

Weather Analysis

Aaron Abromowitz, Simi Augustine, David Camacho, Ivan Chavez

Introduction	1
Project Overview	1
Database	2
Dashboard	6
Exploratory Data Analysis	9
Predictive Models	12
Conclusion	17

Introduction

Information about the weather has always been useful for people to know. Knowing the temperature, humidity, or chance of rain, can assist people with planning their scheduling or choosing what clothes to wear. These factors can be based on the time of year or recent weather trends. In addition, it is even more important to understand weather trends over time with current proposed rates of climate change.

Project Overview

Our project has three components: a database of weather data, a dashboard to visualize the data, and models to predict future weather.

The original data came from [Kaggle](#). We put it into 7 tables in a MySQL database: city_attributes, temperature, pressure, humidity, wind_speed, wind_direction, and weather_description. This allowed us to easily visualize and analyze the data.

We made a Dashboard in Tableau to help us visualize trends in the data. This was useful in seeing the seasonality in the data, how that changes per city, and how this seasonality changed for each variable. In addition, it allowed us to visualize model assumptions so that we could make smarter decisions about the model as well.

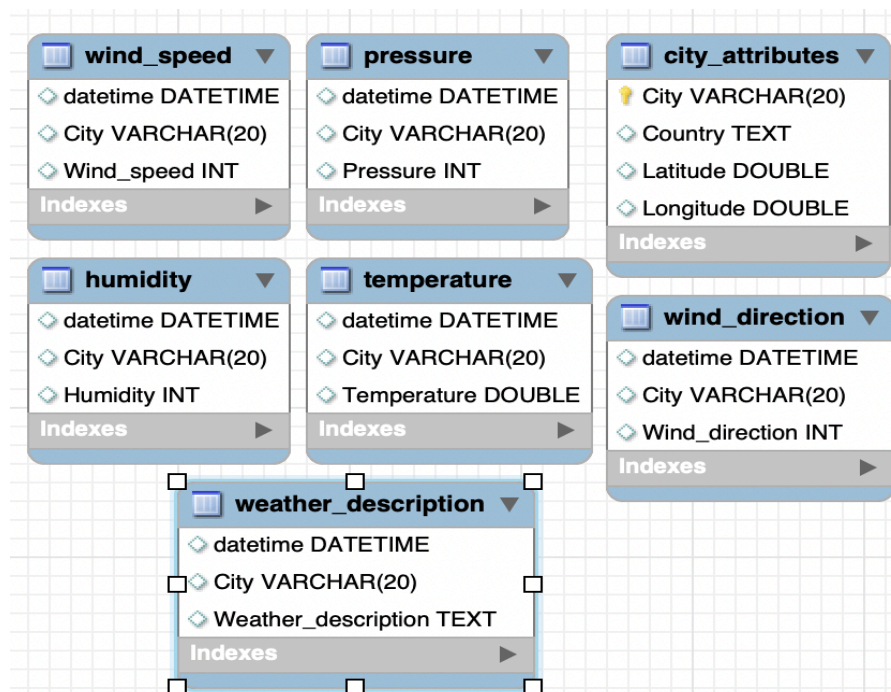
Lastly, we wanted to use modeling techniques to see if we could predict future data. We tested four different models that used the seasonality in weather data: Basic Linear Model, Holt Seasonal Model, SARIMA Model, and Vector Autoregression (VAR) Model. We held out data from 2017 to use as new data for validation.

Database

The data from Kaggle was in the form of seven CSV files: information about each city, humidity, pressure, temperature, verbal description of the weather, wind direction, and wind speed at each hour of the day between 10/01/2012 and 11/30/2017. Each of our CSV's had the cities in column format as can be seen below.

	datetime	Vancouver	Portland	San Francisco	Seattle	Los Angeles	San Diego	Las Vegas	Phoenix	Albuquerque	Denver	San Antonio	Dallas	Houston	Kansas City	Minneapolis
1	2012-10-01 12:00:00															
2	2012-10-01 13:00:00	0.0	0.0	150.0	0.0	0.0	0.0	0.0	10.0	360.0	20.0	0.0	340.0	270.0	0.0	
3	2012-10-01 14:00:00	6.0	4.0	147.0	2.0	0.0	0.0	8.0	9.0	360.0	22.0	5.0	339.0	268.0	6.0	
4	2012-10-01 15:00:00	20.0	18.0	141.0	10.0	0.0	0.0	23.0	9.0	360.0	31.0	18.0	338.0	265.0	20.0	
5	2012-10-01 16:00:00	34.0	31.0	135.0	17.0	0.0	0.0	37.0	9.0	360.0	39.0	31.0	337.0	263.0	34.0	
6	2012-10-01 17:00:00	47.0	44.0	129.0	24.0	0.0	0.0	51.0	8.0	360.0	47.0	44.0	335.0	260.0	49.0	
7	2012-10-01 18:00:00	61.0	57.0	123.0	32.0	0.0	0.0	65.0	8.0	360.0	55.0	57.0	334.0	258.0	63.0	
8	2012-10-01 19:00:00	75.0	70.0	117.0	39.0	0.0	0.0	79.0	7.0	360.0	64.0	70.0	333.0	256.0	77.0	
9	2012-10-01 20:00:00	89.0	83.0	110.0	47.0	0.0	0.0	94.0	7.0	360.0	72.0	83.0	332.0	253.0	92.0	

First, we imported all csv files to 7 corresponding tables to mysql database and applied normalization in order to reduce redundancy and improve data integrity. Also, we were interested only in Texas cities. So we extracted Dallas, San Antonio and Houston data for all the 6 attributes and created the tables with the below table structure.



