Forecasting the weather with deep learning

A predictive analysis of daily temperature and precipitation in 21 United States cities

STATS 604 - Project 4

Due December 4th, 2023

Contents

1 Introduction

1.1 Background

Weather prediction is a notoriously challenging task due to the uncertainty inherent in dynamic meteorological systems. It is not uncommon for short-term forecasts of temperature, precipitation, and other conditions to be off the mark — these errors are a nuisance for everyday human activity and can negatively impact sectors like aviation and tourism [Kul03; Wil+18]. Weather forecasts have traditionally been made by using supercomputers to simulate massive physical systems based on complex meteorological equations. This approach is known as numerical weather prediction, and it has been the predominant forecasting method for decades [PK19].

In recent years, however, meteorologists, computer scientists, and statisticians have discovered that deep neural networks are a powerful tool for producing accurate weather forecasts [Sch+21; Lam+23]. As a result, there has been a shift away from numerical weather prediction and toward models based on machine learning. Our focus in this report is on the latter — we train models to predict temperature and precipitation up to four days into the future for 21 cities throughout the United States. Specifically, we produce forecasts of the minimum, average, and maximum daily temperature in these cities, as well as whether it snowed and whether any precipitation occurred. We develop a transparent and reproducible workflow via Git and Docker — our models can be trained (and retrained, if desired) and run on a daily basis to make predictions for the next four days in each city.

1.2 Objectives and outline

The remainder of this report is motivated by the following questions:

1. What are the best data sources for training a model for our prediction tasks?
    - The performance of our final models will be assessed using METAR (Meteorological Aerodrome Report) data. Is METAR data also the best option for training our models, or are there other data sets that are better suited for training? Do the benefits of using a different data set outweigh the advantage of training on the same data set that will be used for assessment?
2. What class of models is most suitable for these prediction tasks?
    - Do neural methods consistently outperform naive baseline models (e.g., models that plug in historical averages or previous observations as future predictions)?
3. How can we organize our training, evaluation, and prediction pipeline in a reproducible way via Git and Docker?
    - What time and storage constraints might one encounter when they attempt to reproduce our work, and how can we proactively overcome these obstacles?

Section 2 addresses the first of these questions — we compare the advantages of the METAR data set to an alternative (Meteostat) and explain our decision to utilize a combination of the two. We also explain our preprocessing methods in detail. In section 3, we discuss two candidate models — long short-term memory (`LSTM`) networks and neural hierarchical interpolation (`NHITS`) — and outline our procedure for training temperature models with the former and precipitation models with the latter. We evaluate these models on held-out test data from the past year in section 4, with a focus on how their predictive accuracy varies across cities and whether they perform better than several baseline models. Section 5 describes how we use Git and Docker to package and present our workflow, which can be used to train our models on all of our training data and report predictions for future dates. Finally, we conclude in section 6 by discussing the limitations of our analysis.

2 Data

We elected to use a combination of two different sources of weather data: METAR and Meteostat. METARs are specifically formatted strings containing weather information for pilots and meteorologists. Iowa State University maintains a repository of METAR data, which can be queried using the `pmetar` package in `R` [Cwi23]. Meteostat is an open-source repository that pulls weather data from several sources, namely the National Oceanic and Atmospheric Administration (NOAA) [Met23].

Both data sources report information on temperature, precipitation, snow, and other weather-related information. Meteostat is significantly faster to pull since data is recorded at an hourly level compared to the five-minute intervals present in the METAR data set. However, the Meteostat data contains far more missing values for several variables, including condition codes that indicate whether it actively rained or snowed during a particular hour. Additionally, the METAR data set will be used to assess our predictions, and training on the same data set that is used at testing time is ideal. At the time we made this decision, our best option for obtaining and decoding METAR data was the `pmetar` package, which was fairly slow. As a result, we decided to use METAR for precipitation and snow only, and Meteostat for all other covariates. We pulled data from the two sources for the past 10 years, from January 1st, 2014 through November 30th, 2023. Section 2.1 and section 2.2 detail our process of obtaining the original data and cleaning it at an hourly level for Meteostat and METAR, respectively. Section 2.3 explains how the two data sets were joined and aggregated to a daily level for the purposes of modeling.

2.1 `Meteostat` data for temperature and atmospheric conditions

Our main data set, obtained from Meteostat using the `meteostat` library in `Python` [Lam23], contained hourly data from weather stations at all 21 airports of interest, which are listed in Figure 1. Specifically, we obtained hourly data on temperature (in °C), dew point, humidity, precipitation amount, wind direction, wind speed, and atmospheric pressure, along with the latitude, longitude, and elevation of each airport weather station. The raw `Meteostat` data also included columns for total amount of sunshine, weather condition codes, wind gust speed, and snow depth, but many (if not all) of the values in these columns were missing. As a result, determining whether it was actively raining or snowing during a particular hour based on the Meteostat data was at best ambiguous and at worst impossible. This was one of the reasons we chose to use METAR for precipitation and snow, as described in section 2.2.

While intended to be hourly, Meteostat did not report information for every hour at every airport. For temperature, dew point, humidity, pressure, and wind speed, we imputed missing values by interpolating linearly between the closest two known values. If only one hour was missing, this means that the imputed value was the midpoint between the values from the previous hour and the next hour. If multiple hours were missing, we assumed a constant rate of change across the hours and interpolated accordingly. For wind direction, we replaced each missing value with the most recent non-NA value from the same airport. This is because wind direction was reported in degrees (0-360), and thus it could shift from, e.g., 10 degrees to missing to 350 degrees in three consecutive hours. In these cases, it does not make sense to linearly interpolate using the midpoint of 10 and 350 since wind direction presumably decreased from 10 to 350 rather than increased. Instead of entirely rescaling the wind direction variable, we implemented the simpler imputation strategy mentioned above.

