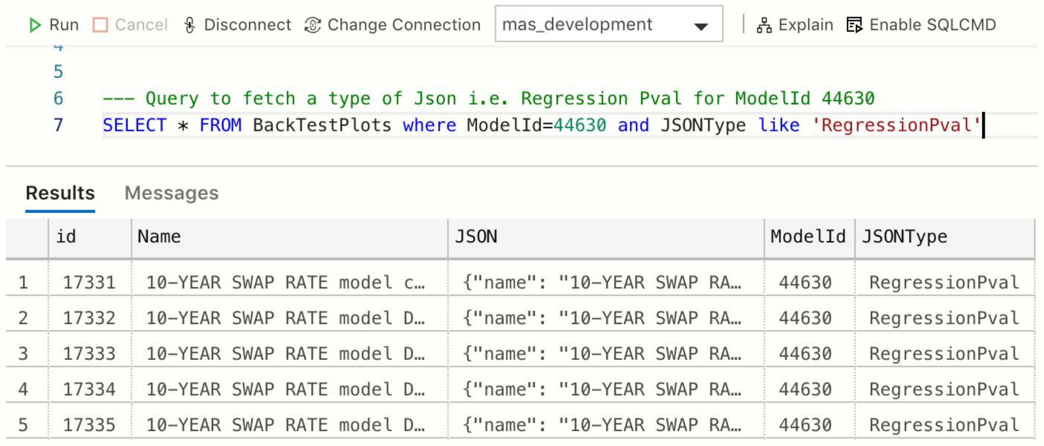
```
SELECT * FROM DependentVariableResult
```

Here '*' is a special character indicating all the columns.

To display output for ShortlistedModels table where the RunId is 557

```
SELECT * FROM ShortlistedModels where RunId=557
```

One more example:



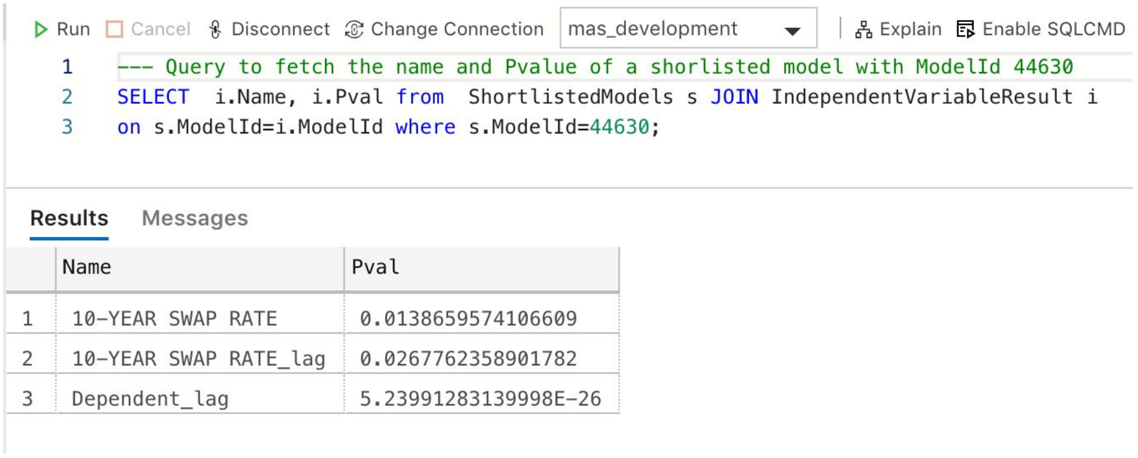
```
--- Query to fetch a type of Json i.e. Regression Pval for ModelId 44630
SELECT * FROM BackTestPlots where ModelId=44630 and JSONType like 'RegressionPval'
```

	id	Name	JSON	ModelId	JSONType
1	17331	10-YEAR SWAP RATE model c…	{"name": "10-YEAR SWAP RA…	44630	RegressionPval
2	17332	10-YEAR SWAP RATE model D…	{"name": "10-YEAR SWAP RA…	44630	RegressionPval
3	17333	10-YEAR SWAP RATE model D…	{"name": "10-YEAR SWAP RA…	44630	RegressionPval
4	17334	10-YEAR SWAP RATE model D…	{"name": "10-YEAR SWAP RA…	44630	RegressionPval
5	17335	10-YEAR SWAP RATE model D…	{"name": "10-YEAR SWAP RA…	44630	RegressionPval

To delete rows from ShortlistedModels table for RunId is 565

```
DELETE from ShortlistedModels where RunId= 565
```

Some queries including joins between tables



Run Cancel Disconnect Change Connection mas_development Explain Enable SQLCMD

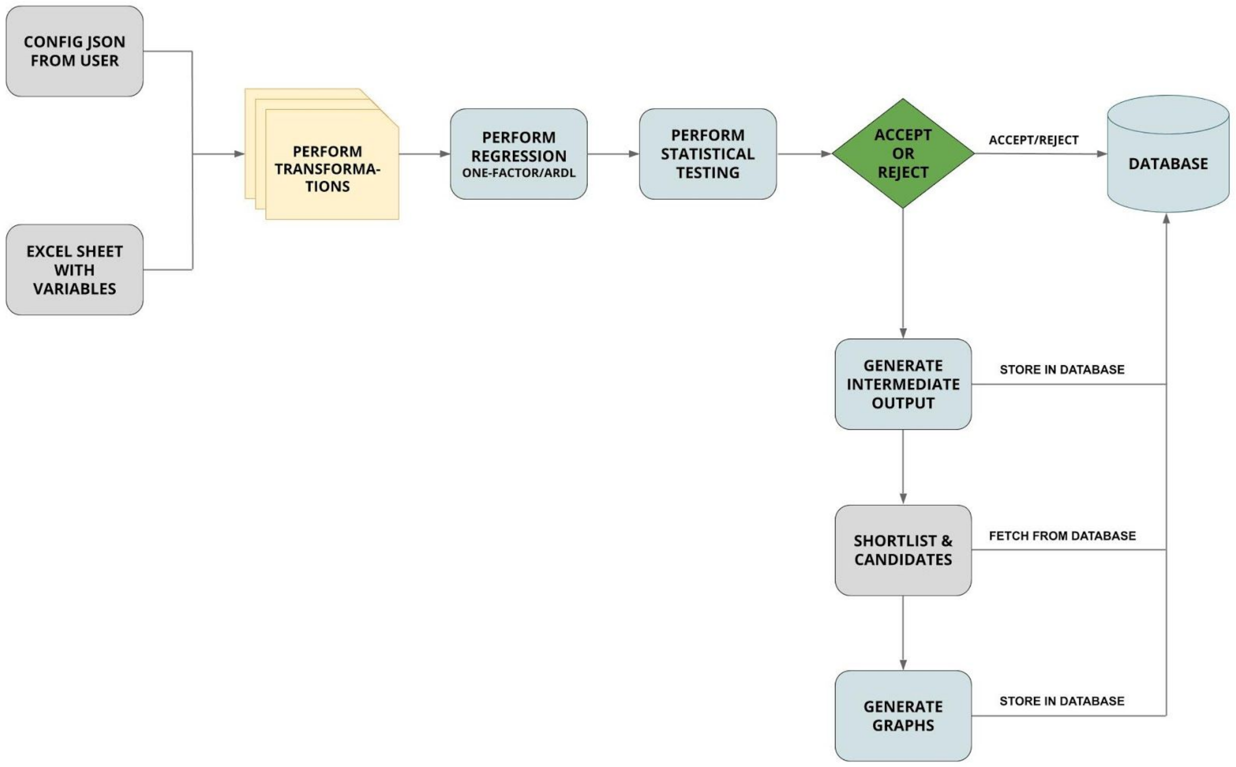
```sql
--- Query to fetch the name and Pvalue of a shorlisted model with ModelId 44630
SELECT  i.Name, i.Pval from  ShortlistedModels s JOIN IndependentVariableResult i
on s.ModelId=i.ModelId where s.ModelId=44630;
```

