I. Introduction

Medecipher is a Denver-based health IT start-up that aims to optimize clinical operations. They provide hospitals with forecasts of the expected number of patients in a given hour so that the hospital can appropriately allocate its resources. Medecipher’s current forecasting model, ARIMAX, helps their hospitals to better match the number of needed nurses to the volume of patients.

The goal of this project was to manage nurse staffing in the emergency environment more efficiently. In hospitals, there is currently a lot of variability in the input and output of patients. For this reason, it is difficult to predict the required amount of staffing for any given time. Using SageMaker, our first task was to create a more accurate and precise predictive model than the one Medecipher is currently using, working with years of hospital data. Nurses have variable-length shifts (8 hours, 10 hours, or 12 hours), and the rule-of-thumb in hospitals is to maintain a 4:1 ratio for patients to nurses. With this information, we created different suggestive nurse schedules, differentiated by how much the 4:1 ratio is violated by the model (20%, 15%, or 5% ratio violation).

Besides statistical modeling, we also worked on the front-end user interface of the Medecipher website. The current state of the Medecipher website allows for hospitals to select one given day for a few different staffing schedules. To better handle the problem, we designed a six-week calendar for clients to use, with color coding and possible alterations for the client to use. We also wanted the client (hospitals) to be able to filter specifications for what they would like in their schedule.
II. Functional Requirements

Task 1 [Improve the Forecasting Model]:

1. Set up AWS workflow to build & deploy models using SageMaker - Jupyter Notebooks
2. Understand current ARIMAX forecasting model parameters, input, accuracy, and challenges
3. Build models for producing hourly census & arrival forecasts using SageMaker
   a. Models to be built: Prophet, LSTM, N-Beats, Simple Feed Forward, DeepAR
4. Publish code to AWS Code Commit using Jupyter notebook workflow
5. Create a GUI to input parameter values
6. Produce control charts for error monitoring

Task 2 [User Interface/Front-end]:

1. Create angular component that displays a schedule planner based on the forecasting model from Task 1
2. Features:
   a. Filters: Fiscal Year (FY), block, day of week
   b. Printing: Ability to print the daily schedule (4a today thru 11a tomorrow)
3. Data interface
   a. Incorporate static, hard-coded data to the front-end using JSON or other data objects.

III. Non-Functional Requirements

1. The code for the forecasting model must be written in Python, R, or AMPL.
2. The program must use the Jupyter Notebook development environment.
3. The forecasting model must be built using AWS SageMaker.
4. UI mockups must have a comprehensive workflow.
IV. System Architecture and Technical Design

Task 1 (Improve the Forecasting Model):

DeepAR is a machine learning algorithm built into Amazon Sagemaker. DeepAR uses Recurrent Neural Networks to make both point and probabilistic predictions. Training data is inputted as a JSON file and prediction requests are made to an endpoint using JSON lines commands. The workflow for implementing the DeepAR model is shown below in Figure 1.

![DeepAR Model Training Workflow](image)

*Figure 1: DeepAR workflow*

Figure 2 below is the workflow for the Simple Feed Forward and N-Beats models. Both these models are built into the GluonTS toolkit for time series modeling, and therefore have similar workflows. Census data is read in from a CSV file and split up into training and testing data; these sets of data are then converted into ListDataSets objects so that they are in the correct format to make predictions. An estimator is created with all the hyperparameters for the model; this is the only part of the code that is different for the two models as each model has its own defined estimator class. A predictor is created from the estimator and is used to make forecasts.
The Prophet model was developed by Facebook to predict website traffic. It uses Fourier analysis to fit seasonal (periodic) data. In addition there are other interesting features that the model can use in its predictions including: standard holiday’s from the country in which the data is taken from, weekly seasonality, specific Fourier orders for each regressor etc. In this application we are using Prophet to predict hospital ER patient arrivals. Figure 3 Below shows the workflow for Prophet.

**Task 2 (User Interface/Front-end):**

Ultimately, the front end of the project will be created by Medecipher in AngularJS. The website will be connected to the rest of the website via a python application, such as Flask, to update the
schedule planner and nursing forecasting model in real time. Each different component of the website will have its own corresponding type script, CSS and HTML so that everything is styled appropriately and located where it should be on the site. As of right now, we have created mock up models of potential designs to pursue with the website.