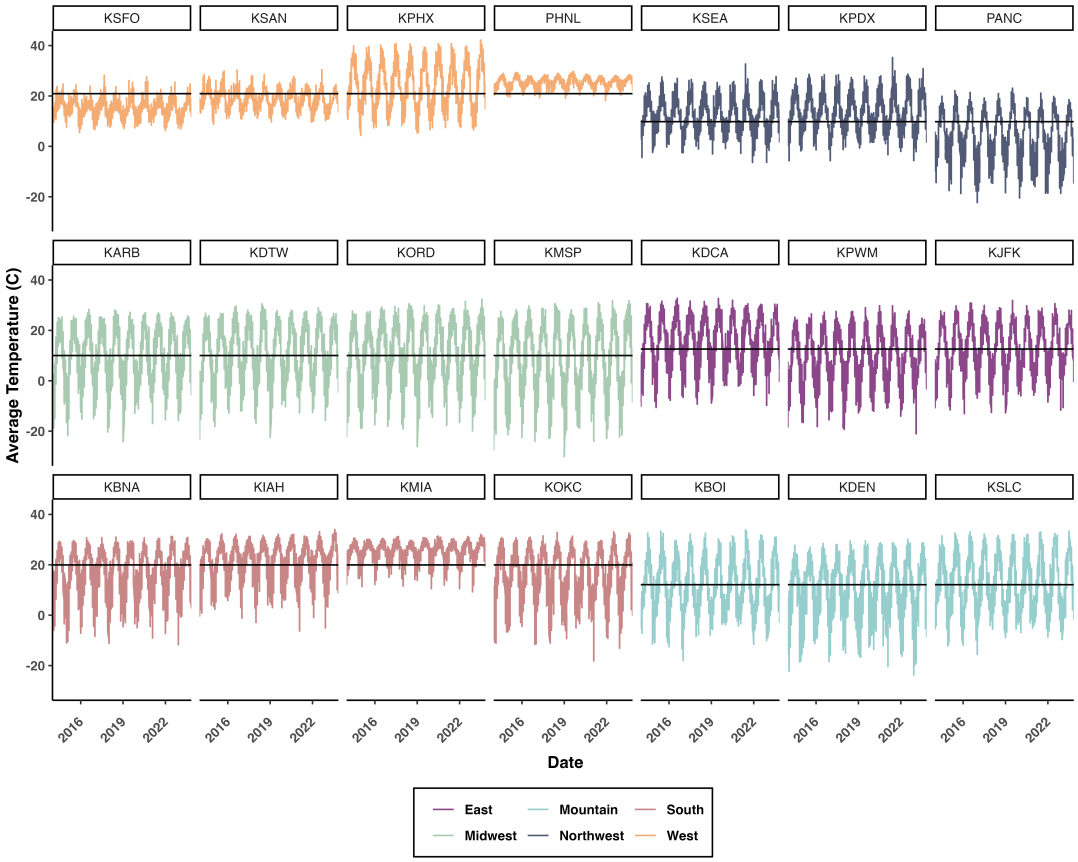
After imputing the missing values, we calculated the minimum, average, and maximum daily temperature at each airport. Figure 1 displays the average daily temperature for all 21 airports, grouped by region. Group means are indicated by the black horizontal lines. Unsurprisingly, all airports follow some sort of seasonality trend, although it is more pronounced in regions with more extreme seasons. Airports in the same region tend to experience similar temperatures, although Anchorage (PANC), Honolulu (PHNL), and Phoenix (KPHX) are notable exceptions to this. Despite this, the shared trends suggest that a single model trained on data from all 21 airports is more sensible than 21 individual models tailored to each airport, as the latter would be unable to capture the shared seasonal patterns in Figure 1.

2.1.1 Incorporating data from neighboring airports

It is reasonable to assume that weather conditions in a radius around a particular airport may be predictive of future weather at that airport. As such, we also pulled Meteostat data from the three airports closest to each airport of interest. Using the `meteostat` library in `Python`, we determined the three closest airports in terms of the geographical distance. The distance (in tens of kilometers) from the center airport to each neighboring airport was also recorded and used as a covariate.

However, we did not include METAR data on snow and precipitation for these neighboring airports due to concerns about the amount of time it would take to pull that data. Instead, we determined rain and snow based on the precipitation amounts and temperatures recorded in the Meteostat data. Specifically, if the precipitation amount at a particular airport during a particular hour was nonzero, we assumed that precipitation occurred during that hour. Similarly, if the precipitation amount was nonzero and the temperature was below freezing, we assumed that it actively snowed during that hour. This process

Figure 1: Average daily temperature (in °C) at 21 United States airports, Jan 2014 through Nov 2023



did not perfectly replicate the METAR data, but it provided a fairly close match. Also, since we are not directly predicting precipitation and snow for these neighboring airports but rather using them as covariates, we considered this method to be sufficient for our purposes. Other than these two variables, we pulled the same covariates from the neighboring airports as we did for the main 21 airports, and we imputed missing values for the neighboring airports using linear interpolation.