- **Wind_speed (135753 rows)**

datetime,DATETIME,
City, TEXT
Wind_speed,INT

- **Wind_direction (135755 rows)**

datetime,DATETIME,
City, TEXT
Wind_direction,INT

- **Pressure (135673 rows)**

datetime,DATETIME,
City, TEXT
Pressure,INT

- **Humidity (134755 rows)**

datetime,DATETIME,
City, TEXT
Humidity,INT

- **Temperature (135751 rows)**

datetime,DATETIME,
City, TEXT
Temperature,DOUBLE

- **Weather_description (135756 rows)**

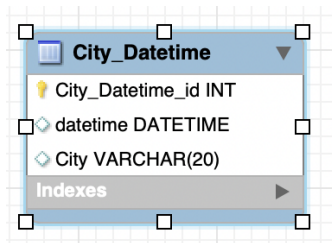
datetime,DATETIME,
City, TEXT
Weather_description,TEXT

- **City_attributes**

City, TEXT
Country,TEXT,
Latitude,DOUBLE,
Longitude,DOUBLE

City	Country	Latitude	Longitude
Dallas	United States	32.783058	-96.806671
Houston	United States	29.763281	-95.363274
San Antonio	United States	29.42412	-98.493629

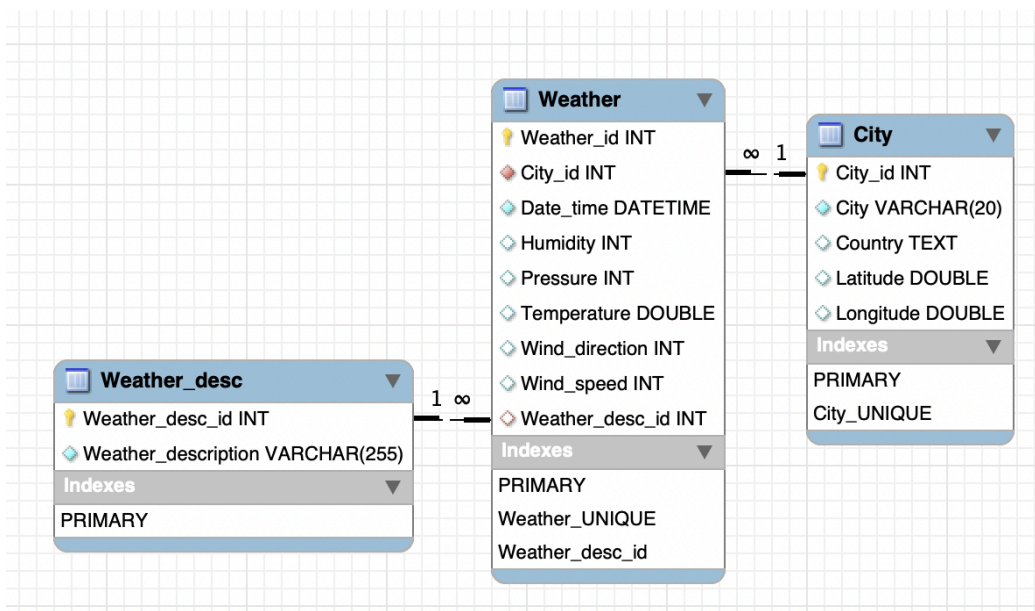
A unique index was created on the combination of datetime and city columns for all the tables. However, these tables were independent with no primary key and no foreign keys. This makes the joins between these tables complex and perform slower. So, we decided to create a table with City and datetime columns with a primary key City_datetime_id and to create foreign key relationship with all other tables on the City_datetime_id as shown below:



While this is normalized for data integrity, it requires complex joins to query different tables to get all values. This can impact read performance, especially with large datasets and makes it highly normalized for reporting and analysis purposes.

So, we decided to denormalize for improved query performances. Considering all tables have data for the same values of city and datetime. We combined all columns to one table with a unique combination and indexed on city and datetime fields. Also, we found there are only 39 unique weather descriptions. Hence for improved storage and to reduce redundancy we created a separate table with these unique descriptions. So we came up with 3 tables, of which one table “weather” has all weather information for a city at a particular hour of the day. Weather_desc has 39 rows that are all unique weather descriptions , City has 3 rows for Dallas, San Antonio and Houston.

Final database model we came up with is as shown below:



An insert/update script was developed to join these tables and create the final database model. The initial challenge we had was city being a text column, and the join between these tables on city and datetime timed out. However once the column datatype was changed to varchar(20), join queries ran faster and the timeout issue was resolved.

In order to further optimize the joins, we added a unique index on city and datetime column for all the tables.

- Indexes can drastically improve the performance of queries.
- Unique indexes enforce uniqueness of the data by not allowing duplicate values in the column or combination of columns on which the index has been created.
- Execution time for queries that join between the tables:
Before index creation: 0.101 seconds
After index creation: 0.0016 seconds

The Insert/Update script shown below was run for all tables to combine data to one weather table.

```
UPDATE Weather w
INNER JOIN City c ON w.City_id = c.City_id
INNER JOIN pressure p ON w.date_time = p.datetime AND c.City = p.City
SET w.Pressure = p.pressure;

INSERT INTO Weather(City_id, Date_time, Pressure)
SELECT City_id, datetime, pressure FROM pressure p
INNER JOIN City ON p.City=City.City
WHERE NOT EXISTS(SELECT 1 FROM Weather w where w.City_id=City.City_id and w.Date_time = p.datetime);
```

The script ran faster with the indexes and a unique combination of 135,756 rows were added to the weather table with all attributes from 6 independent tables. Since the temperature was in Kelvin, we updated the temperature to Fahrenheit before we moved further for data analysis.

We also created views that join the description and city table, also group by the date_time field on day and month for easier data access.

```
CREATE VIEW weather_vw AS
SELECT
    c.City,
    date_time,
    temperature,
    Humidity,
    Pressure,
    Wind_direction,
    wind_speed,
    wd.weather_description
FROM weather w
inner join City c on w.City_id=c.City_id
inner join weather_desc wd on w.weather_desc_id=wd.weather_desc_id;
```

```
CREATE VIEW weatherbydt_vw AS
SELECT date(date_time) as date,
       max(temperature) as MaxTemperature,
       avg(Humidity) as AvgHumidity,
       avg(Pressure) as AvgPressure,
       avg(Wind_direction) as AvgWindDirection,
       avg(wind_speed) as AvgWindSpeed
FROM weather group by date(date_time);
```

44 • `select * from weather_vw;`

45 •

100% 26:44 2 errors found

Result Grid Filter Rows: Search Export: Fetch rows:

City	date_time	temperature	Humidity	Pressure	Wind_directi...	wind_speed	weather_descript
Dallas	2012-10-01 13:00:00	289.74	87	1011	340	3	mist
Dallas	2012-10-01 14:00:00	289.7629742	86	1011	339	3	sky is clear
Dallas	2012-10-01 15:00:00	289.8307669	86	1011	338	3	sky is clear
Dallas	2012-10-01 16:00:00	289.8985597	86	1011	337	3	sky is clear
Dallas	2012-10-01 17:00:00	289.9663524	86	1011	335	3	sky is clear
Dallas	2012-10-01 18:00:00	290.0341452	86	1011	334	3	sky is clear
Dallas	2012-10-01 19:00:00	290.1019379	85	1011	333	3	sky is clear
Dallas	2012-10-01 20:00:00	290.1697307	85	1011	332	3	sky is clear

45 • `select * from weatherbydt_vw;`

46

00% 1:46 2 errors found

Result Grid Filter Rows: Search Export: Fetch rows:

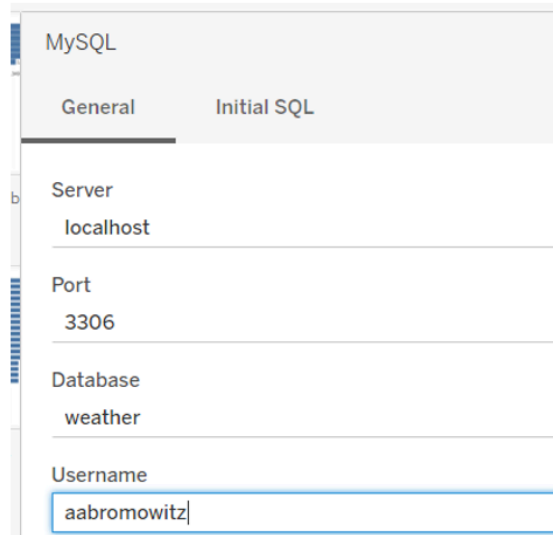
date	MaxTemperature	AvgHumidity	AvgPressure	AvgWindDirecti...	AvgWindSpeed
2012-10-01	290.3731089	88.0303	1011.0303	216.6667	1.5152
2012-10-02	301.13	70.2083	1010.8472	256.3750	2.6528
2012-10-03	300.04	62.1528	1013.3611	237.7639	2.2917
2012-10-04	301.77	64.0833	1015.3333	90.6806	1.6111
2012-10-05	304.81	66.4444	1018.0000	143.5417	3.2361
2012-10-06	303.85	70.2083	1018.4444	150.6250	2.7361
2012-10-07	302.88	73.7222	1017.8194	65.4306	2.3611

Having a normalized database gave us the data that we could then use in the data visualization and model creation aspects of the project. The indexes and views allowed us to access this data quickly.

Dashboard

To visualize the data and further our analysis, we used Tableau to make a Dashboard. Temperature, humidity, pressure, and wind speed are the most important data to our analysis, so those were included in the Dashboard. This data was plotted over the time period of the data set to show trends.

The data for the dashboard was established through a connection to the MySQL database, which took a few steps to set up. Before making the connection, you had to install a driver for MySQL. And in order to publish the dashboard however, the MySQL data needed to be extracted and saved off in a .hyper file. The dashboard could then be published online so that others can see it: [Dashboard](#).

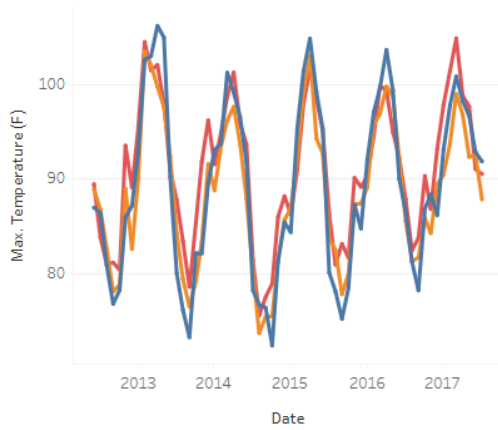


The image shows a screenshot of the MySQL configuration window. The window has a title bar that says "MySQL". Below the title bar, there are two tabs: "General" and "Initial SQL". The "General" tab is selected. The window contains several fields for configuration:

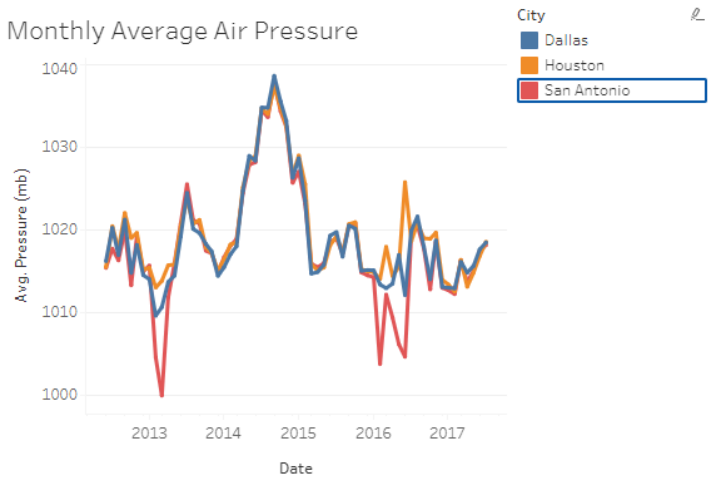
- Server:** localhost
- Port:** 3306
- Database:** weather
- Username:** aabromowitz

The Dashboard is interactive. It allows for the user to select which city they'd like to see the data for: Dallas, Houston, or San Antonio. The default is to show data for all three cities superimposed.