**Final Website Mockup**

**Model Description:**

**Schedule module:**

This model is based on a more interactive spreadsheet style planner. It also is based on the scheduling software used at golf courses.

**Nurses Column:**

Each nurse has their own row. Clicking on the [↓] in the nurse row provides information on the selected nurse. To register a nurse for a specific time, the user needs to just click on a box in the grid, and the nurse will be scheduled for that time. Clicking on a box with a shift already scheduled removes the shift for that hour.

**Warning Row:**

The colored row above the time is the warning indicator row. If there is no warning, it will simply light up as a green box. If there is a warning, the respective column for the hour will change color to the most severe warning and create a pulldown menu ( [↓] ) containing a description of all the warnings for a given hour.
Role Selector:
This selects which role to display the schedule for.

Week Selector:
This selects which week to display the schedule for.

Print Button:
Brings the user to the print screen.

Add Nurse Button:
The “Add Nurse” button creates a popup that will ask for information on the nurse being added. After being added, a nurse will appear in a new role for the role(s) they were selected for. The other information can be accessed from the ↓ in the respective nurse’s row.

Limitations:
- Shift must be punched in hour by hour.
- Potential display issue from the end of a week to the beginning of the next.
- Removing a nurse from one role could end up removing them from all roles. If not, reregistering that nurse could create a duplicate.
- Roles are in separate views and may be difficult to compare with each other.

Possible Small Changes:
- Add a way to schedule a shift from a Start to Finish.
- Connecting to a database that contains “Nurse” objects would make “ADD NURSE” much easier.
- Make it obvious how many nurses are needed at any given hour so the manager doesn’t have to guess and get a warning first.
V. Quality Assurance

Task 1 (Back End):
- Code review
  - During one meeting, we determined that the LSTM model was proving too difficult to implement, due to lack of documentation and little versatility, so we moved on to other algorithms such as simple feed forward and N-Beats.
- User acceptance testing
  - Error metrics of new model should beat error metrics of previous model and work well over time.
  - Met with Medecipher and ran through Jupyter notebooks for validation.
  - Each of us on the data science side met with Medecipher’s data science specialist, Kevin, one-on-one to ensure the quality of our Jupyter notebooks. All of us were told that the notebook code and documentation is acceptable for their use.

Task 2 (User Interface/Front-end):
- Design Mock-ups for review before implementing design
  - A discussion and/or wireframe was thought through before the next stage of the UI is implemented.
- Weekly review meetings with Parker from Medecipher
  - Every Thursday, we met with the main connection between us and Medecipher, Parker, to ensure that we understood the task at hand and how to proceed with the project.
  - Meetings with Medecipher occur 2-3 times a week in order to make sure priorities stay in line with expectations. Every other week there is a larger scale meeting with Medecipher in which Task 1 and Task 2 work correlate with 10+ members of Medecipher to produce a plan for the following weeks.
  - Within these meetings, we also discussed the requirements of the final project, as they changed on more than one occasion.
VI. Results

Task 1 (Forecasting/Back End):

Throughout trying to improve the forecasting algorithm, we experimented with 5 different models: DeepAR, Prophet, Simple Feed Forward, N-Beats, and LSTM. The ultimate goal in each of these models is to minimize the desired error metrics mean absolute percentage error (MAPE), root mean square error (RMSE), and mean absolute error (MAE), and to reach better error metrics than the current model Medecipher is using, called ARIMAX.

DeepAR

Figure 4 shows the graph of a six-week forecast from DeepAR versus the test data. The DeepAR model has slightly worse error metrics than the ARIMAX model. This could be improved with more hyperparameter tuning. Because DeepAR takes a long time to train models and requires creating and using an endpoint, it is more time-consuming to tune than other models.

![Figure 4: DeepAR Prediction vs. Test Data](image)

After creating the error monitoring chart in Figure 5, we decided to try averaging the models together as shown in Figure 6 to see if it is more accurate than Arimax. The error metrics for the combined model were slightly better than both Arimax and DeepAR. It could be worth exploring combining the models to improve accuracy.