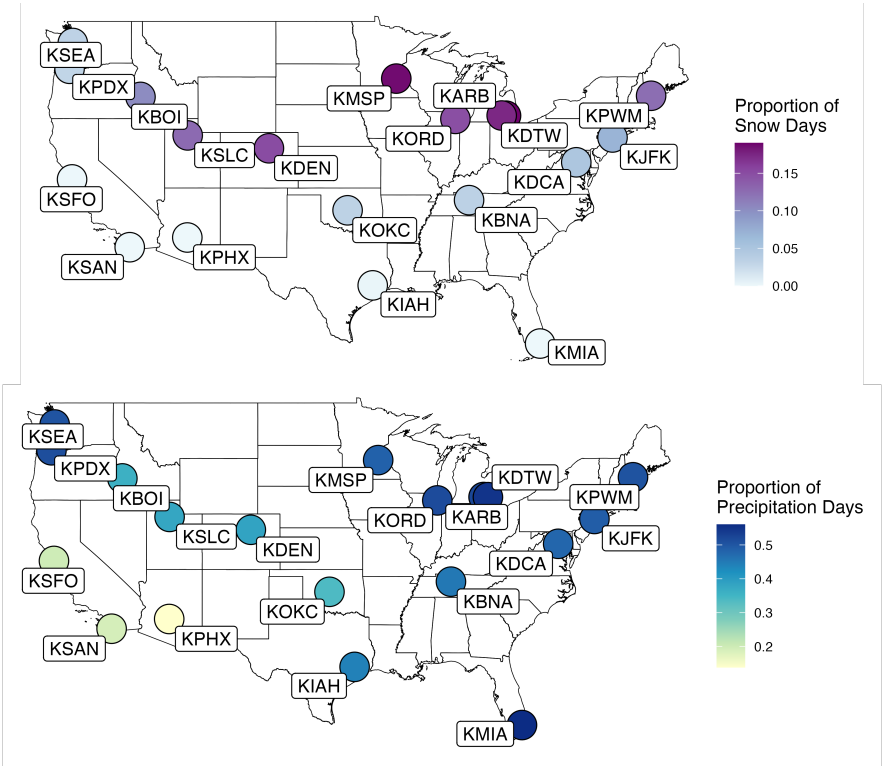
2.2 `METAR` data for precipitation

Since the accuracy of our predictions will be assessed using METAR data, we originally intended to exclusively use METAR data to train our models. Unfortunately, the `pmetar` package in `R` does not allow the user to specify time granularity; instead, it returns data at approximately five minute intervals. As a result, we found that pulling METAR data was slow. For efficiency reasons, we elected to use METAR only for precipitation and snow, even though the METAR data contains roughly the same weather information as Meteostat.

The precipitation and snow indicators in our data set are essentially the same as those that will be used to test our model. Specifically, we considered it to have snowed at a particular airport during a particular hour if one of the following weather condition codes was present: `SN` (snow), `SG` (snow grains), or `GS` (snow pellets). Precipitation was slightly more complicated. The ground truth for precipitation is based on precipitation values returned from the `Metar` library in `Python`, and unlike this `Python` library, the `pmetar` package in `R` does not directly return information on precipitation. Instead, we recognized in the `pmetar` data that precipitation is indicated when `P####` is present in the METAR string, where the `#`s are integers. The precipitation value (in hundredths of inches) is the number following the `P`. For example, the presence of `P0005` in the METAR string indicates precipitation of 0.05 inches. There is one exception to this — `P0000` indicates precipitation of 0.005 inches, instead of 0. If precipitation is 0, the pattern `P####` is absent from the string entirely.

After decoding the strings in this manner, the data set consisted of two binary variables indicating the occurrence of snow or any precipitation, respectively, at approximately five-minute intervals for each

Figure 2: Occurrence of precipitation in 19 U.S. cities, Jan 2014 through Nov 2023
(Anchorage, AK (p_{snow} = 0.202, p_{precip} = 0.561) and Honolulu, HI (p_{snow} = 0, p_{precip} = 0.586) omitted)



airport. We then aggregated the data to an hourly level, such that the precipitation and snow variables took a value of 1 if the precipitation or snow conditions were met at any point during an hour, and a value of 0 otherwise. There were approximately 2,000 cases where data was missing for an entire hour. If data was also missing immediately before and after the missing hour in question, we imputed a 0. If data was missing for one of the neighboring hours but available for the other, we imputed the value that matched the hour with available data. If data was available for both of the neighboring hours and the values matched, we used those values; if the neighboring values conflicted, we imputed values based on the probability of snow/any precipitation for the airport and month in question.

Figure 2 displays the proportion of days on which snow (top) or any precipitation (bottom) occurred at each airport. Unsurprisingly, the proportion of snow days varies considerably across the airports — those in the south (Miami, Houston, Phoenix, San Diego, and San Francisco) experienced almost no snow between 2014 and the present, while those in Alaska, the Midwest, or the mountain regions experienced it on 10 to 15 percent of days. Since there are so many airports with no snow days, the classification task for snow is extremely unbalanced. On the other hand, precipitation (i.e., rain or snow) is significantly more balanced between the two classes. There also appears to be substantial spatial correlation here — San Francisco, San Diego, and Phoenix all have fewer days with precipitation, for instance.

2.3 Preprocessing

After the initial cleaning, both data sets contained data for every hour. We initially intended to use all of the hourly information as covariates, but this resulted in a high-dimensional data set. Thus, we aggregated and removed variables in a way that retained the most important information in the hourly data sets. We calculated the daily minimum, average, and maximum temperature by airport across all of the hours, and we also saved the temperatures from hours 0, 12, and 23. For each airport, we also calculated daily snow and precipitation (determined by the occurrence of snow or any precipitation, respectively, at any hour), and we saved the precipitation and snow conditions at hours 0, 12, and 23.