Results Messages

	Name	Pval
1	10-YEAR SWAP RATE	0.0138659574106609
2	10-YEAR SWAP RATE_lag	0.0267762358901782
3	Dependent_lag	5.23991283139998E-26

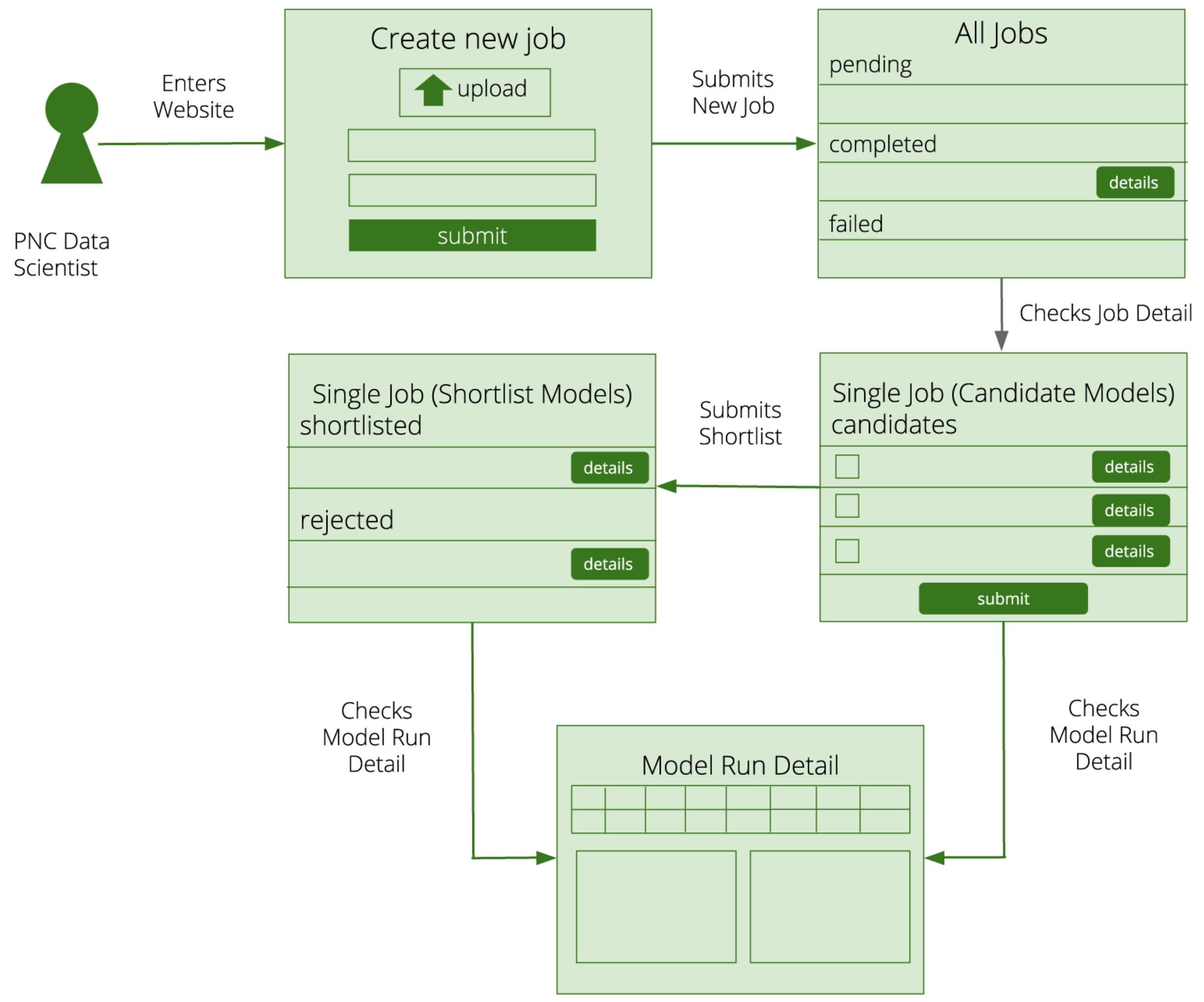
Further reference for the database queries the following link will be helpful
https://www.w3schools.com/sql/default.asp

Data Science Model Flow



The diagram above describes the flow of data science modeling script. When config json file and dataset are uploaded by a user and the data science script is spawned by the frontend, data transformation will be performed. After the transformations are completed, the regression and statistical tests will be performed and the intermediate outputs of the tests will be stored in the database. The user will be able to view the output of each model with the data of which tests passed or failed from the web application frontend. Then, the user will select which model runs to be included in `shortlist` and those model runs selected will under-go further testing including backtests, stress test, and out-out-time tests, and then generate additional outputs to be stored in database, including the json data points to generate interactive graphs.

UI Flow



The previous diagram goes over the flow of the front end for the web application. The target user is a PNC Data Scientist.

1. **Create New Job** - User can upload an excel sheet with a time series and set parameters such as what exclusion dates and what statistical tests to run
2. **All Jobs** - shows all jobs sorted by status
3. **Single Job (candidate models)** - Shows all the candidate models and failed models for a job. The user can choose to keep (shortlist) one or more models

4. **Single Job (shortlisted models)** - Shows the shortlisted and rejected models for a single job
5. **Model Run Detail** - Shows the details for a single model run. Shows graphs and calculated outputs.