![Figure 5: Error monitoring chart for DeepAR vs. Arimax](image)
Figures 7 and 8 show the error metrics for DeepAR compared to Arimax for a six and two week forecast respectively. Due to time constraints, long training time, and model limitations the metrics for the combined model and some other data are missing. DeepAR did not perform better than Arimax over a one-year cycle.

### Table 1: Error Metrics for DeepAR and Arimax

<table>
<thead>
<tr>
<th>train_start_date</th>
<th>train_end_date</th>
<th>forecast_start_date</th>
<th>forecast_end_date</th>
<th>Arimax</th>
<th>DeepAR</th>
</tr>
</thead>
<tbody>
<tr>
<td>2019-03-01</td>
<td>2019-09-30</td>
<td>2019-10-01</td>
<td>2019-10-31</td>
<td>MAE: 0.15 RMSE: 0.25 MAPE: 10%</td>
<td>MAE: 0.13 RMSE: 0.22 MAPE: 8%</td>
</tr>
<tr>
<td>2019-04-01</td>
<td>2019-11-30</td>
<td>2019-12-01</td>
<td>2019-12-31</td>
<td>MAE: 0.16 RMSE: 0.26 MAPE: 11%</td>
<td>MAE: 0.14 RMSE: 0.24 MAPE: 9%</td>
</tr>
<tr>
<td>2019-05-01</td>
<td>2019-12-31</td>
<td>2019-01-01</td>
<td>2020-01-31</td>
<td>MAE: 0.17 RMSE: 0.27 MAPE: 12%</td>
<td>MAE: 0.15 RMSE: 0.25 MAPE: 10%</td>
</tr>
</tbody>
</table>

### Figure 6: DeepAR and Arimax Combined Prediction

![Figure 6](image_url)

### Figure 7: DeepAR vs. Arimax 6-week Error Metrics

![Figure 7](image_url)

### Figure 8: DeepAR vs. Arimax 2-week Error Metrics

![Figure 8](image_url)
Prophet

Figure 9 below shows an example of a Prophet forecast (shown in orange) in comparison to the actual number of ER patients (shown in blue) for a six week period.

Figure 9: Example Prophet Prediction vs Test Data

A benefit of the Prophet model is that it uses Fourier transforms to fit each model, instead of gradient descent like other machine learning methods; this means that it doesn’t take very long to train a model for testing. Because of this convenient fact many different possible combinations of hyperparameters could be tested to see which ones performed best. Overall, 540 different combinations of hyperparameters were tested. Error metrics for each 6-week testing period were collected for all 540 combinations of the following hyperparameters: 9 different training period lengths, 20 different hourly Fourier orders, whether or not a day-of-the-week integer was included, and whether or not U.S. holiday data was included. Figure 10 below shows the best possible combination of hyperparameters for each 6-week period.