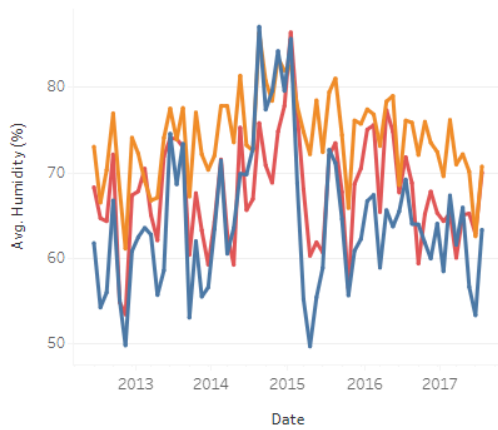
Monthly Maximum Temperature



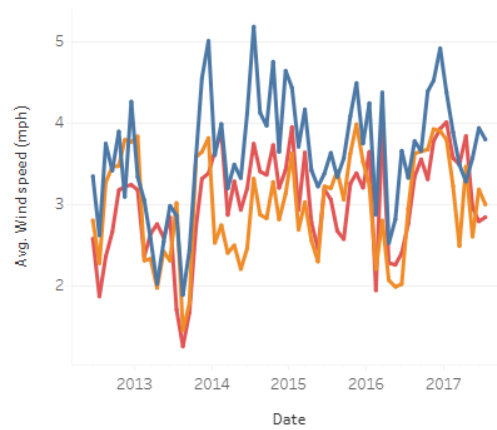
Monthly Average Air Pressure



Monthly Average Humidity



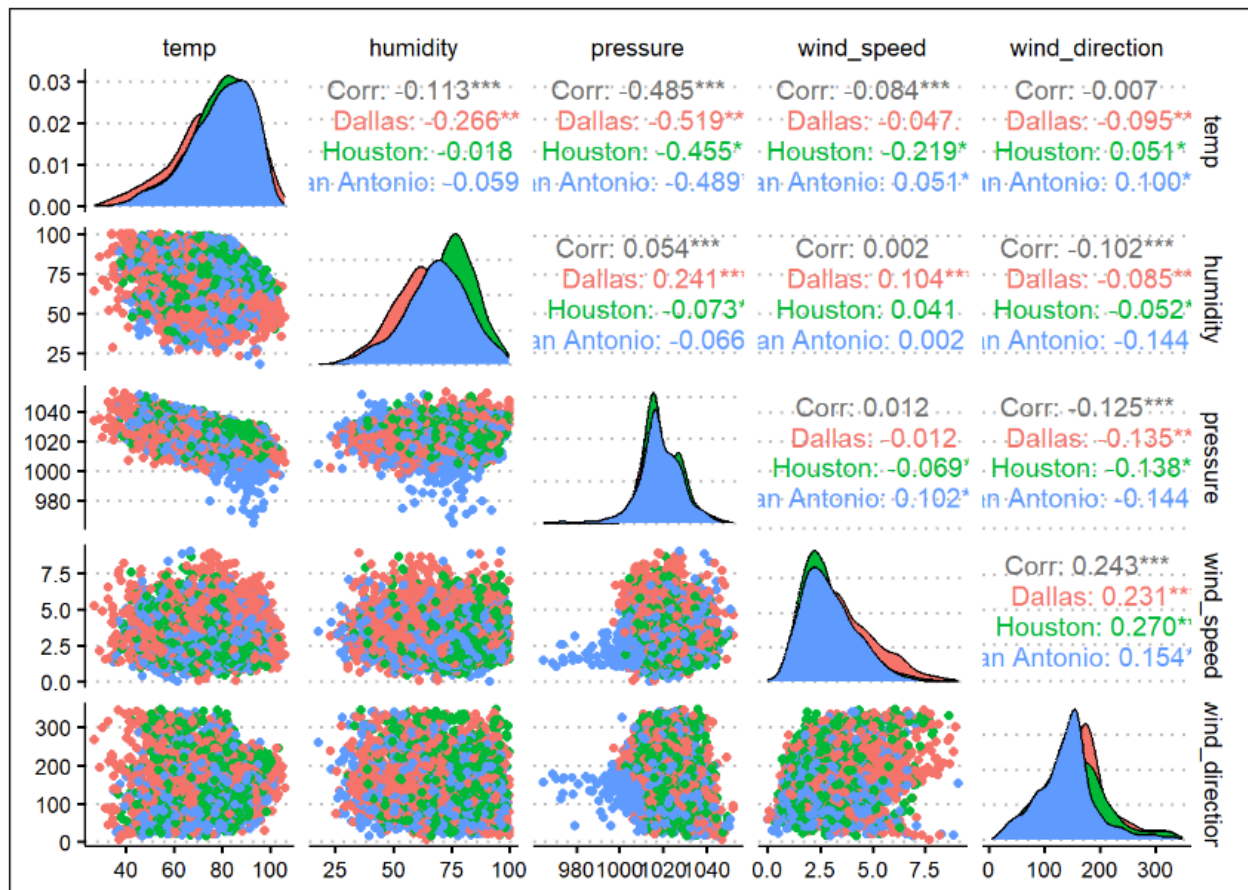
Monthly Average Wind Speed



Exploratory Data Analysis

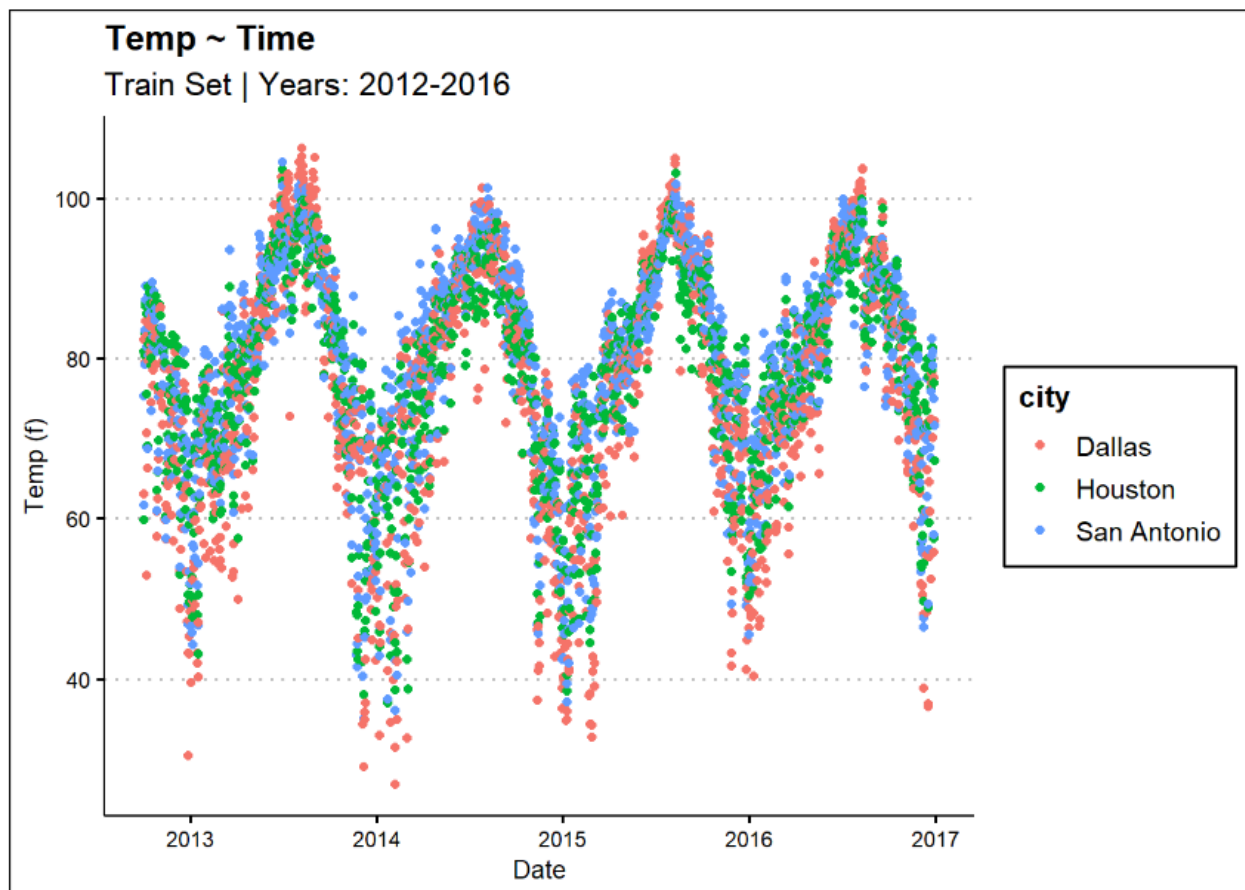
Data analysis and model creation were performed in the R programming language. In order to access the data from R, we established a connection to the MySQL database using the RMySQL package. This allowed us to directly interact with our database without needing to export the data.

As part of our analysis it was crucial to perform exploratory data analysis to understand the data we were working with. Our EDA is based solely on the training data set which spans from 2012-2016 (leaving 2017 out). Because we had to consolidate our values by the day