- *For further details on web application, please refer to Build Instructions - the Web Application User Guide section.*

Technical Stack

Project Management/Planning

- dbdesigner.net

Data Cleaning/Modelling

- Python (pandas, statsmodels, scipy.stats, numpy, sklearn)

Back End Development and Database

- SQL
    - ORM Node: sequelize https://sequelize.readthedocs.io/en/v3/
    - ORM Python: sql alchemy https://www.sqlalchemy.org/
- Node.js
    - Multer: https://www.npmjs.com/package/multer
    - Express
    - Body-parser: https://www.npmjs.com/package/body-parser
    - Child processes

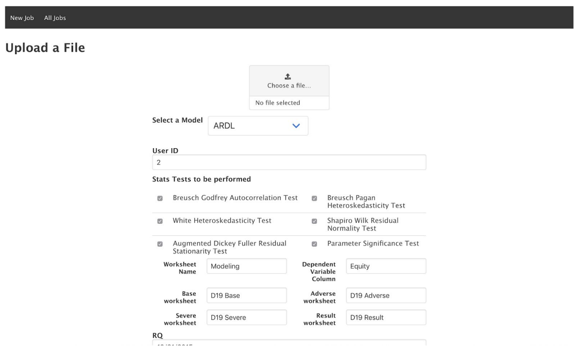
Front End Development

- D3.js for visualizations
- Pug Templating
- HTML/CSS
- Bulma CSS
- jQuery

Results & Next Steps

Result of the Project

USF-PNC team was successfully able to build a web application where a user can upload a time series excel file and select a model to run (of two: ARDL and One-Factor Regression) and set various parameters. Below are the screenshots of the application.



New Job All Jobs

Upload a File

Choose a file...
No file selected

Select a Model: ARDL

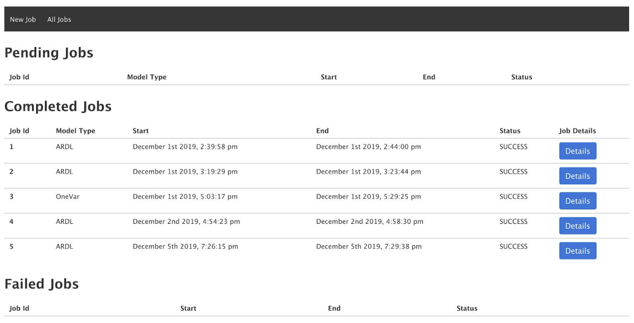
User ID: 2

Stats Tests to be performed

✓ Breusch Godfrey Autocorrelation Test	✓ Breusch Pagan Heteroskedasticity Test
✓ White Heteroskedasticity Test	✓ Shapiro Wilk Residual Normality Test
✓ Augmented Dickey Fuller Residual Stationarity Test	✓ Parameter Significance Test

Worksheet Name	Modeling	Dependent Variable Column	Equity
Base worksheet	D19 Base	Adverse worksheet	D19 Adverse
Severe worksheet	D19 Severe	Result worksheet	D19 Result

RQ



New Job All Jobs

Pending Jobs

Job Id	Model Type	Start	End	Status

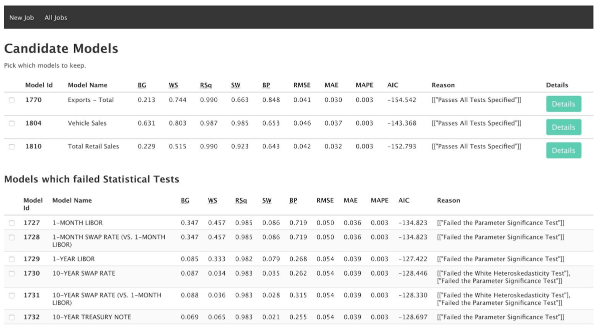
Completed Jobs

Job Id	Model Type	Start	End	Status	Job Details
1	ARDL	December 1st 2019, 2:39:58 pm	December 1st 2019, 2:44:00 pm	SUCCESS	Details
2	ARDL	December 1st 2019, 3:19:29 pm	December 1st 2019, 3:23:44 pm	SUCCESS	Details
3	OneVar	December 1st 2019, 5:03:17 pm	December 1st 2019, 5:29:25 pm	SUCCESS	Details
4	ARDL	December 2nd 2019, 4:54:23 pm	December 2nd 2019, 4:58:30 pm	SUCCESS	Details
5	ARDL	December 5th 2019, 7:26:15 pm	December 5th 2019, 7:29:38 pm	SUCCESS	Details

Failed Jobs

Job Id	Start	End	Status

Left: Landing page where user can input dataset and configure statistical tests and parameters.
Right: All jobs page which lists the jobs by their statuses



New Job All Jobs

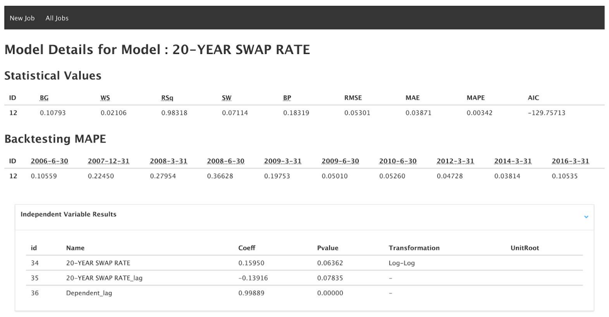
Candidate Models

Pick which models to keep.

Model Id	Model Name	BG	WS	RSq	SW	BP	RMSE	MAE	MAPE	AIC	Reason	Details
1770	Exports – Total	0.213	0.744	0.990	0.663	0.848	0.041	0.030	0.003	-154.542	[["Passes All Tests Specified"]]	Details
1804	Vehicle Sales	0.631	0.803	0.987	0.985	0.653	0.046	0.037	0.003	-143.368	[["Passes All Tests Specified"]]	Details
1810	Total Retail Sales	0.229	0.515	0.990	0.923	0.643	0.042	0.032	0.003	-152.793	[["Passes All Tests Specified"]]	Details

Models which failed Statistical Tests

Model Id	Model Name	BG	WS	RSq	SW	BP	RMSE	MAE	MAPE	AIC	Reason
1727	1-MONTH LIBOR	0.347	0.457	0.985	0.086	0.719	0.050	0.036	0.003	-134.823	[["Failed the Parameter Significance Test"]]
1728	1-MONTH SWAP RATE (VS. 1-MONTH LIBOR)	0.347	0.457	0.985	0.086	0.719	0.050	0.036	0.003	-134.823	[["Failed the Parameter Significance Test"]]
1729	1-YEAR LIBOR	0.085	0.333	0.982	0.079	0.268	0.054	0.039	0.003	-127.422	[["Failed the Parameter Significance Test"]]
1730	10-YEAR SWAP RATE	0.087	0.034	0.983	0.035	0.262	0.054	0.039	0.003	-128.446	[["Failed the White Heteroskedasticity Test"], ["Failed the Parameter Significance Test"]]
1731	10-YEAR SWAP RATE (VS. 1-MONTH LIBOR)	0.088	0.036	0.983	0.028	0.315	0.054	0.039	0.003	-128.330	[["Failed the White Heteroskedasticity Test"], ["Failed the Parameter Significance Test"]]
1732	10-YEAR TREASURY NOTE	0.069	0.065	0.983	0.021	0.255	0.054	0.039	0.003	-128.697	[["Failed the Parameter Significance Test"]]