We chose to keep the information from hours 0, 12, and 23 instead of relying solely on the daily aggregated covariates because we thought there was still some information to be gained from the hourly data. For example, a day with snow at 11pm should probably predict snow for the next day with a higher probability

than a day with snow at 1am. Similarly, the temperature just before midnight on one day is likely highly predictive of the temperature just after midnight on the next day, which may be close to the minimum for that day. Also, a day on which snow occurred at hours 0, 12, *and* 23 might indicate a large snowstorm, which could influence predictions for the following day. Similarly, a day with below-average temperatures at hours 0, 12, and 23 would indicate a general period of much colder weather.

After preprocessing, the final data set had 3,621 days of data for each of the 21 airports, for a total of 76,401 rows and 126 columns. The final set of covariates includes basic date and airport information *(year, month, day, airport code, latitude, longitude, and elevation)*, the five response variables *(minimum daily temperature, average daily temperature, maximum daily temperature, occurrence of precipitation, and occurrence of snowfall)*, weather conditions for hours 0, 12, and 23 *(occurrence of precipitation, occurrence of snow, temperature, dew point, humidity, wind speed, wind direction, and atmospheric pressure)*, and information for the three nearest airports *(the same response variables and observed weather conditions, plus distance to the center airport)*.

3 Methods

3.1 Model objectives, software, and evaluation strategy

Recall that our aim is to forecast the (1) minimum, (2) average, and (3) maximum daily temperature in the 21 cities of interest, as well as whether (4) any precipitation occurred or (5) snow occurred on a given day in each of these cities. There are several combinations of models one could consider to make predictions for these five targets using the data set constructed in section 2. For a particular target, one could build a single model that makes predictions for all 21 cities, or they could build 21 city-specific models. As mentioned in section 2.1, we view the former as a more defensible choice, as the latter would likely fail to capitalize on seasonal weather patterns that are shared across cities. Regarding temperature, one could build a single sub-daily model that makes predictions for multiple time points per day and then computes the minimum, average, and maximum of these subpredictions, or they could model the daily minimum, average, and maximum temperature separately. Similarly, one could consider a single multi-class classification model for precipitation, or they could model the two precipitation targets separately. Ultimately, we elected to train five separate models for the five targets; we theorize that these separate models may be capable of capturing target-specific trends that would not be detected by an overall temperature model or a multi-class precipitation model.

There are also numerous classes of models that could feasibly be employed to track these five targets over time. Most of these suitable models can be categorized as either classical (ARIMA-based) or modern (neural network-based) [STN18]. To decide among these, we considered several requirements: (1) Our models need to take in observations for some number of previous days and spit out predictions for the next four-day horizon; (2) they need to allow for both temporal regression (continuous response) and classification (binary response); and (3) they need to be computationally efficient and capable of handling a large number of static and time-varying covariates. Given these requirements — particularly the third one — we decided to use deep learning methods for our predictions tasks.

We explored several of the models built into the `NeuralForecast` [Oli+22] library in `Python`, which offers an extensive suite of features for training and evaluating deep learning-based forecasting methods. We found `NeuralForecast` to be especially convenient for incorporating the different types of covariates in our data set — i.e., static attributes like latitude, longitude, and elevation and time-varying weather conditions like humidity and wind speed. As will be discussed in section 3.2 and section 3.3, we ultimately employed `NeuralForecast`'s implementation of long short-term memory (`LSTM`) networks for our temperature models and neural hierarchical interpolation (`NHITS`) for our precipitation models.

Recall that our models will eventually be containerized, at which point users will be able to train them on our full data set and evaluate their predictive performance on future dates. However, before this containerization occurs, we need to determine an optimal set of hyperparameters for each model, and we would also like to ensure that these optimized models outperform a few naive baseline methods on an entirely held-out set of observations. To accomplish this, we partition our data into a training set spanning from January 1st, 2014 through November 30th, 2022 and a test set spanning from December 1st, 2022 through November 30th, 2023. We train our precipitation and temperature models on the training set, using cross-validation to tune their respective hyperparameters (see section 3.2 and section 3.3 for details). We then evaluate the tuned models by further partitioning the set of test dates into four-day

windows, making one four-day prediction for each model in each window, and summarizing the accuracy of each model's predictions.

3.2 Long short-term memory (LSTM) networks for temperature

We first examine the task of temporal regression for our three temperature targets. The response variables for these models are minimum, average, and maximum daily temperature, respectively, and we include all other variables listed in section 2.3 as covariates. This includes those that serve as the response variable in another model — e.g., minimum daily temperature, maximum daily temperature, daily occurrence of snow, and daily occurrence of any precipitation are all covariates in our model for average daily temperature.

We initially considered two of `NeuralForecast`'s network architectures for these temperature models. The first, long short-term memory (`LSTM`) networks, are a special class of recurrent neural network (`RNN`) that are known to exhibit strong predictive performance in many sequential data settings [She20]. The second, neural hierarchical interpolation (`NHITS`), is a novel architecture developed by the `NeuralForecast` authors that uses several multilayer perceptron blocks to learn and synthesize temporal structures of various resolutions — e.g., daily and seasonal trends in weather conditions [Cha+23]. The `NHITS` architecture is actually designed for long-horizon forecasting, but in preliminary training attempts on our short-horizon prediction task (`horizon = 4`), we found that its predictive accuracy was comparable to that of `LSTM`. However, for all three temperature models, we found that `LSTM` was considerably more efficient than `NHITS`. As such, we ultimately decided to only consider `LSTM` models for temperature.