instead of by the hour since our data was too big, we used the average of humidity, pressure, wind_speed, and wind_direction while using the max temperature for each day. Additionally, the following units of measurement were used for each metrics: temp = fahrenheit, humidity = percentage, pressure = hPa, wind_speed = m/s, and wind_direction = meteorological degrees.

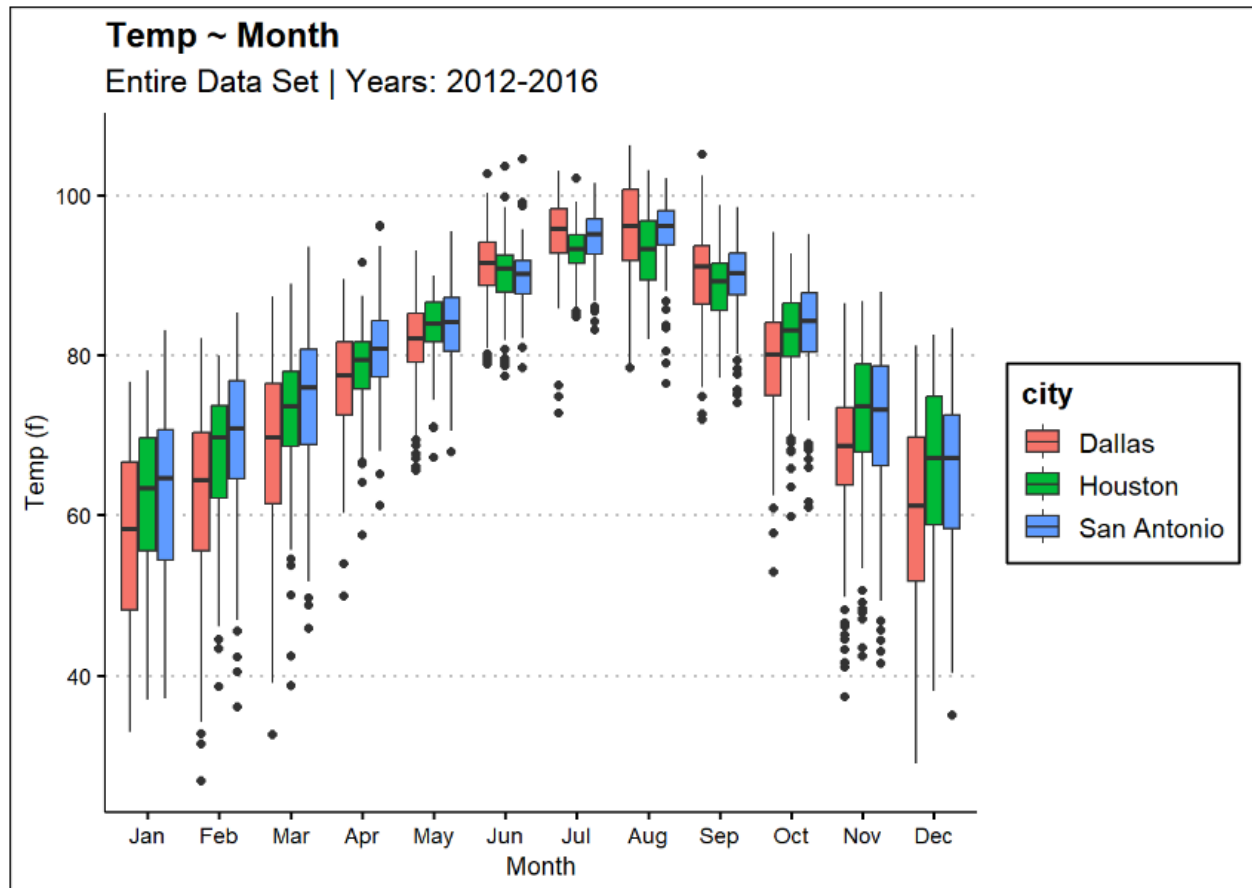


The first stage of EDA involved conducting a generalized view of the response and predictor variables as seen above. Our intent was to identify any correlation between the response and independent variables as well as identifying any collinearity between the predictor variables. We did not identify any collinearity and only found a negative correlation between temp and pressure. The only predictor variable that was missing from this plot was time. We