New Job All Jobs

Model Details for Model : 20-YEAR SWAP RATE

Statistical Values

ID	BG	WS	RSq	SW	BP	RMSE	MAE	MAPE	AIC
12	0.10793	0.02106	0.98318	0.07114	0.18319	0.05301	0.03871	0.00342	-129.75713

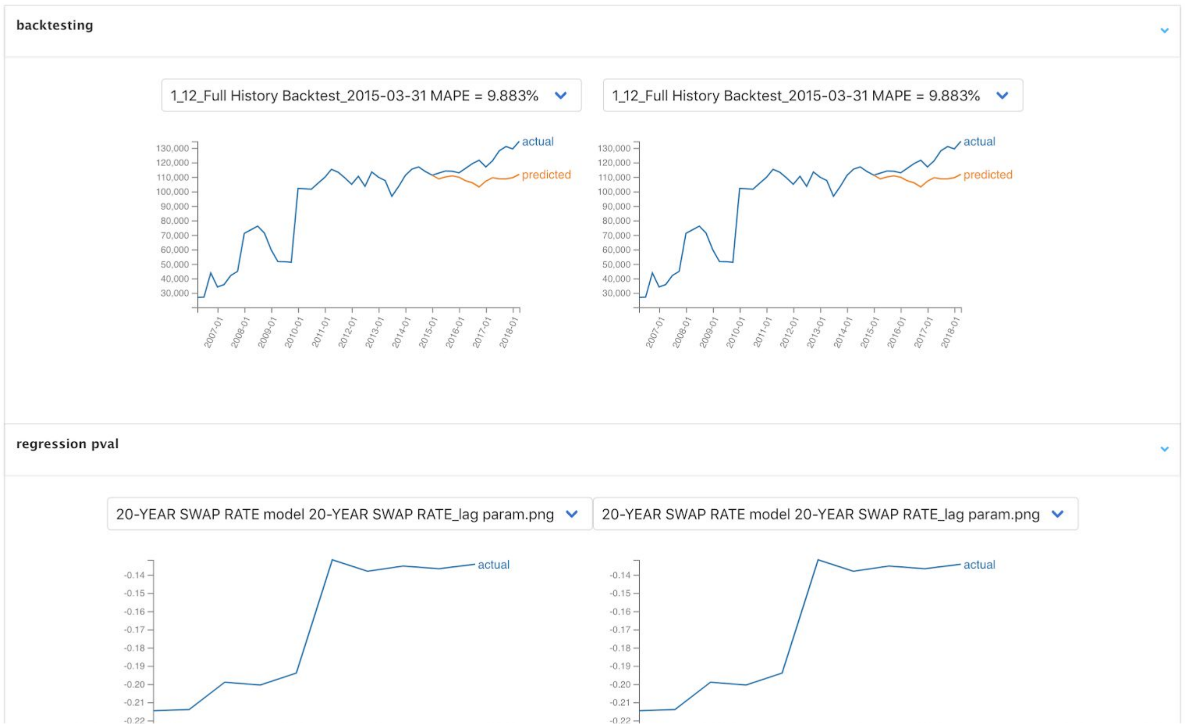
Backtesting MAPE

ID	2006-6-30	2007-12-31	2008-3-31	2008-6-30	2009-3-31	2009-6-30	2010-6-30	2012-3-31	2014-3-31	2016-3-31
12	0.10559	0.22450	0.27954	0.36628	0.19753	0.05010	0.05260	0.04728	0.03814	0.10535

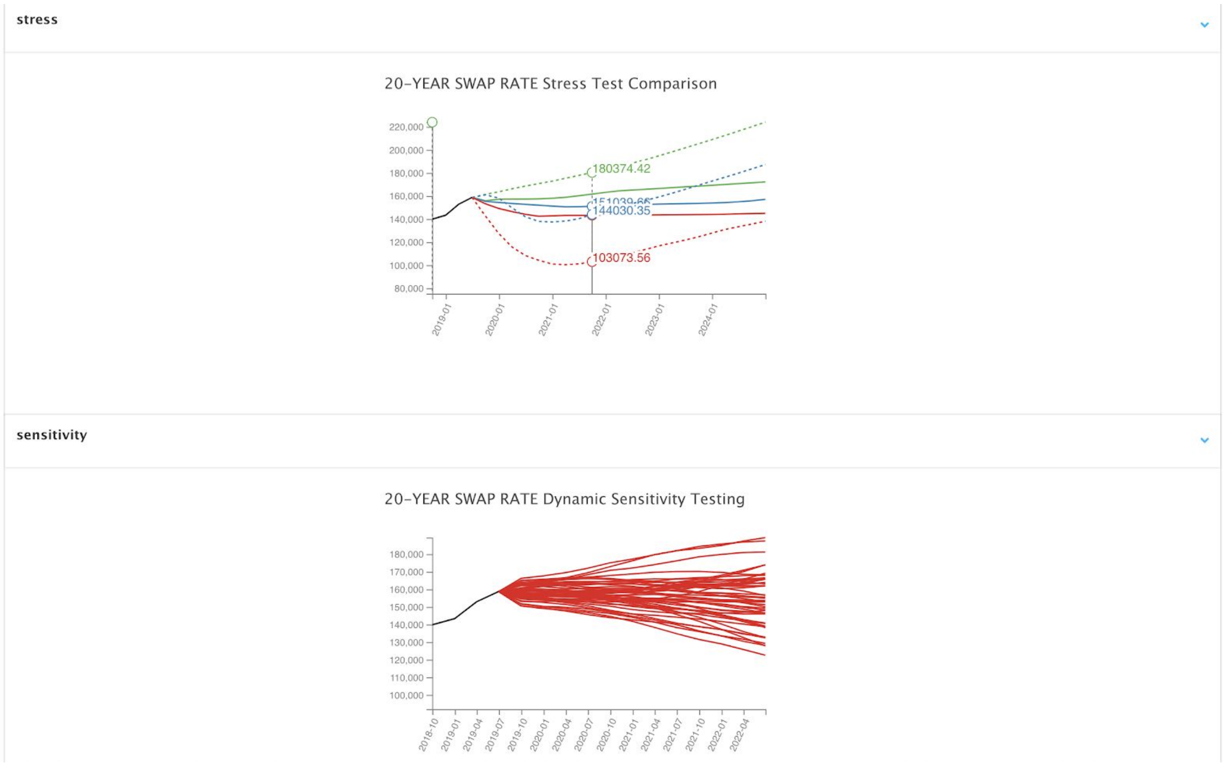
Independent Variable Results

id	Name	Coeff	Pvalue	Transformation	UnitRoot
34	20-YEAR SWAP RATE	0.15950	0.06362	Log-Log	
35	20-YEAR SWAP RATE_lag	-0.13916	0.07835	-	
36	Dependent_lag	0.99889	0.00000	-	

Left: After a job is complete, a user can view the model run intermediate outputs and select model runs to shortlist (run further testing on).
Right: User can view an individual model run's details.