For each of the three temperature models, we used the `AutoLSTM` function in `NeuralForecast` to tune the `LSTM` hyperparameters via cross-validation on the training set and, subsequently, train a model on the entire training set using the best-performing hyperparameters. Due to time and computational constraints, we considered a relatively small hyperparameter search space — it took roughly 80 minutes per model to try 20 configurations of encoder and decoder structure *(2, 3, or 4 layers with 128 or 256 nodes each)*, input size *(30, 60, or 90 days)*, and learning rate *(1e-5 to 1e-2)*. We found that validation set accuracy did not vary substantially across these configurations, and we also found that the optimal hyperparameters were nearly the same for the three different temperature models.[1]

3.3 Neural hierarchical interpolation (NHITS) for precipitation

We now turn to temporal classification — our response variables are the occurrence of snow and the occurrence of any precipitation, respectively, and we again include all other covariates listed in section 2.3. While we initially planned to implement `LSTM` models for these precipitation targets like we did for temperature, we found that the `NeuralForecast` library does not currently support temporal classification for recurrent architectures like `LSTM`. As such, we proceeded with `NHITS` for these two tasks.

We used the `AutoNHITS` function to tune and train the two precipitation models, following a similar procedure as section 3.2 but with slightly different hyperparameters since this is a different architecture. The starkest difference between the training procedures for temperature and precipitation was speed — it took roughly twice as long to try half as many configurations for the `NHITS` precipitation models compared to the `LSTM` temperature models. Nonetheless, we found that validation set accuracy was relatively stable across the attempted configurations, and the best-performing set of hyperparameters was similar for the two precipitation models.

4 Model evaluation

4.1 Performance metrics and baseline methods

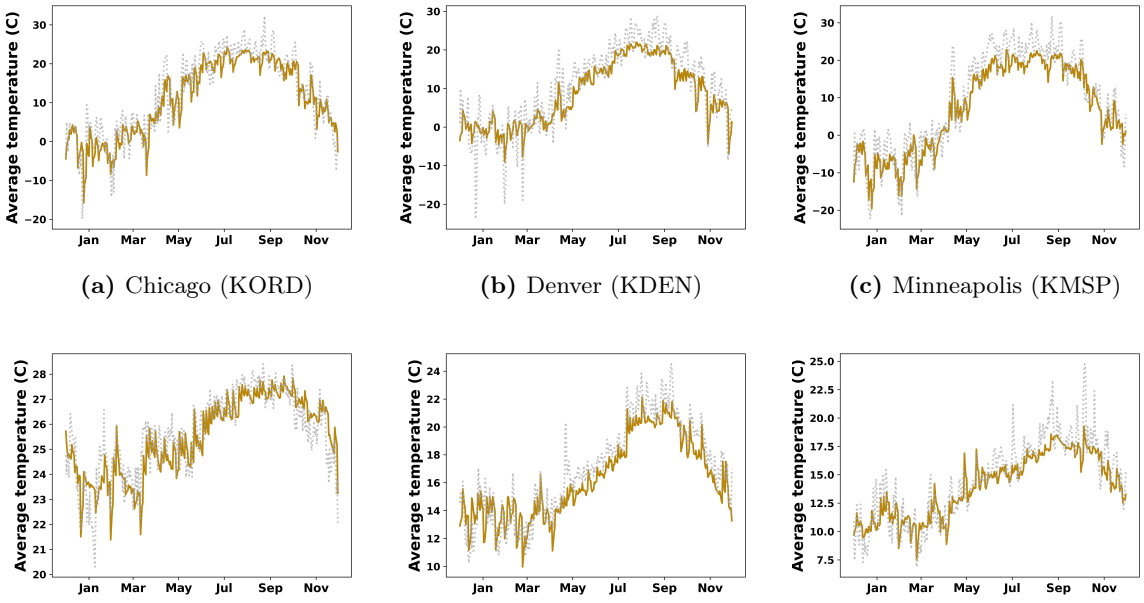
We aim to assess the predictive accuracy of the trained models from section 3.2 and section 3.3 in a manner that reflects how the containerized versions will be evaluated on future dates. To accomplish this, we evaluate the predictions made by our models on the test set, which spans from December 1st, 2022 through November 30th, 2023. We compute the root mean squared error (RMSE) and mean absolute error (MAE)

[1] In the training script of our Docker image, we felt that it was reasonable to forgo cross-validation and equip all three temperature models with the same (approximately) optimal set of hyperparameters. We made the same decision for the two precipitation models.

Table 1: Temperature prediction errors (°C) for `LSTM` vs. baselines, Dec 2022 through Nov 2023
(RMSE = root mean squared error; MAE = mean absolute error)

	Minimum temp		Average temp		Maximum temp	
	RMSE	**MAE**	**RMSE**	**MAE**	**RMSE**	**MAE**
`LSTM` (ours)	3.39	2.47	3.50	2.57	3.91	2.87
`Naive`	4.32	3.09	3.95	2.77	4.62	3.31
`SeasonalNaive`	5.54	4.08	5.32	3.92	6.13	4.57
`HistoricAverage`	8.14	6.62	8.52	6.97	9.22	7.56

Figure 3: True and predicted average temperature (°C) for selected cities, Dec 2022 through Nov 2023
(Dotted gray line indicates truth, solid gold line indicates `LSTM` predictions)



(a) Chicago (KORD) **(b)** Denver (KDEN) **(c)** Minneapolis (KMSP)

(d) Honolulu (PHNL) **(e)** San Diego (KSAN) **(f)** San Francisco (KSFO)

of our `LSTM` models' daily temperature predictions, and we compute the classification accuracy, area under the ROC curve (AUC), and Brier score [Ruf10] of our `NHITS` models' daily precipitation predictions.

