Emily Moeller

12/13/2019

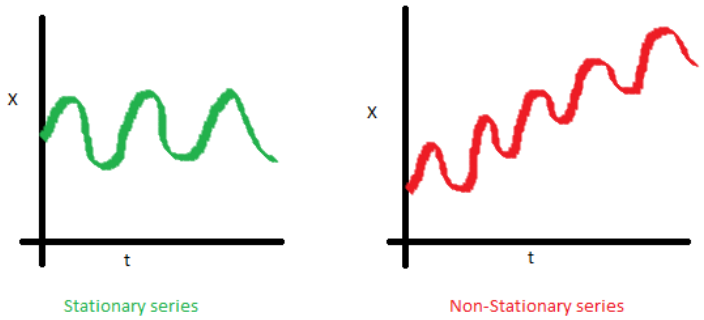
Using Time Series Algorithms for Predicting Global Land Temperatures for Major Cities

Introduction

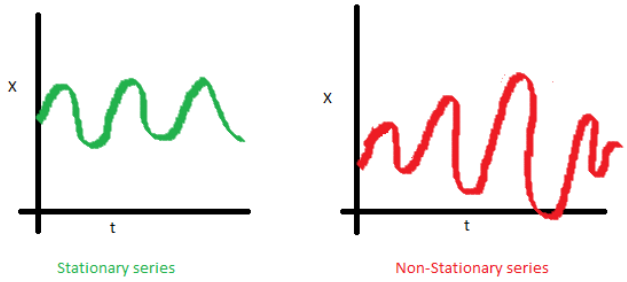
The Earth's surface temperature record is one of the many pieces of evidence indicating that Earth is being affected by climate change. Researchers at Berkeley Earth, which is affiliated with Lawrence Berkeley National Laboratory, have proved this fact by studying the Earth's temperature while dispelling the concerns that global warming skeptics had identified [1]. They assembled a variety of datasets that track the global land temperatures of cities, states and countries. Understanding the trend of global land temperatures is important in gaining greater insight of global warming as a whole and being able to adequately prepare for the way it will impact human life. For this reason, I will be using the data on global land temperatures in 100 major cities to develop models to predict the average temperatures for future dates. To accomplish this task, I first need to gain a background on the basic principles of time series algorithms. In particular, I learned about the different techniques that are used to satisfy the stationarity condition. After learning about the requirements for a time series algorithm, I reviewed a variety of time series algorithms such a Simple Moving Average, Exponential Moving Average, Auto-Regressive Integrated Moving Averages, and Seasonal Autoregressive Integrated Moving-Average (SARIMA). Because order is an important feature in time series algorithms, it is also impossible to shuffle the data and perform typical cross-validation techniques. As a result, I examined two different strategies to solve this problem. After all this research, I was able to apply what I learned to the original task of predicting land temperatures for 100 major cities around the globe. First, I give an overview of what the data looks like and what preprocessing steps are involved before applying a time series algorithm. After understanding the qualities of the data, I was able to choose the time series algorithm that I thought would work best. I discuss my implementation of this algorithm and evaluate my results, while reflecting on areas of improvement.

Overview of time series algorithms

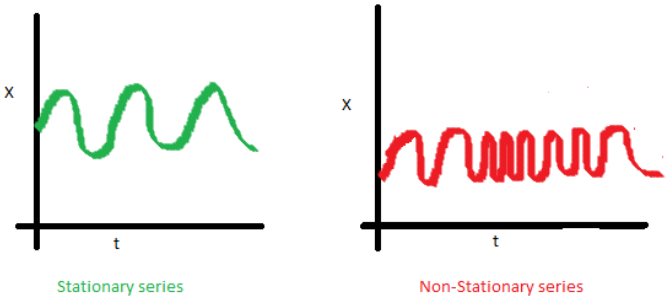
Before discussing the approach I took to solve the problem outlined above, it is necessary to present necessary background information on time series algorithms and to give an overview of some of the popular algorithms that are used. To build a time series model, the data must be classified as a stationary series. For a dataset to be a stationary series, there are three conditions that must hold. These conditions are that the mean, variance, and covariance cannot be a function of time, meaning they must be constant throughout the entire series. The following graphs give an example of what type of data is allowed and what is not allowed [8].