- Interactive graphs on the model run detail page where a user can compare different instances of the same type of graph.


stress
20-YEAR SWAP RATE Stress Test Comparison
180374.42
151039.66
144030.35
103073.56
sensitivity
20-YEAR SWAP RATE Dynamic Sensitivity Testing

- Interactive graphs on the model run detail page where a user can view values on a graph

Next Steps

Improvements

- Finalization One Factor Regression Model
- Improvement on the UI
    - On the Job Detail page allowing the user to filter by the reasons of failure for the Failed Models.
    - Include a page showing all the Candidate Models.
- Comprehensive testing and documentation for all the testing
- Adding support for concurrency (multiple users running multiple runs at once)
    - Not using config.json, but storing the user input in the database. As there can be a situation, if different users run the job simultaneously, the config parameters can be replaced.
    - Locking a row or table while updating the database. This will not allow concurrent modifications to the same row.

Reference : https://groups.google.com/forum/?fromgroups=#!topic/sqlalchemy/8WLhbsp2nls

- Converting the web application to a containerized application for easier deployment
  - Advantages of Containerization : https://tsa.com/top-5-benefits-of-containerization/ https://hackernoon.com/what-is-containerization-83ae53a709a6
  - Reference / starting point : https://sysally.com/blog/deploy-node-js-and-mysql-in-docker-container/
- As the size of the database and hence tables will grow with respect to data, using joins to get the results from the database (example for Candidates, which involves querying different tables) will take larger time.

Additional Features

- Converting last static image graph to interactive data visualization
- Adding more models, such as two-factor regression and moving average model

Build Instructions

Github repository: https://github.com/azywong/mas-project
instructions to setup to run locally

Setting up MSSQL database

Installation

- Node.js and npm https://nodejs.org/en/
- Run the following commands to install packages
  - `npm install -g mssql`
  - `npm install -g tedious`
  - `npm install -g sequelize`
  - `npm install -g sequelize-cli`
- Make sure you have your mssql database setup

Setup

- After cloning the code from github
- Create config file at mas-webapp/config/config.json
  - Follow format of `exampleconfig.json` in that folder
  - Copy the contents of `exampleconfig.json` to a new file named config.json
  - At minimum replace the username and password in the example with the credentials for the database

    - These credentials are how the node.js and the python scripts communicate with the database
- In the terminal, navigate to the mas-project/mas-webapp directory and run
    - `sequelize db:migrate`
    - This runs the migrations to create the tables in the database

Setting up the web application environment

Installation

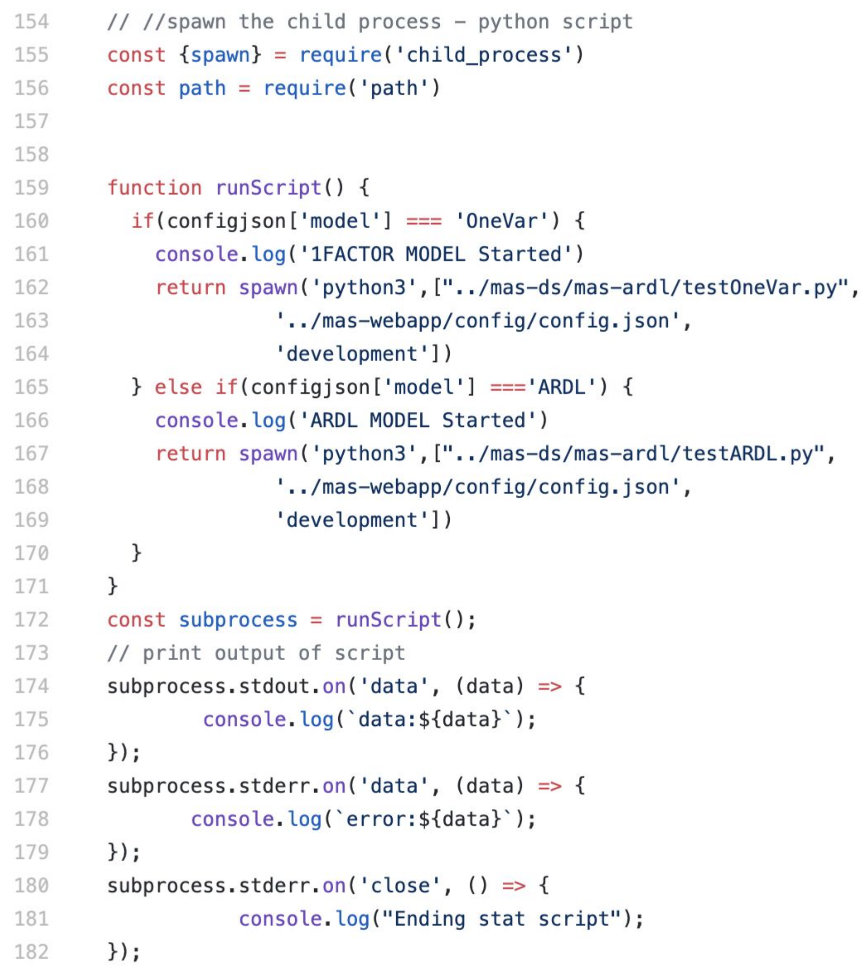
- navigate to the mas-project/mas-webapp directory
- Run the following to install necessary packages
    - `npm install`

Setup

- Run the following to start the web application normally
    - `npm start`
- Run the following to start the web application in development/debug mode (this listens for changes in the code)
    - `DEBUG=mas-webapp:* npm run devstart`

Integrating python modeling code to web application

- The python modeling script will be spawned whenever a new job is submitted.
- The user input from the main page will be saved as `"../mas-ds/mas-ardl/config.json”` and the code for spawning the child process from node.js will be triggered (located in `runScript()` function of “POST new job” in ../`mas-webapp/routes/jobs.js`).
- Based on the model type of in `config.json` file created, the type of data script to be run will be determined.
- The addition of the new python modeling script will be similar to the following snippet from ../`mas-webapp/routes/jobs.js`