We also contextualize the performance of our models by comparing them to three baselines: (1) `Naive`, which uses the previous day's weather as its prediction for the current day's weather in each city; (2) `SeasonalNaive`, which uses the weather from one year ago as its prediction for the current day's weather in each city; and (3) `HistoricAverage`, which uses a moving average of the training and test data as its prediction for the current day's weather in each city. These baseline methods are built into `StatsForecast` [Gar+22], which is a companion library of `NeuralForecast`.

4.2 Results for temperature

In Table 1, we report the RMSE and MAE of our `LSTM` models and the three baselines for all three temperature prediction tasks. We find that our models are more accurate than the three baselines on all three tasks, achieving an average error of around three degrees Celsius. The `Naive` method is the most accurate baseline, followed by `SeasonalNaive` and then `HistoricAverage`.

Figure 3 validates that our `LSTM` models produced reasonable predictions — it displays the true and predicted average daily temperature across the test set for six cities, the three with the largest RMSE for this task (Chicago, Denver, and Minneapolis) and the three with the smallest RMSE (Honolulu, San Diego, and San Francisco). For all six cities, we find that our model captures both general and more precise trends in daily average temperature.

Table 2: Temperature (`LSTM`) and precipitation (`NHITS`) predictions by city, Dec 2022 through Nov 2023
($\overline{\widehat{y}}$ = mean of predicted daily values; $\widehat{p}$ = proportion of days with precip/snow predicted)

	Min temp		Avg temp		Max temp		Any precip		Snow	
	$\overline{\widehat{y}}$	RMSE	$\overline{\widehat{y}}$	RMSE	$\overline{\widehat{y}}$	RMSE	$\widehat{p}$	Accuracy	$\widehat{p}$	Accuracy
Ann Arbor (KARB)	3.7	4.5	8.6	4.0	14.3	4.5	0.49	0.58	0.16	0.86
Anchorage (PANC)	0.5	3.2	2.3	2.9	5.7	3.1	0.53	0.59	0.22	0.78
Boise (KBOI)	6.6	3.4	10.5	4.1	17.5	4.4	0.34	0.64	0.12	0.87
Chicago (KORD)	7.5	4.1	10.7	4.3	15.8	4.7	0.52	0.56	0.12	0.86
Denver (KDEN)	3.2	4.3	8.4	4.9	18.1	5.5	0.36	0.66	0.15	0.85
Detroit (KDTW)	7.0	3.7	10.9	3.8	15.8	4.2	0.51	0.57	0.14	0.83
Honolulu (PHNL)	22.9	1.4	25.4	0.9	29.2	1.0	0.53	0.61	0.00	1.00
Houston (KIAH)	17.2	4.0	21.0	3.8	27.6	4.1	0.43	0.62	0.00	1.00
Miami (KMIA)	23.3	2.0	25.8	1.7	29.5	1.8	0.49	0.62	0.00	1.00
Minneapolis (KMSP)	4.3	4.4	7.1	4.8	12.7	4.5	0.47	0.50	0.20	0.79
Oklahoma City (KOKC)	10.2	4.1	16.9	4.3	22.7	4.7	0.33	0.66	0.05	0.97
Nashville (KBNA)	11.9	4.2	16.9	4.0	22.2	4.4	0.46	0.52	0.03	0.97
New York (KJFK)	10.1	3.0	12.6	3.1	16.5	3.9	0.44	0.52	0.04	0.95
Phoenix (KPHX)	18.6	2.7	22.9	3.4	30.4	3.1	0.18	0.86	0.01	1.00
Portland (ME) (KPWM)	5.9	3.9	8.2	3.9	13.0	4.5	0.47	0.52	0.12	0.86
Portland (OR) (KPDX)	9.4	2.8	12.4	2.8	18.4	3.6	0.43	0.69	0.03	0.96
Salt Lake City (KSLC)	7.3	3.5	10.5	4.2	17.2	4.7	0.35	0.63	0.14	0.82
San Diego (KSAN)	14.1	2.0	16.2	1.4	19.8	2.3	0.22	0.75	0.00	1.00
San Francisco (KSFO)	11.5	1.8	14.0	1.7	18.1	2.6	0.21	0.74	0.00	1.00
Seattle (KSEA)	8.4	2.4	11.6	2.5	16.1	3.3	0.47	0.70	0.03	0.95
Washington DC (KDCA)	12.0	3.1	14.9	3.3	20.0	4.1	0.41	0.55	0.02	0.98

We find that the predictive accuracy of the three `LSTM` models is fairly consistent across the 21 cities of interest, although there is some variation. See Table 2, which reports the RMSE by city for each of the three `LSTM` models; we observe that the city-specific RMSE ranges between roughly one and five degrees Celsius for all three tasks. Table 2 also reports the means of the predicted minimum, average, and maximum daily temperatures, respectively, across the test set for each city. These columns demonstrate that our `LSTM` models produced predictions that are compatible with our prior knowledge of the temperatures of U.S. cities — they are consistent with the raw data trends in Figure 1, for instance.

4.3 Results for precipitation

We find that our `NHITS` models for predicting the occurrence of snow and any precipitation, respectively, are more accurate than the three baseline methods. Table 3 indicates that `NHITS` achieved a higher accuracy, higher AUC, and lower Brier score than the baselines on both tasks, and Figure 4 corroborates this by displaying the ROC curves for `NHITS` and the baselines for each task. Notably, all four methods exhibit a strong performance in snow prediction. However, we theorize that this may be due to class imbalance in the binary response, as snow occurred on a small proportion of days in the test set across all 21 airports. In contrast, the four methods perform relatively poorly when it comes to predicting general precipitation. We theorize that this is likely due to the inherent challenge of predicting multiple types of precipitation simultaneously — the atmospheric conditions that tend to accompany various types of precipitation are likely different and potentially conflicting.