<table>
<thead>
<tr>
<th>Start Train (datetime)</th>
<th>End Train (datetime)</th>
<th>Start Forecast (datetime)</th>
<th>End Forecast (datetime)</th>
<th>Forecast Duration (# weeks)</th>
<th>RMSE</th>
<th>Weekday Regressor Included?</th>
<th>U.S. Holiday Information Included?</th>
<th>Hourly Seasonality Fourier Order</th>
</tr>
</thead>
<tbody>
<tr>
<td>2/3/2019 00:00</td>
<td>7/9/2019 23:00</td>
<td>6</td>
<td>10/20/2019 23:00</td>
<td>5</td>
<td>6.7303311</td>
<td>no</td>
<td>no</td>
<td>6</td>
</tr>
<tr>
<td>2/3/2019 00:00</td>
<td>4/20/2019 23:00</td>
<td>22</td>
<td>4/22/2019 23:00</td>
<td>6</td>
<td>5.3702445</td>
<td>no</td>
<td>no</td>
<td>7</td>
</tr>
<tr>
<td>2/3/2019 00:00</td>
<td>6/1/2019 23:00</td>
<td>18</td>
<td>7/13/2019 23:00</td>
<td>6</td>
<td>4.787651</td>
<td>no</td>
<td>yes</td>
<td>2</td>
</tr>
<tr>
<td>2/3/2019 00:00</td>
<td>7/11/2019 23:00</td>
<td>24</td>
<td>8/24/2019 23:00</td>
<td>6</td>
<td>4.7061014</td>
<td>no</td>
<td>yes</td>
<td>2</td>
</tr>
<tr>
<td>2/3/2019 00:00</td>
<td>8/24/2019 23:00</td>
<td>30</td>
<td>10/5/2019 23:00</td>
<td>6</td>
<td>5.4260013</td>
<td>no</td>
<td>no</td>
<td>12</td>
</tr>
<tr>
<td>2/3/2019 00:00</td>
<td>10/5/2019 23:00</td>
<td>36</td>
<td>11/16/2019 23:00</td>
<td>6</td>
<td>6.3879429</td>
<td>no</td>
<td>no</td>
<td>2</td>
</tr>
<tr>
<td>2/3/2019 00:00</td>
<td>11/16/2019 23:00</td>
<td>42</td>
<td>12/28/2019 23:00</td>
<td>6</td>
<td>6.7312990</td>
<td>no</td>
<td>yes</td>
<td>2</td>
</tr>
<tr>
<td>2/3/2019 00:00</td>
<td>12/28/2019 23:00</td>
<td>48</td>
<td>3/8/2019 23:00</td>
<td>6</td>
<td>n/a</td>
<td>n/a</td>
<td>n/a</td>
<td>n/a</td>
</tr>
<tr>
<td>2/3/2019 00:00</td>
<td>3/8/2019 23:00</td>
<td>54</td>
<td>3/23/2019 23:00</td>
<td>6</td>
<td>7.5966248</td>
<td>no</td>
<td>no</td>
<td>18</td>
</tr>
</tbody>
</table>

Figure 10: The best RMSE value for each combination of hyperparameters using the Prophet model

Only one of the 6-week period’s RMSE value outperformed ARIMAX as shown in Figure 11 below:
Thus, even when fully tuned, the Prophet model rarely beats ARIMAX. The conclusion is that Prophet is not a better model than ARIMAX for this application.

### Simple Feed Forward

The client’s current model outperforms this model in terms of all three reported error metrics; therefore, this model does not meet our client’s functional requirements. Figure 12 shows the error metrics for six-week forecasts using the Simple Feed Forward model and compares it to the error metrics from ARIMAX. The green indicates that a model produces better error metrics than the other for that training period, while red indicates the model performs more poorly.

### N-Beats

This model produces slightly worse error metrics than the ARIMAX model for some of the six-week blocks. Given more time to tune the hyperparameters for specific blocks in the fiscal year, this model could produce improved metrics. Figure 13 shows the graph of a six-week forecast from N-Beats and the test data.
N-Beats was used to run both two-week and six-week forecasts. Figures 14 and 15 show the error metrics for each forecast length compared to the error metrics for ARIMAX. These figures are color-coded such that the model with the better error metric for that training period is highlighted green, and the model with the worse error metric is highlighted red. From Figures 14 and 15, it is clear that ARIMAX outperforms N-Beats for both the two-week and six-week forecasts over a one year cycle.