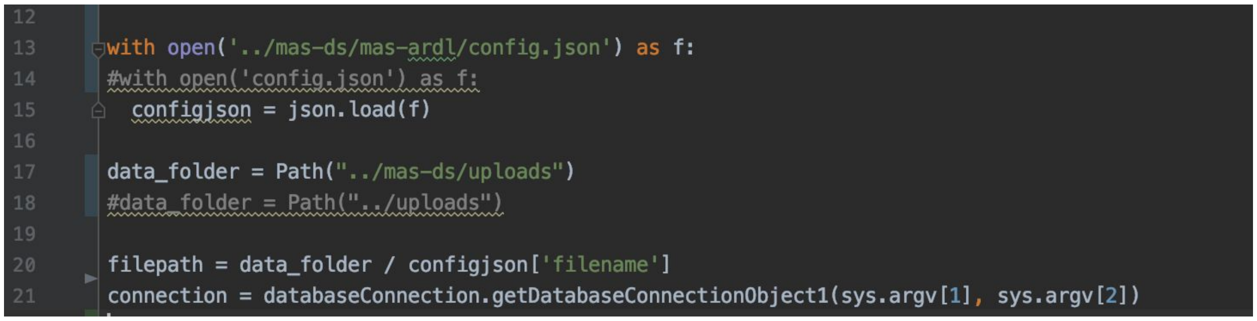
```javascript
// //spawn the child process - python script
const {spawn} = require('child_process')
const path = require('path')


function runScript() {
  if(configjson['model'] === 'OneVar') {
    console.log('1FACTOR MODEL Started')
    return spawn('python3',["../mas-ds/mas-ardl/testOneVar.py",
              '../mas-webapp/config/config.json',
              'development'])
  } else if(configjson['model'] ==='ARDL') {
    console.log('ARDL MODEL Started')
    return spawn('python3',["../mas-ds/mas-ardl/testARDL.py",
              '../mas-webapp/config/config.json',
              'development'])
  }
}
const subprocess = runScript();
// print output of script
subprocess.stdout.on('data', (data) => {
        console.log(`data:${data}`);
});
subprocess.stderr.on('data', (data) => {
       console.log(`error:${data}`);
});
subprocess.stderr.on('close', () => {
           console.log("Ending stat script");
});
```

Setting up data science python script

To set up the python script:

- Create a folder “**`uploads`**” folder in **`mas-ds`** folder.

- For running the python script with the web application give the path as in line 13 and 17 in the below screenshot.



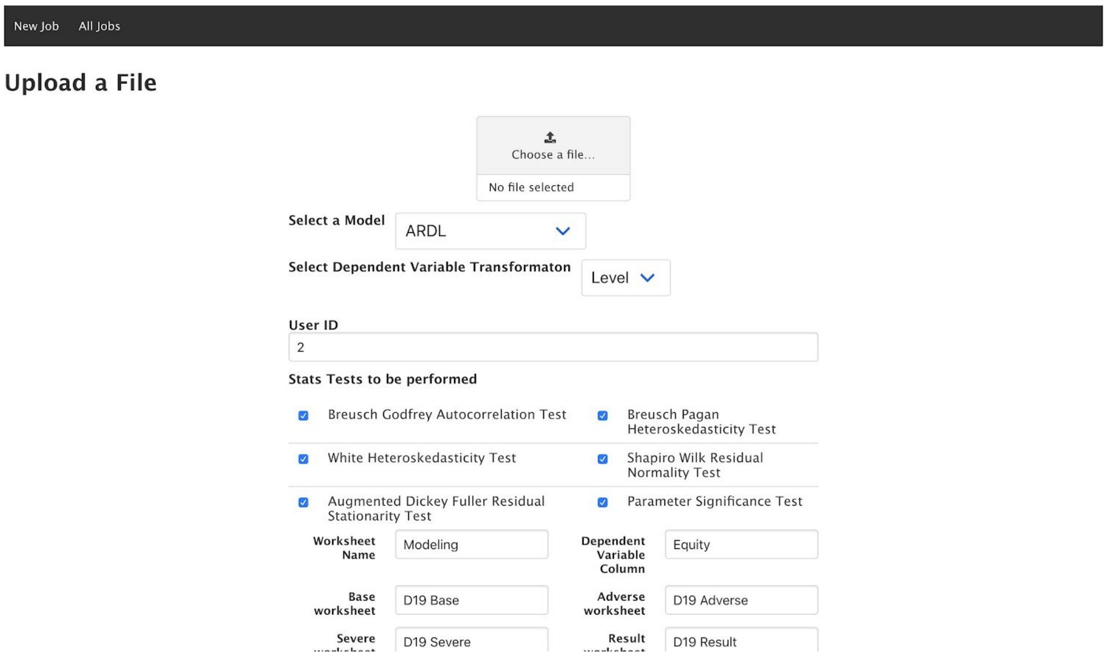
```python
12
13  with open('../mas-ds/mas-ardl/config.json') as f:
14  #with open('config.json') as f:
15    configjson = json.load(f)
16
17  data_folder = Path("../mas-ds/uploads")
18  #data_folder = Path("../uploads")
19
20  filepath = data_folder / configjson['filename']
21  connection = databaseConnection.getDatabaseConnectionObject1(sys.argv[1], sys.argv[2])
```

- For running the script from the terminal uncomment the line 14 and 18 instead of 13 and 17.
  For running the scripts individually from the terminal the command line arguments required are as follows:
  1. `python3 testARDL.py config/config.json development`
  2. `python3 afterShortlist.py config/config.json development <RunId>`

Here **`config.json`** is the json for configuring the database and **`development`** is the name of the database from config.json where the user/modeler wants the output to be stored.

The database connection can be initiated by calling the `getDatabaseConnectionObject1()` method from the `database.py`.