Nevertheless, our `NHITS` models produce reasonable predictions. Table 2 reports the classification accuracy of each model by city, as well as the proportion of days in the test set on which snow and any precipitation are predicted to occur. For both tasks, we observe that classification accuracy is generally higher for cities that experience less precipitation (e.g., Phoenix) and lower for cities that experience more precipitation (e.g., Anchorage). We also find that our models correctly predict higher proportions of rain and snow in notoriously rainy (e.g., Seattle) or snowy (e.g., Minneapolis) cities, respectively.

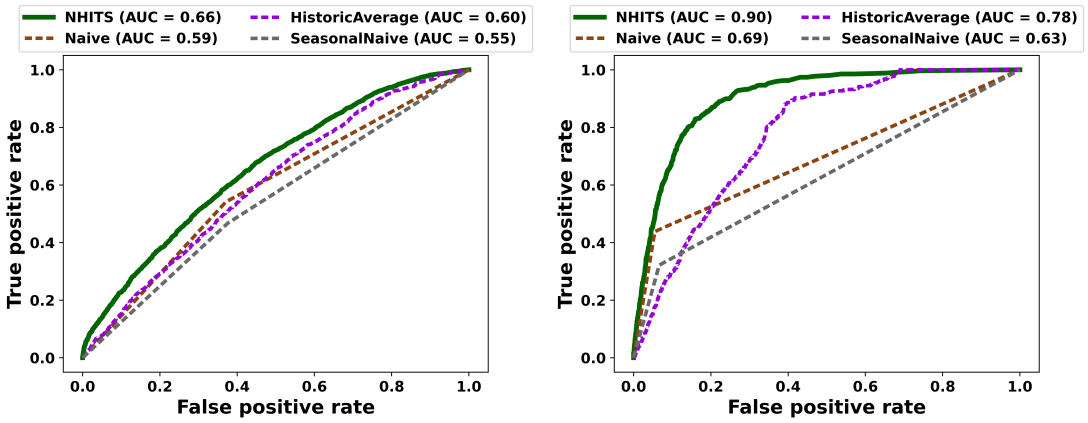
5 Reproducibility

Docker is a powerful platform for developing, shipping, and running applications that uses containerization techniques to ensure consistent deployment behavior [Mer14]. An image is a standalone package with

Table 3: Precipitation prediction metrics for `NHITS` vs. baselines, Dec 2022 through Nov 2023
(Accuracy = proportion of days correct; AUC = area under ROC curve; Brier = Brier score)

	Any precipitation			Snow		
	Accuracy	**AUC**	**Brier**	**Accuracy**	**AUC**	**Brier**
`NHITS` (ours)	0.62	0.66	0.23	0.92	0.90	0.06
`Naive`	0.59	0.59	0.41	0.90	0.69	0.10
`SeasonalNaive`	0.56	0.55	0.44	0.88	0.63	0.12
`HistoricAverage`	0.58	0.60	0.24	0.92	0.78	0.07

Figure 4: ROC curves for precipitation models, `NHITS` vs. baselines



(a) Any precipitation **(b)** Snow

everything needed to run a piece of software, such as libraries, code, or configuration files. A container is an instance running the image — i.e., a live version of it that behaves identically regardless of the environment in which it is deployed. Using Docker ensures the reproducibility of our results across devices. In this project, we tackled the nuances of writing efficient, multi-platform Docker images that are consistent across time zones and that support AMD64 and ARM architectures.[2]

Specifically, our Docker image relies on the latest Ubuntu image as a foundation, which facilitates simultaneous support for both `Python` and `R`, two critical pieces of our computational workflow. To ensure support for these applications, we installed various development libraries to produce a runtime environment crafted to address the challenges of each stage of our analysis, ranging from data acquisition, cleaning, and updating to training and prediction.

For the purpose of training and prediction, in particular, the following commands are available within our Docker container:

`docker run gapatronstats604_project04 train`

> This command initiates the data acquisition process and ultimately generates a file called `data_final.csv` that contains clean METAR and Meteostat data from January 1st, 2014 through November 30th, 2023, as reported in section 2. It then trains all of our models, as per section 3, and saves their output. The chosen parameters are based on the outcomes of our temporal cross-validation approach, the details of which are available in the `report` folder of our GitHub repository.[3]

`docker run gapatronstats604_project04 predict`

> To smoothly transition from model training to prediction, this command is responsible for updating our data sets by calling the updating scripts for both `METAR` and `Meteostat`, which pull data from December 1st, 2023 to the current day (in UTC). The original data is obtained from the saved models, and hence this `predict` command can be run

[2] Our image is available on Docker Hub at https://hub.docker.com/r/gapatron/stats604_project04.
[3] Our GitHub repository can be accessed at https://github.com/gapatron/stats604_project04.

independently of `train`. The predictions are then generated using the saved model weights and output to the console in `JSON` format. This command can be run with one of two tags `:arm` or `:amd64`, defaulting to `:arm`. The `:arm` tag should be used if the image is being run on an Apple M1/M2 architecture, while the `:amd64` tag is necessary for most Windows and Linux systems. Docker containers are supposed to run irrespective of a device's environment; however, we discovered that differences between the ARM and AMD64 architectures are notable exceptions to this.