The red graph above depicts data where the mean is a function of time.



The red graph above depicts data where the variance is a function of time.



The red graph above depicts data where the covariance is a function of time.

However, in many cases of real-world datasets, especially in the case of a dataset about global land temperatures in major cities, we cannot expect the stationarity property to hold [8]. When this condition is not satisfied, the first task then becomes to make the data stationary and then use stochastic models to predict this time series.

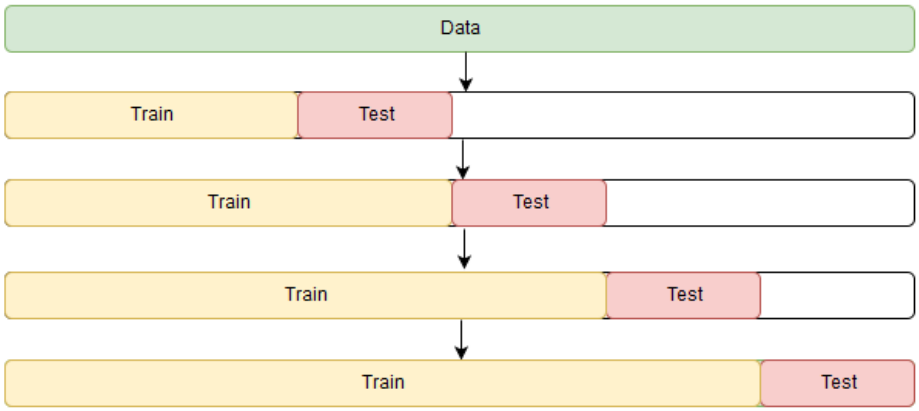
There are a variety of techniques to translate data into a stationary time series and the method that you choose depends of the patterns in your dataset. If the dataset simply contains a trend that needs to be eliminated, three possible techniques are aggregation, smoothing, and polynomial fitting. Aggregation involves taking the average over a period like weeks or months, smoothing means taking rolling averages, and polynomial fitting means fitting a regression model. However, if your dataset contains a trend and is seasonal, then you will have to use different techniques, such as differencing and decomposition. Differencing means taking the difference with some time lag, and decomposition means

that you model both trend and seasonality to remove them from the model itself. It is also worth noting that some algorithms that I'll be describing shortly have these processes built into their implementation, so for the purposes of this project I won't dive deep into their implementations. However, it is still important to understand which properties apply to a dataset because it can help in choosing the proper time series algorithm.

This first and most simple time series algorithm is the Simple Moving Average (SMA). This algorithm models the next step in the sequence as a linear function of the residual errors from a mean process at prior time steps [9]. A moving average model takes advantage of the average of the data points that exist in a specific overlapping subsection of the series. An average is computed from the first subset of the data, and then it is moved forward to the next data point while dropping out the first data point [11]. A SMA is often a preprocessing step before forecasting because it can help identify trends in the data. This method works well for a univariate time series without trend and seasonal components [9]. The Exponential Moving Average (EMA) differs from the SMA because it uses all data points before the current data point. Weights are assigned to each data point, in an exponentially decreasing fashion so that the points closer to the new predicted value are given more significance than the old values [11].

The Autoregressive Integrated Moving Average (ARIMA) is a univariate linear function that is used for predicting future data points based on past data. ARIMA forecasting is a linear regression equation that depends on three variables which are the number of AR (Auto-Regressive) terms, the number of MA (Moving Average) terms, and the number of nonseasonal differences. The AR terms are the lags of the dependent variable and the MA terms are the lagged forcasted errors in the prediction equation [13]. Because of ARIMA's reliance on it's own past data, a longer series is preferable to get more accurate results. This algorithm is appropriate for univariate time series with trend and without seasonal components. Seasonal Autoregressive Integrated Moving-Average (SARIMA) acts an extension of ARIMA by allowing a seasonal component in addition to a trend component [12]. This is enabled by selecting analogous parameters for both the trend and seasonal elements of the series.