The Web Application User Guide



Upload a File

No file selected

Select a Model: ARDL

Select Dependent Variable Transformaton: Level

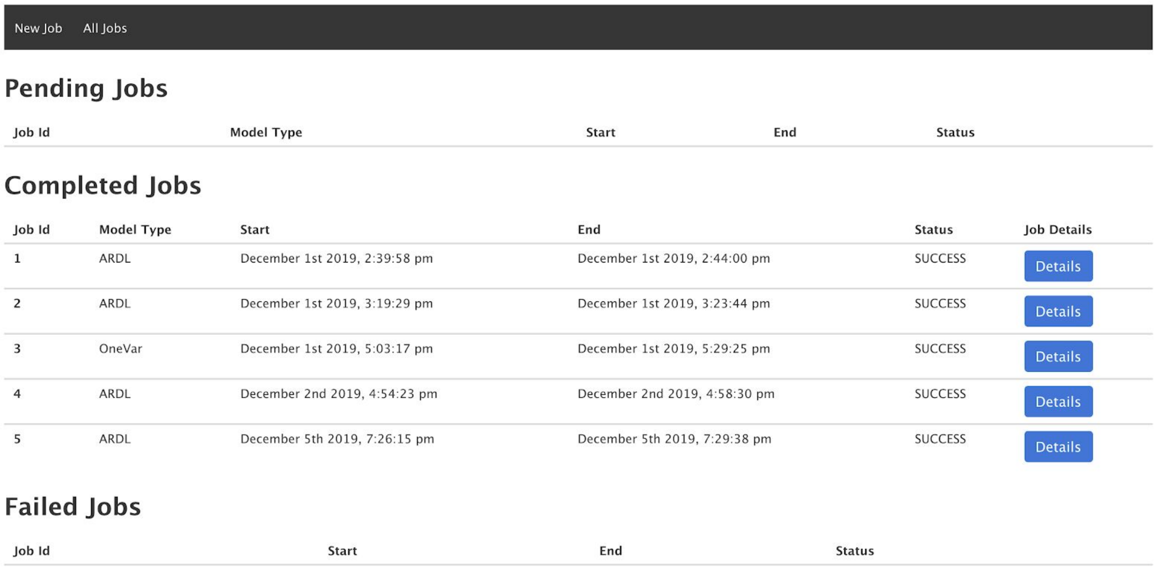
User ID: 2

Stats Tests to be performed

- ✓ Breusch Godfrey Autocorrelation Test
- ✓ Breusch Pagan Heteroskedasticity Test
- ✓ White Heteroskedasticity Test
- ✓ Shapiro Wilk Residual Normality Test
- ✓ Augmented Dickey Fuller Residual Stationarity Test
- ✓ Parameter Significance Test

Worksheet Name	Modeling	Dependent Variable Column	Equity
Base worksheet	D19 Base	Adverse worksheet	D19 Adverse
Severe	D19 Severe	Result	D19 Result

1. When a modeler enters the main page of the web application, he or she will be able to upload the dataset, select the model and statistical tests to be run, and configure other inputs related to the uploaded excel file.



Pending Jobs

Job Id	Model Type	Start	End	Status

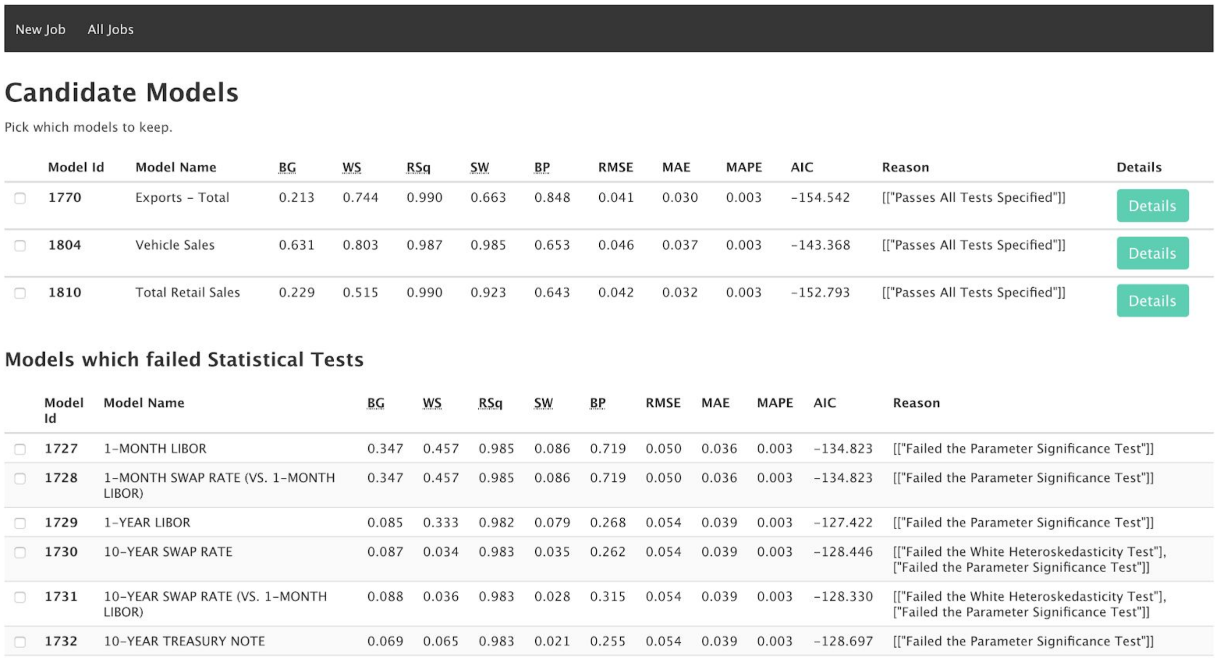
Completed Jobs

Job Id	Model Type	Start	End	Status	Job Details
1	ARDL	December 1st 2019, 2:39:58 pm	December 1st 2019, 2:44:00 pm	SUCCESS	Details
2	ARDL	December 1st 2019, 3:19:29 pm	December 1st 2019, 3:23:44 pm	SUCCESS	Details
3	OneVar	December 1st 2019, 5:03:17 pm	December 1st 2019, 5:29:25 pm	SUCCESS	Details
4	ARDL	December 2nd 2019, 4:54:23 pm	December 2nd 2019, 4:58:30 pm	SUCCESS	Details
5	ARDL	December 5th 2019, 7:26:15 pm	December 5th 2019, 7:29:38 pm	SUCCESS	Details

Failed Jobs

Job Id	Start	End	Status

2. Once the submit button is clicked, it will be redirected to `All jobs` page where the modeler will be able to see the statuses of all the job (including the one which he/she submitted).
    a. Note: Each job takes some time to complete (5-10 min). The job will be shown as completed after the script is finished AND the web page is refreshed.



Candidate Models

Pick which models to keep.

	Model Id	Model Name	BG	WS	RSq	SW	BP	RMSE	MAE	MAPE	AIC	Reason	Details
☐	1770	Exports - Total	0.213	0.744	0.990	0.663	0.848	0.041	0.030	0.003	-154.542	[["Passes All Tests Specified"]]	Details
☐	1804	Vehicle Sales	0.631	0.803	0.987	0.985	0.653	0.046	0.037	0.003	-143.368	[["Passes All Tests Specified"]]	Details
☐	1810	Total Retail Sales	0.229	0.515	0.990	0.923	0.643	0.042	0.032	0.003	-152.793	[["Passes All Tests Specified"]]	Details