These commands facilitate users' interaction with the image, allowing them to focus only on the principal aspects of their request to train or predict without thinking about the complicated pieces that make up such a pipeline. Naturally, users can also access the container interactively if they want to explore the different steps involved in greater detail. In summary, our Docker image is user-friendly, robust and versatile, and it constructs a well-crafted ecosystem that seamlessly supports our entire workflow.

6 Discussion

Throughout this project, we made many decisions regarding data acquisition, preprocessing steps, model specification, and hyperparameter tuning. First, our choice of data sources and data preprocessing methods tended to prioritize smaller data sets, and therefore faster models. In addition to the hours dropped from both Meteostat and METAR, we also ignored any non-snow weather condition codes present in the METAR data. These condition codes indicate the presence of a wide range of weather phenomena, including rain, mist, fog, and thunderstorms, among others. They also indicate the intensity of each condition, which can be light, moderate, or heavy. Intensity, in particular, might have been a useful covariate in our models.

Additionally, we noticed that the report time returned by the `pmetar` package did not always match the report time generated by the `Metar` package. The report time from the `pmetar` package matched the time contained in the string itself, but the observation for an identical raw string in the `Metar` package often had a slightly different report time. This report time often appeared to be a rounded version of the `pmetar` version. As a result of this discrepancy, identical reports were often assigned to different days, depending on which package was used. This has the potential to impact the results, particularly if the report in question contained the minimum temperature or different precipitation values compared to the other hours of the day.

Making these data processing decisions was quite tedious and limited the amount of time we had to develop our model training and prediction workflow. Despite the fact that each of our five models used the other four response variables as covariates, their predictions are not guaranteed to be logical when evaluated as an ensemble. For example, our model for snow could predict `false` on the same day that our model for precipitation predicts `true` and our model for maximum temperature predicts a negative value. It would have been interesting to test the other combinations of models discussed in subsection 3.1, such as creating one model for all temperature predictions. Additionally, we narrowed our focus to deep learning methods in the early stages of this project, and we did not seriously consider classical ARIMA-based models or any other feasible model classes.

Despite these limitations, our models exhibit relatively strong predictive performance on our test set, and we believe that this performance is likely to carry over when they are tested on future dates. Unlike traditional forecasts based on numerical weather prediction, our models do not require running massive simulations of physical systems on a supercomputer; in fact, they require essentially no background knowledge of meteorology. They can be trained and utilized for prediction in an efficient manner via our Docker image, and all of the code and analysis underlying them is publicly accessible on GitHub. Given that we were able to develop this transparent, reproducible, and relatively successful set of deep learning-based forecasting models in just under one month, it is perhaps not surprising that the domain of weather prediction has seen such a rapid shift toward these methods in recent years.

References

[Cha+23] Cristian Challu et al. "NHITS: Neural hierarchical interpolation for time series forecasting". In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 37. 6. 2023, pp. 6989–6997.

[Cwi23] Pawel Cwiek. *pmetar: Processing METAR Weather Reports*. R package version 0.5.0. 2023. URL: https://CRAN.R-project.org/package=pmetar.

[Gar+22] Federico Garza et al. *StatsForecast: Lightning fast forecasting with statistical and econometric models*. PyCon Salt Lake City, Utah, US 2022. 2022. URL: https://github.com/Nixtla/statsforecast.

[Kul03] Gloria Kulesa. "Weather and aviation: How does weather affect the safety and operations of airports and aviation, and how does FAA work to manage weather-related effects?" In: *The Potential Impacts of Climate Change on Transportation*. 2003.

[Lam+23] Remi Lam et al. "Learning skillful medium-range global weather forecasting". In: *Science* (2023), eadi2336.

[Lam23] Christian Sebastian Lamprecht. *Meteostat Python*. Python package version 1.6.7. 2023.

[Mer14] Dirk Merkel. "Docker: lightweight linux containers for consistent development and deployment". In: *Linux journal* 2014.239 (2014), p. 2.

[Met23] Meteostat. *Meteostat: The weather's record keeper*. 2023. URL: https://meteostat.net/en/.

[Oli+22] Kin G. Olivares et al. *NeuralForecast: User friendly state-of-the-art neural forecasting models*. PyCon Salt Lake City, Utah, US 2022. 2022. URL: https://github.com/Nixtla/neuralforecast.

[PK19] Zhaoxia Pu and Eugenia Kalnay. "Numerical weather prediction basics: Models, numerical methods, and data assimilation". In: *Handbook of Hydrometeorological Ensemble Forecasting* (2019), pp. 67–97.

[Ruf10] Kaspar Rufibach. "Use of Brier score to assess binary predictions". In: *Journal of Clinical Epidemiology* 63.8 (2010), pp. 938–939.

[Sch+21] Martin G Schultz et al. "Can deep learning beat numerical weather prediction?" In: *Philosophical Transactions of the Royal Society A* 379.2194 (2021), p. 20200097.

[She20] Alex Sherstinsky. "Fundamentals of recurrent neural network (RNN) and long short-term memory (LSTM) network". In: *Physica D: Nonlinear Phenomena* 404 (2020), p. 132306.

[STN18] Sima Siami-Namini, Neda Tavakoli, and Akbar Siami Namin. "A comparison of ARIMA and LSTM in forecasting time series". In: *2018 17th IEEE International Conference on Machine Learning and Applications (ICMLA)*. IEEE. 2018, pp. 1394–1401.

[Wil+18] Emily Wilkins et al. "Effects of weather conditions on tourism spending: implications for future trends under climate change". In: *Journal of Travel Research* 57.8 (2018), pp. 1042–1053.