Cross validation also has a different approach for time series algorithms. In the case of the absence of time, data can be randomly selected to evaluate the accuracy of the measurement, but when time is involved the validation data always must occur after the training data because we are predicting a value for the future [2]. There are two schemas to accomplish this: sliding-window and Forward Chaining validation methods [6]. In the sliding-window method, we establish a fixed window of size n for each the training set and test set. Then we simply train on the first set of n records and validate on the next set of n records. The training set and test set windows are then incremented by n until we reach the end of the dataset [6]. For the forward chaining method, we train on the first t records of data and we test from t to t + n records. On the next iteration, we train on the first t + n data points and predict on the data points from t + n to t + 2*n. This process continues until we run out of data [10].



This diagram above depicts the forward chaining cross validation for 4 folds, and the diagram below compares the sliding window method and the forward chaining method.

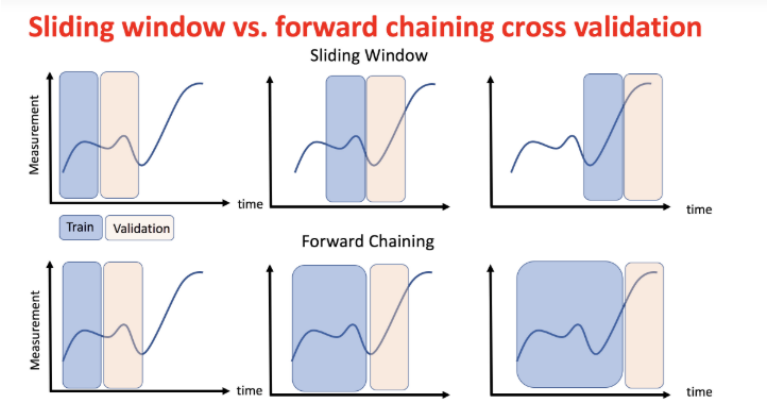


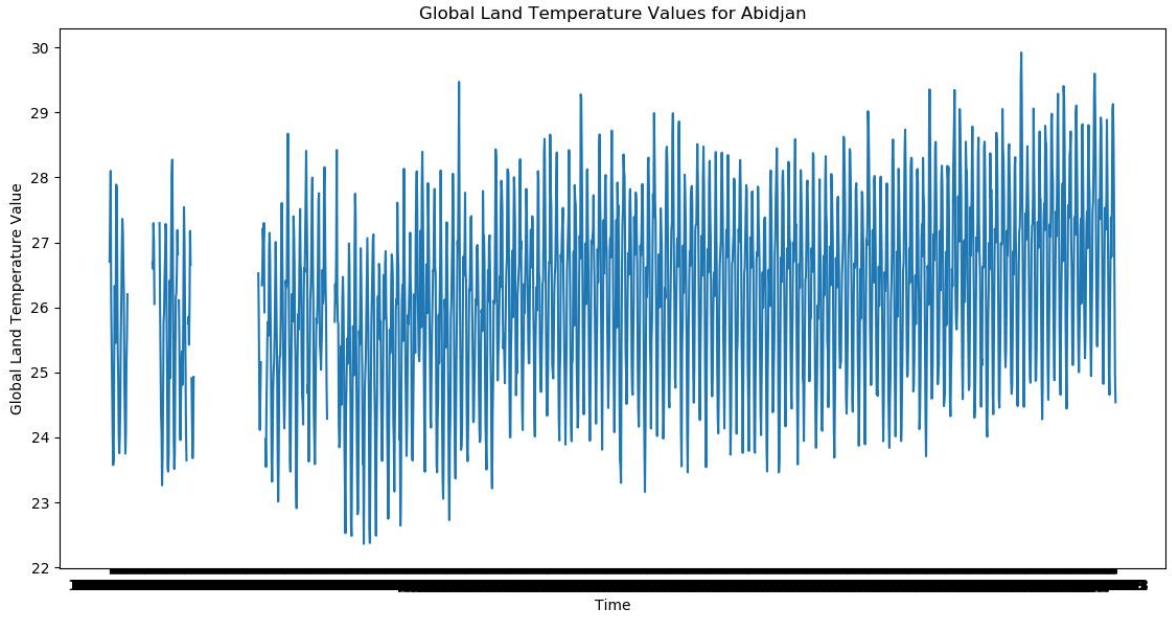
Fig. 5) Basically, there are two kinds of cross-validation for the time series sliding window and forward chaining. In this post, we will consider forward chaining cross-validation method