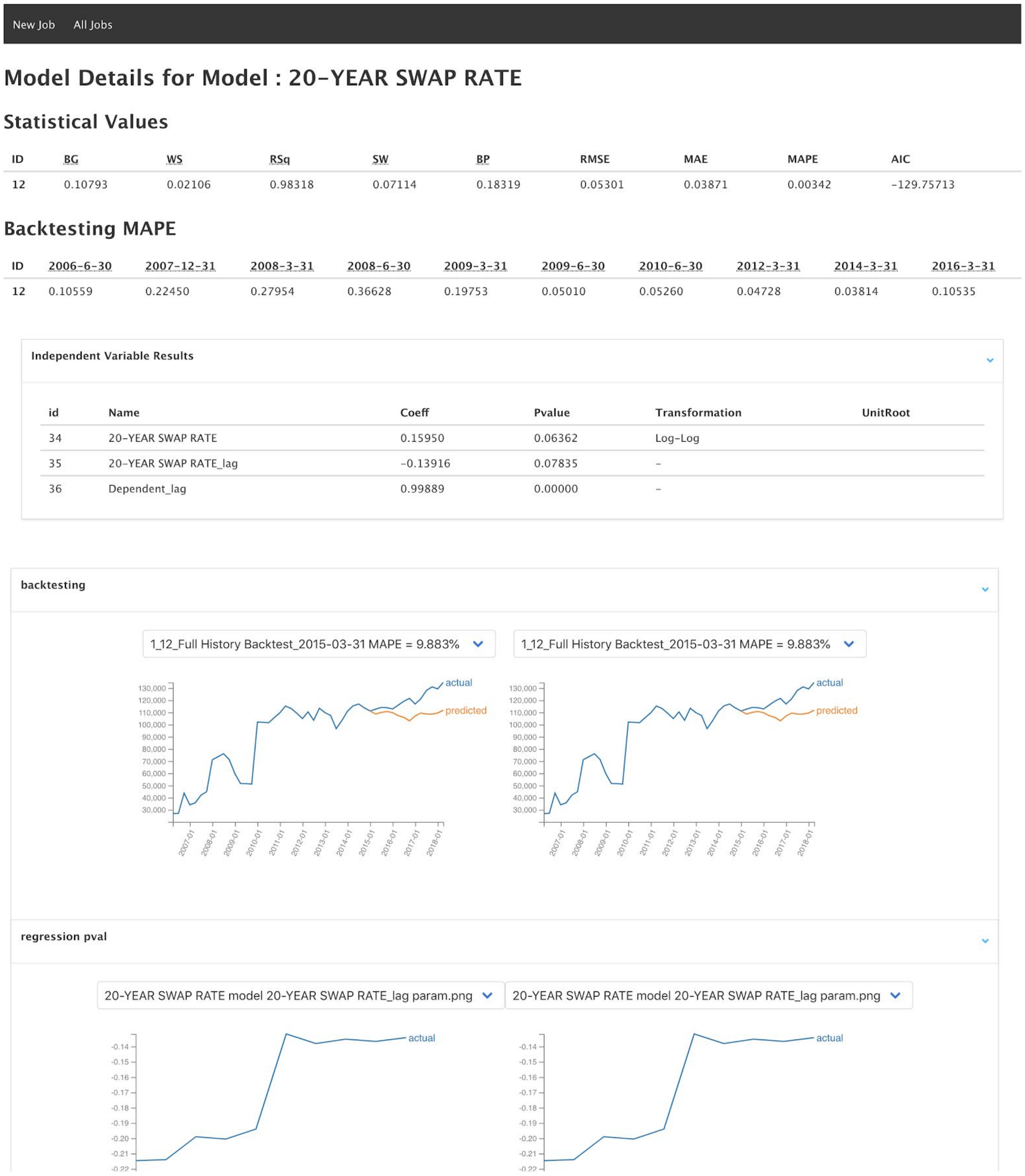
Models which failed Statistical Tests

	Model Id	Model Name	BG	WS	RSq	SW	BP	RMSE	MAE	MAPE	AIC	Reason
☐	1727	1-MONTH LIBOR	0.347	0.457	0.985	0.086	0.719	0.050	0.036	0.003	-134.823	[["Failed the Parameter Significance Test"]]
☐	1728	1-MONTH SWAP RATE (VS. 1-MONTH LIBOR)	0.347	0.457	0.985	0.086	0.719	0.050	0.036	0.003	-134.823	[["Failed the Parameter Significance Test"]]
☐	1729	1-YEAR LIBOR	0.085	0.333	0.982	0.079	0.268	0.054	0.039	0.003	-127.422	[["Failed the Parameter Significance Test"]]
☐	1730	10-YEAR SWAP RATE	0.087	0.034	0.983	0.035	0.262	0.054	0.039	0.003	-128.446	[["Failed the White Heteroskedasticity Test"], ["Failed the Parameter Significance Test"]]
☐	1731	10-YEAR SWAP RATE (VS. 1-MONTH LIBOR)	0.088	0.036	0.983	0.028	0.315	0.054	0.039	0.003	-128.330	[["Failed the White Heteroskedasticity Test"], ["Failed the Parameter Significance Test"]]
☐	1732	10-YEAR TREASURY NOTE	0.069	0.065	0.983	0.021	0.255	0.054	0.039	0.003	-128.697	[["Failed the Parameter Significance Test"]]

3. When the user clicks on `Details` button from `Completed Jobs` section, s/he will be directed to a page that shows the list of each model's preliminary statistical values and pass/fail log. Here, the user can select which models to be included in

the shortlist. Once the models are selected and `Keep this` button is clicked, the second batch of tests will be spawned.

   a. Note: The second data science script may take longer than the preliminary tests as it will be running more tests.



Model Details for Model : 20-YEAR SWAP RATE

Statistical Values

ID	BG	WS	RSq	SW	BP	RMSE	MAE	MAPE	AIC
12	0.10793	0.02106	0.98318	0.07114	0.18319	0.05301	0.03871	0.00342	-129.75713

Backtesting MAPE

ID	2006-6-30	2007-12-31	2008-3-31	2008-6-30	2009-3-31	2009-6-30	2010-6-30	2012-3-31	2014-3-31	2016-3-31
12	0.10559	0.22450	0.27954	0.36628	0.19753	0.05010	0.05260	0.04728	0.03814	0.10535

Independent Variable Results

id	Name	Coeff	Pvalue	Transformation	UnitRoot
34	20-YEAR SWAP RATE	0.15950	0.06362	Log-Log	
35	20-YEAR SWAP RATE_lag	-0.13916	0.07835	-	
36	Dependent_lag	0.99889	0.00000	-	

backtesting

regression pval

20-YEAR SWAP RATE model 20-YEAR SWAP RATE_lag param.png
actual
-0.14
-0.15
-0.16
-0.17
-0.18
-0.19
-0.20
-0.21
-0.22
20-YEAR SWAP RATE model 20-YEAR SWAP RATE_lag param.png
actual

4. Once the second round of tests are finished, the user will be able to click on `Details` on /jobs/:id page. Once `Details` button is clicked, the user will be able to

see detailed results of each model, including statistical test outputs, backtesting outputs, and interactive graphs.