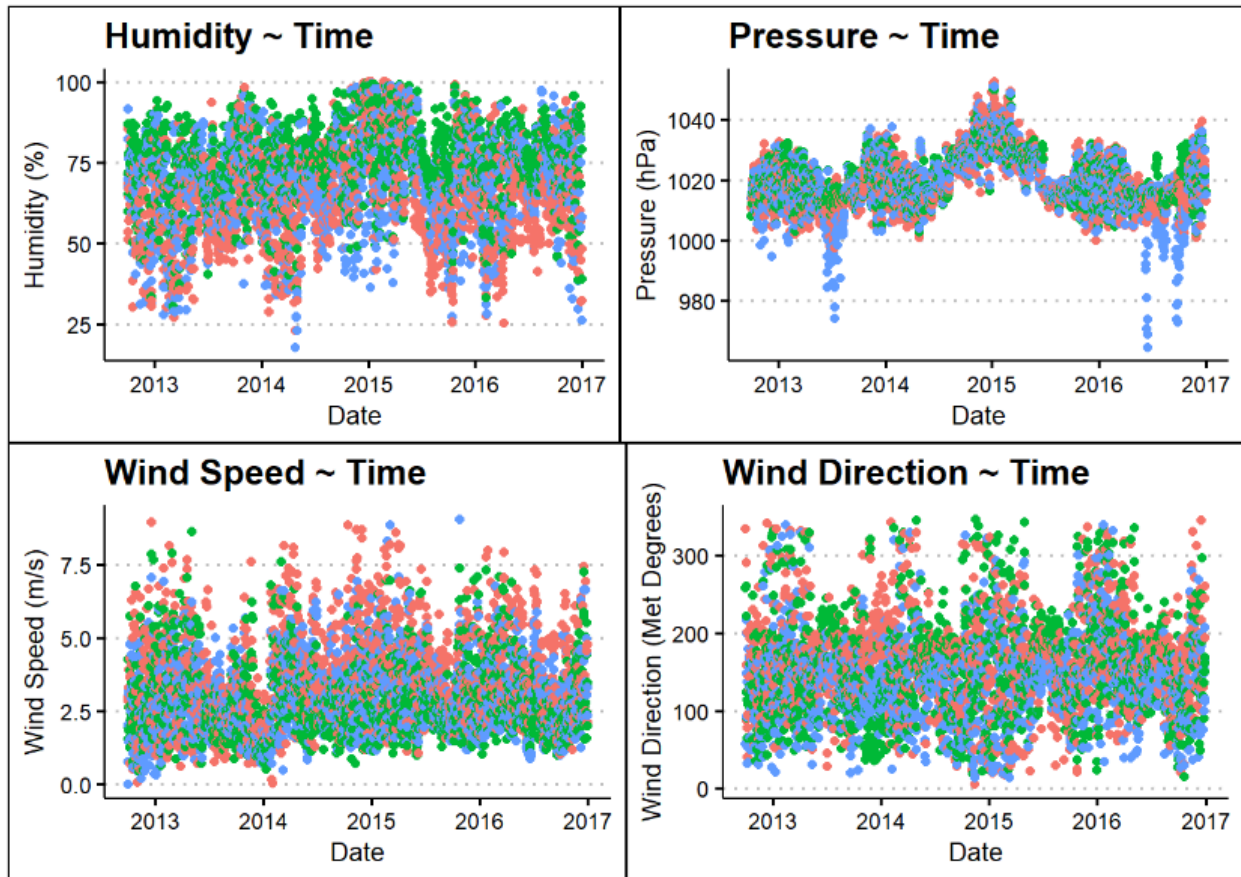
decided to give this its own plot since we knew it'd play a crucial role as this data set was appropriate for time series modeling.



After plotting temp with respect to date, we were not surprised that a pattern over the four years was established. Across all three cities, temperature was at its lowest during the end and beginning of the year and at its highest towards the middle of the year. The scatter plot (above) along with the box plots (below) were a clear indicator that we may want to utilize time-series models to predict our test set (2017).

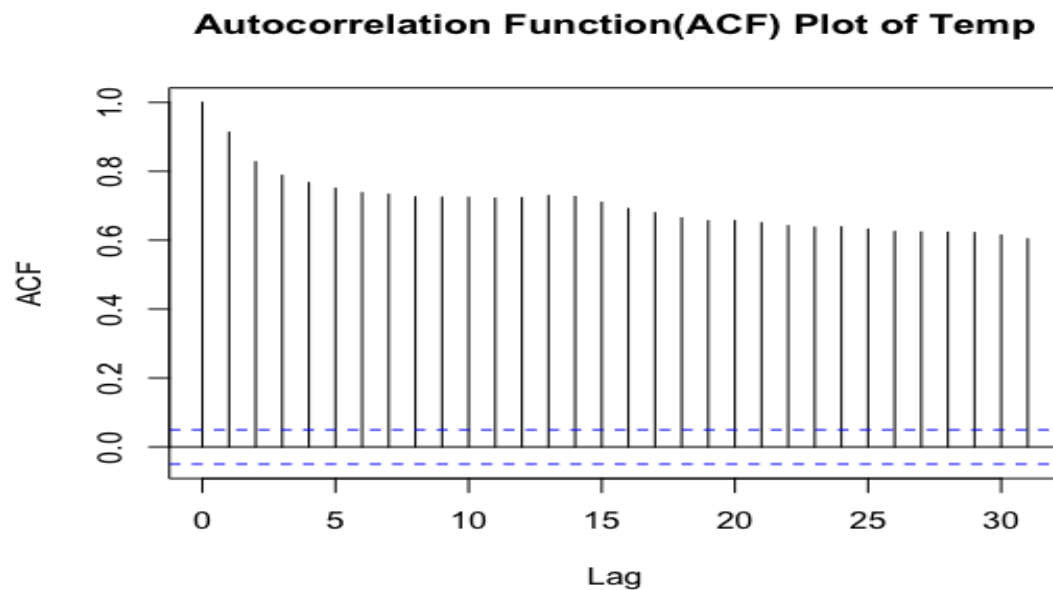


Finally, to conclude our EDA, we plotted time with respect to the other predictors (below) to see if we could identify a trend of some sort. As we can see, there wasn't much of a discernible pattern.