Approach and Methods

Using the knowledge gained from researching time series algorithms, it's important to examine the properties of the data. The dataset contains temperature data for 100 different cities for every month dating back to the 1800s. Here is a snippet of what some of the data looks like for the first city of Abidjan [4].

	A	B	C	D	E	F	G
1	dt	AverageTemperature	AverageTemperatureUncertainty	City	Country	Latitude	Longitude
2	1849-01-01	26.704	1.435	Abidjan	Côte d'Ivoire	5.63N	3.23W
3	1849-02-01	27.434	1.362	Abidjan	Côte d'Ivoire	5.63N	3.23W
4	1849-03-01	28.101	1.612	Abidjan	Côte d'Ivoire	5.63N	3.23W
5	1849-04-01	26.14	1.387	Abidjan	Côte d'Ivoire	5.63N	3.23W
6	1849-05-01	25.427	1.2	Abidjan	Côte d'Ivoire	5.63N	3.23W
7	1849-06-01	24.844	1.402	Abidjan	Côte d'Ivoire	5.63N	3.23W
8	1849-07-01	24.058	1.254	Abidjan	Côte d'Ivoire	5.63N	3.23W
9	1849-08-01	23.576	1.265	Abidjan	Côte d'Ivoire	5.63N	3.23W
10	1849-09-01	23.662	1.226	Abidjan	Côte d'Ivoire	5.63N	3.23W
11	1849-10-01	25.263	1.175	Abidjan	Côte d'Ivoire	5.63N	3.23W
12	1849-11-01	26.332	1.507	Abidjan	Côte d'Ivoire	5.63N	3.23W
13	1849-12-01	25.45	1.838	Abidjan	Côte d'Ivoire	5.63N	3.23W
14	1850-01-01	25.803	1.943	Abidjan	Côte d'Ivoire	5.63N	3.23W
15	1850-02-01	27.89	1.43	Abidjan	Côte d'Ivoire	5.63N	3.23W
16	1850-03-01	27.852	2.173	Abidjan	Côte d'Ivoire	5.63N	3.23W
17	1850-04-01	26.547	1.662	Abidjan	Côte d'Ivoire	5.63N	3.23W
18	1850-05-01	25.379	1.355	Abidjan	Côte d'Ivoire	5.63N	3.23W
19	1850-06-01	24.903	1.178	Abidjan	Côte d'Ivoire	5.63N	3.23W
20	1850-07-01	24.04	1.301	Abidjan	Côte d'Ivoire	5.63N	3.23W

As shown in the data above, the longitude and latitudes are simply represented with the degree value and the compass direction. To make these values more compatible with machine learning models the latitude values were converted from their 0 to 90 range with directional components to a -90 to 90 range, by simply making the south degrees negative. The same thing was done for west degree values for longitudes, which range from -180 to 80 [7].



The graph above shows the land temperatures in the city Abidjan in Côte d'Ivoire each month from 1849 to 2013. We can see from this graph that there is an upward trend in the values of the global land temperatures over time. In addition, we must take into account that the data will have seasonal

components, because we would expect the temperatures to be similar in the same parts of a year, over the course of many years. With that in mind, this led me to the choice of the Seasonal Autoregressive Integrated Moving-Average (SARIMA) algorithm to solve this problem because it is suitable for data with trend and seasonal component in a univariate time series [9]. My plan was to create a model for each city by testing different parameters to model and choosing the model that results in the lowest error rate.