<table>
<thead>
<tr>
<th>End Train</th>
<th>End Forecast</th>
<th>ARIMAX MAE</th>
<th>ARIMAX RMSE</th>
<th>ARIMAX MAPE</th>
<th>ARIMAX OSE</th>
<th>N-Beats MAE</th>
<th>N-Beats RMSE</th>
<th>N-Beats MAPE</th>
<th>N-Beats OSE</th>
</tr>
</thead>
<tbody>
<tr>
<td>4/20/2019</td>
<td>6/1/2019</td>
<td>4.054394</td>
<td>5.368517</td>
<td>0.335044</td>
<td>4.721882</td>
<td>4.677498</td>
<td>6.074622</td>
<td>0.331355</td>
<td>5.50209</td>
</tr>
<tr>
<td>6/1/2019</td>
<td>7/13/2019</td>
<td>3.460312</td>
<td>4.450026</td>
<td>0.343689</td>
<td>3.661305</td>
<td>3.938129</td>
<td>5.11222</td>
<td>0.33547</td>
<td>4.541053</td>
</tr>
<tr>
<td>7/13/2019</td>
<td>8/24/2019</td>
<td>3.607251</td>
<td>4.629062</td>
<td>0.347111</td>
<td>3.975303</td>
<td>4.029145</td>
<td>5.16964</td>
<td>0.339115</td>
<td>4.49488</td>
</tr>
<tr>
<td>8/24/2019</td>
<td>10/5/2019</td>
<td>4.287617</td>
<td>5.71923</td>
<td>0.31099</td>
<td>4.968171</td>
<td>4.65502</td>
<td>6.162937</td>
<td>0.330648</td>
<td>5.297149</td>
</tr>
<tr>
<td>10/5/2019</td>
<td>11/16/2019</td>
<td>4.004722</td>
<td>5.272637</td>
<td>0.386437</td>
<td>4.658459</td>
<td>4.386376</td>
<td>5.74175</td>
<td>0.422951</td>
<td>4.943494</td>
</tr>
<tr>
<td>11/16/2019</td>
<td>12/28/2019</td>
<td>4.632309</td>
<td>6.01252</td>
<td>0.338734</td>
<td>5.207099</td>
<td>5.040016</td>
<td>6.595489</td>
<td>0.335578</td>
<td>5.790252</td>
</tr>
<tr>
<td>12/28/2019</td>
<td>2/8/2020</td>
<td>5.022607</td>
<td>6.59298</td>
<td>0.322758</td>
<td>5.911756</td>
<td>6.573953</td>
<td>8.425699</td>
<td>0.364757</td>
<td>7.50885</td>
</tr>
<tr>
<td>2/8/2020</td>
<td>3/21/2020</td>
<td>5.105865</td>
<td>6.3732</td>
<td>0.394469</td>
<td>5.313599</td>
<td>6.188191</td>
<td>8.308261</td>
<td>0.413328</td>
<td>7.304069</td>
</tr>
</tbody>
</table>

Figure 14: N-Beats vs. ARIMAX 6-week Error Metrics
The overarching goal of the user interface side of the project was to develop a potential model for Medecipher to implement into their current website to help schedule and adjust hospital nursing schedules. Originally, we were expected to code and develop a website in AngularJS, but after discussing our lack of web development experience with our clients, our project was changed to help them think of design ideas for the current website. We have developed a few different models, all with different layouts and user interface options, and gone through the strengths and limitations of each of them to assist Medecipher in deciding what to fully implement.

### Unimplemented Features

**Task 1 (Forecasting model):**
- Explore combining models to improve error metrics

**Task 2 (UI/Frontend):**
- We did not end up using AngularJS for our final project, as learning and implementing AngularJS was a little out of our experience area.

### Lessons Learned
- Training deep learning models takes a long time. Increasing the number of epochs produces more accurate forecasts, but it can also result in the model taking hours to train;
this is also true for several other hyperparameters. In general, we can create pretty accurate forecasts but the tradeoff is increased training time.

- To code in AngularJS, the user must be completely organized in every aspect of their project. It is very difficult to develop a website that is actually worth implementing just because of how many different files are required for one page.
- Between efficiently communicating, understanding clear goals, and delivering a product, working for a real-world client is not easy.
- Staying organized over the course of a project is very helpful in determining what is feasible by the end of the project.
- Communicating with your client is truly the best way to progress in the project. If there are any questions at all, asking our client would immediately clear up issues and problems.
- Make sure your client understands what you are capable of so that they have realistic expectations.
- Python is a very useful language for constructing deep learning models. The programming language offers several libraries and packages with pre-existing deep learning models already built in.
VII. Appendix

All Front End Mockups

Design 1

![Design 1 Diagram]

Design 2

![Design 2 Diagram]
Design 3

Design 4
Error Metrics

Mean Absolute Error:

\[ MAE = \frac{1}{n} \sum_{h=1}^{n} |e_{T+h}| \]

Root Mean Squared Error:

\[ RMSE = \sqrt{\frac{1}{n} \sum_{h=1}^{n} (e_{T+h})^2} \]

Mean Absolute Percent Error:

\[ MAPE = \frac{1}{n} \sum_{h=1}^{n} \frac{|p_{T+h}|}{|e_{T+h}|} \]

Mean One-Sided Error:

\[
\max(\text{Actual Count} - \text{Forecast}, 0) / \text{(number of positive values)}
\]