Implementation

The first step of the implementation for the program was preprocessing the data so that it is in the correct format for other parts of the program. As shown in the snippet of code in the previous section, the longitude and latitudes are represented in the dataset with the degree value and the compass direction. To make these values more compatible for programming, I converted the latitude values from their 0 to 90 range with directional components to a -90 to 90 range by simply making the south degrees negative. The same thing was done for west degree values for longitudes, which range from -180 to 80 [7]. Once the latitudes and longitudes are in the correct format, the dates, average land temperature per month, and average temperature uncertainty are extracted for each city. Next, this data is split 70 – 30, where 70% is used for training and 30% is used for testing. I decided not to use the cross-validation techniques that I described in the first section because the runtime was already quite slow.

For the training of the model, I decided to use the statsmodels library, which has a good selection of time series algorithms. In particular, I choose to use the SARIMAX model [5]. The first step is creating the model. To have the model account for both trend and seasonality in the data, it's necessary to pass in three different parameters. The first is simply the observed time-series process y, which is the average temperatures in my case. The next parameter is order, which takes a list of three values represented by p, d and q. The p is the trend autoregression order, the d is the trend difference order, and the q is the trend moving average order [12]. The final parameter is seasonal_order which also takes a list of P, D, Q values and an additional m value [12]. The P, D, Q values are just the seasonal versions of the previous values and the m value represents the number of time steps for a single seasonal period. Configuring these parameters can be done with advance techniques, like using graphs of the Autocorrelation Function (ACF) and Partial Autocorrelation Function (PACF) to determine initial values for p and q, respectively, and then doing a grid search to find the best model. However, due to time constraints, I ended up setting all the parameters to 1 to just make use of the previous trend and seasonal values.

The second step is fitting the model, in which I only pass in one parameter that is simply for ignoring convergence messages. The third and final step is for predicting values, so I supplied the start and the end of how many values I would like to predict. The start value is one time stamp past where the training set stopped, and the end is the length of the test set minus one after the starting spot [5]. The following snippet of code is one example of how to execute the previous three steps.

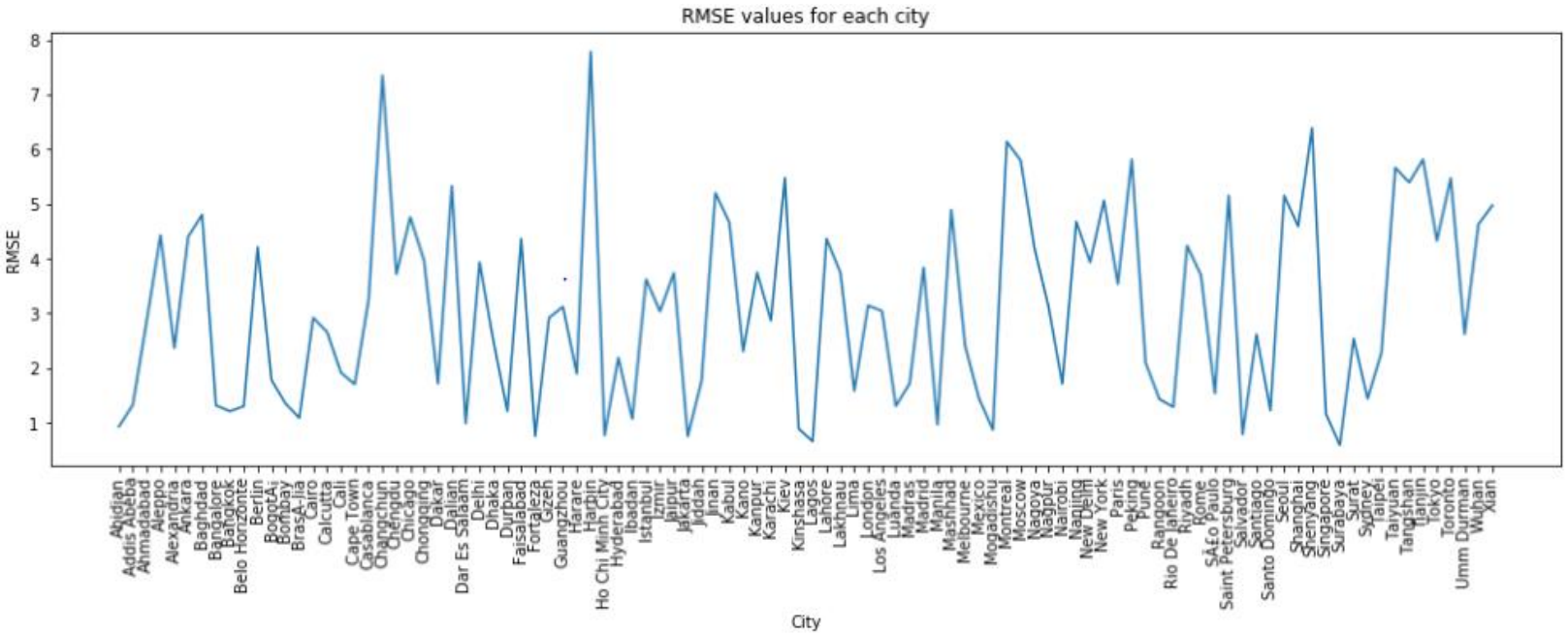
```
#fit model for my data
model = SARIMAX(ytrain, order=(0, 0, 1), seasonal_order=(0, 1, 1, 12))
model_fit = model.fit(disp=False)

#Predict temp values
start = len(xtrain) + 1
end = start + len(xtest) - 1
yPredict = model_fit.predict(start=start, end=end, dynamic=False)
```

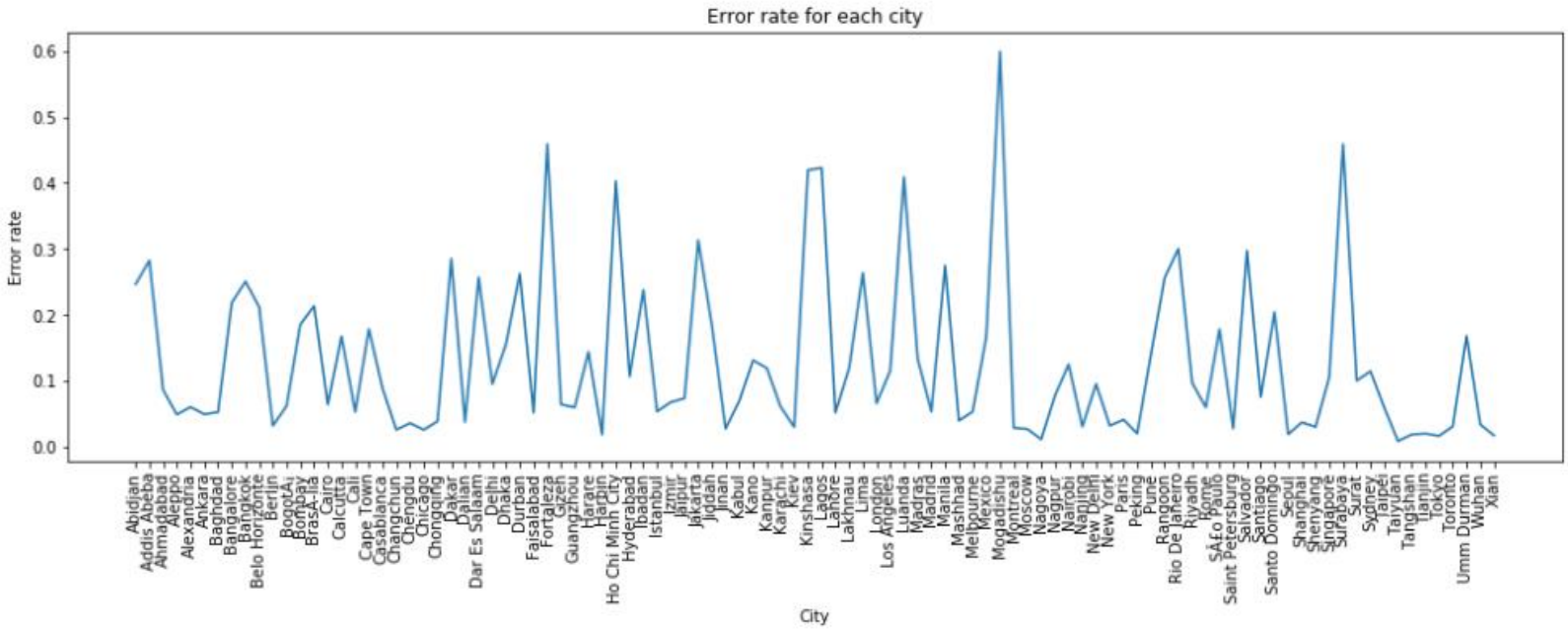
One challenge that I ran into during my implementation was deciding how to handle when there was an absence of recorded temperature data. My thinking was that I did not want to remove this data before the model had been trained because it would alter the seasonal nature of the model by not having all 12 months of a year's data. Also, for the records that didn't have a temperature, those values were input as a nan, so they didn't play a role in altering future predictions. Considering all of that, I opted to remove them after all values were predicted and before I calculated the RMSE values and error rates.

Evaluate Results

In terms of evaluating my results, I choose to use root mean-squared error because it is a very common metric used to evaluate machine learning algorithms and it is a metric that I was already familiar with implementing. I also calculated the error rate by using the average temperature uncertainty value from the dataset with the true average temperature to see if the predicted value fell with that range.



The graph above shows the RMSE values for each city. The average RMSE value was 3.06526



The graph above shows the error rate for each city. The average error rate was 0.129991, or 12.99%.

Conclusion

After doing research on times series algorithms, I was able to build a model that could predict the average land temperatures for 100 major cities around the world with a reasonable accuracy. Of course, there are always improvements that could be made. For example, I would have liked to implement a grid search to find the best parameters for SARIMAX model. I would have also liked to implement both the forward chaining and sliding window methods for cross validation and see how they compared against each other. It may have also been beneficial to try other time series algorithms and see how they compare against SARIMA. There were also other datasets available from the same source that contained global land temperatures for states, countries, and even more cities so it would have been interesting to work with those datasets as well or potentially in conjunction with each other.

References

[1] http://berkeleyearth.org/about/

[2] https://machinelearningmastery.com/time-series-forecasting-supervised-learning/

[3] https://scikit-learn.org/stable/modules/linear_model.html#ordinary-least-squares

[4] http://berkeleyearth.lbl.gov/city-list/

[5] http://www.statsmodels.org/dev/generated/statsmodels.tsa.statespace.sarimax.SARIMAX.html

[6] https://towardsdatascience.com/time-series-machine-learning-regression-framework-9ea33929009a

[7] https://knowledge.domo.com/Visualize/Adding_Cards_to_Domo/KPI_Cards/Building_Each_Chart_Type/Latitude-Longitude_Map

[8] https://www.analyticsvidhya.com/blog/2015/12/complete-tutorial-time-series-modeling/

[9] https://machinelearningmastery.com/time-series-forecasting-methods-in-python-cheat-sheet/

[10] https://medium.com/open-machine-learning-course/open-machine-learning-course-topic-9-time-series-analysis-in-python-a270cb05e0b3

[11] https://algorithmia.com/blog/introduction-to-time-series

[12] https://machinelearningmastery.com/sarima-for-time-series-forecasting-in-python/

[13] https://www.analyticsvidhya.com/blog/2016/02/time-series-forecasting-codes-python/