The key takeaway from our exploratory data analysis was that there was no collinearity present, there was a negative linear correlation between temp and pressure, and the correlation between temp and date was a seasonal one that allowed for the application of time series models to be implemented.

To determine how related the temperature is to the time variable, we looked at Autocorrelation. This is looking at the correlation value between the current time and the time X days ago. The further back in time the data goes, the less correlated it will be. Here is the plot of the lag term vs the correlation.



Since a correlation value of above 0.5 is fairly correlated, it appears that temperature data is highly correlated with past data. The correlation value is above 0.6 even going back 25 days. To confirm this, we ran a Durbin-Watson test and calculated a DW metric of 0.22729. This tells us that since the DW value is very close to 0 there is positive autocorrelation. The overall conclusion is that a time series approach will make sense.

Predictive Models

The models used maximum daily temperature as the variable of interest. Our goal was to accurately predict today's maximum temperature, given the prior day's data along with data back to 2012. As mentioned previously, the potential explanatory variables included: temperature, pressure, humidity, wind speed, wind direction, and weather description. These could be from the previous day, two days ago, etc. all the way back to the beginning of the dataset.

We compared four models to see which did the best job at predicting future data. The Root Mean Squared Error (RMSE) for the 2017 data was used as the comparison. The training and analysis of the models was only performed on data from 2012-2016. This allowed the future data to have no influence over our model creation. We also only limited our model creation and testing to Dallas, but the same approach could be used for other cities.

The first model we created was a simple linear model. This was intended as a basic model to compare the Time Series models against. It used month as a categorical variable, which functioned as a way to add seasonality to the data. The best way of knowing today's weather is to know what the weather was like yesterday. So we included the temperature going back 3 days, as well as yesterday's humidity. Adding data any further back or using either wind speed or air pressure didn't seem to improve model performance. So the final form of the basic linear model that we chose was:

Temp ~ Month + Temp(day-1) + Temp(day-2) + Temp(day-3) + Humidity(day-1)

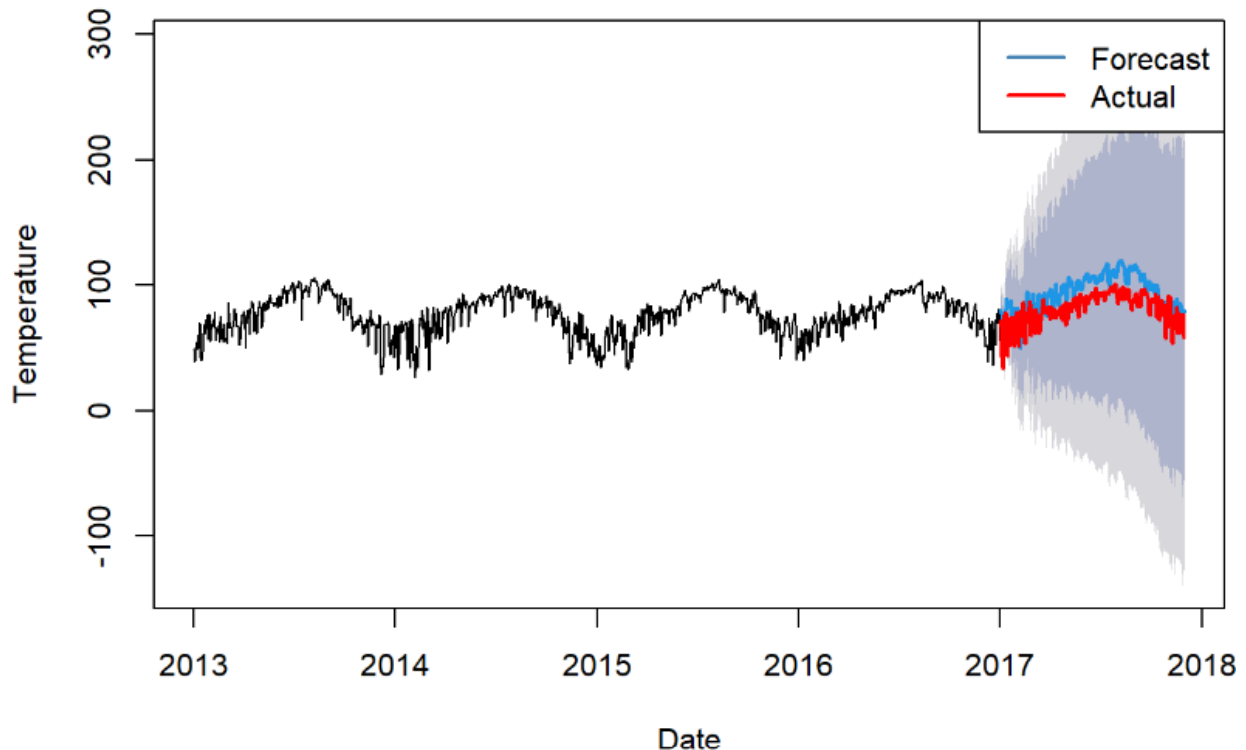
The coefficient table for the linear model is as follows:

Coefficients:					
	Estimate	Std. Error	t value	Pr(> t)	
(Intercept)	30.11261	2.06408	14.589	< 2e-16	***
monthAugust	6.30932	0.86241	7.316	4.10e-13	***
monthDecember	-5.25360	0.78438	-6.698	2.96e-11	***
monthFebruary	-5.26406	0.83938	-6.271	4.64e-10	***
monthJanuary	-6.07133	0.87285	-6.956	5.17e-12	***
monthJuly	6.14457	0.85499	7.187	1.03e-12	***
monthJune	5.63441	0.83129	6.778	1.73e-11	***
monthMarch	-2.63010	0.78462	-3.352	0.000822	***
monthMay	2.26136	0.76686	2.949	0.003238	**
monthNovember	-2.78466	0.74021	-3.762	0.000175	***
monthOctober	0.96026	0.72591	1.323	0.186090	
monthSeptember	4.39747	0.82361	5.339	1.07e-07	***
temp_1	0.81877	0.02546	32.159	< 2e-16	***
temp_2	-0.26526	0.03236	-8.197	5.16e-16	***
temp_3	0.10567	0.02530	4.176	3.13e-05	***
humidity_1	-0.06535	0.01059	-6.171	8.68e-10	***

When we calculated the RMSE for data in 2017, the value was 5.912. This is roughly 10% of an average temperature (in Fahrenheit), so a good starting point for a simple model.

Our next model was a Holt's Seasonal model. This is a time-series forecasting technique that focuses solely on the time and the response variable (temp). This model can be tuned by altering the smoothing, leveling and seasonal components. However, we left each of these thresholds at their default values. The only feature we altered was specifying our model to be an additive one instead of a multiplicative one. Specifying that our model was an additive one meant that our pattern was roughly constant throughout the series. That is, our temperatures were consistently cold during the beginning of the year and hot during the middle of the year across all four years.

2017 Temperature Forecast vs Actual



After feeding our training set into the model, we can observe the forecasted vs the actual temperatures for 2017 (above). Our MAE (Mean Absolute Error) was 14.04 degrees which translates to say that on average, the forecast is off by 14.04 degrees. This is not a desirable MAE, however, we could definitely tune it if more time was allowed. Our RMSE was 16.32 which is slightly higher because this metric penalizes larger errors by squaring them first. In conclusion, this model was the weakest out of all of the models that we attempted. However, we could potentially improve the predicted values by fine tuning its features.

The next model that we worked on was the SARIMA model. The SARIMA model is an extension for the ARIMA model which stands for autoregressive integrated moving average; the difference between the two is that the SARIMA accounts seasonality while the ARIMA model doesn't. When we were performing our EDA we found signs of autocorrelation and with understanding that our data would have seasonality in it since it's a weather dataset we decided to move forward with the SARIMA model. Similar to the holt seasonal model, our SARIMA model focused only on the time and temp variable. The SARIMA model adds in three new auto parameters that the ARIMA doesn't have those are autoregression, differencing, and moving average. After doing some EDA such as running ACF/PACF plots and running the `auto.arima` function in R we're able to tune our model to perform optimally with our data. There are four seasonal elements in the SARIMA model that we needed to tune P - seasonal autoregressive order, D - seasonal difference order, Q - seasonal moving average order, and m - the number of time steps for a single seasonal period. We set these values at P = 5, D = 0, Q = 1, and m = 1. For our final SARIMA model we had a RMSE value of 5.43787 and a MAE value of 3.978499. Below is a summary of our final SARIMA model:

Call:

```
arima(x = univariate_ts2, order = c(5, 0, 1), seasonal = list(order = c(5, 0, 1), period = 1))
```

Coefficients:

	ar1	ar2	ar3	ar4	ar5	ma1	sar1	sar2	sar3	sar4
	1.9903	-1.5334	0.9809	-0.8693	0.4262	-0.9155	0.6794	0.0953	-0.1950	0.2676
s.e.	0.1569	0.3617	0.3611	0.2313	0.0840	NaN	0.1772	0.1738	0.1064	0.1190
	sar5	sma1	intercept							
	0.1344	-0.9161	64.0155							
s.e.	0.1298	NaN	19.3990							

sigma^2 estimated as 29.85: log likelihood = -1043.48, aic = 2114.97

Training set error measures:

	ME	RMSE	MAE	MPE	MAPE	MASE	ACF1
Training set	0.1539997	5.463787	3.978499	-0.3533463	5.45172	0.891374	-0.000773112

The final Time Series model that we tried is called a Vector Autoregressive (VAR) model. This type of model uses time series approaches for the main variable (temperature), but also uses other variables (humidity, air pressure, wind speed). The model is included in the vars package in R. Some useful parameters include the amount of steps to go back (p), whether to include seasonality, any exogenous variables to include, and if a constant or trend term should be included as well.

After some testing using different parameters, we determined that a well performing model set $p = 12$, to not include seasonality, and to include both a constant and a trend term. The formula for the model looked like this:

```
temp = temp.l1 + humidity.l1 + pressure.l1 + wind_speed.l1 + temp.l2 + humidity.l2 + pressure.l2 + wind_speed.l2 + temp.l3 + humidity.l3 + pressure.l3 + wind_speed.l3 + temp.l4 + humidity.l4 + pressure.l4 + wind_speed.l4 + temp.l5 + humidity.l5 + pressure.l5 + wind_speed.l5 + temp.l6 + humidity.l6 + pressure.l6 + wind_speed.l6 + temp.l7 + humidity.l7 + pressure.l7 + wind_speed.l7 + temp.l8 + humidity.l8 + pressure.l8 + wind_speed.l8 + temp.l9 + humidity.l9 + pressure.l9 + wind_speed.l9 + temp.l10 + humidity.l10 + pressure.l10 + wind_speed.l10 + temp.l11 + humidity.l11 + pressure.l11 + wind_speed.l11 + temp.l12 + humidity.l12 + pressure.l12 + wind_speed.l12 + const + trend
```

There are lag terms going from 1 to 12 for each of the four variables (temp, humidity, pressure, and wind_speed). At the end is the const and the trend terms. A parameter table was also included, which included both estimates and p-values:

	Estimate	Std. Error	t value	Pr(> t)	
temp.11	8.663e-01	2.585e-02	33.516	< 2e-16	***
humidity.11	-6.577e-02	1.536e-02	-4.281	1.98e-05	***
pressure.11	1.689e-04	6.787e-03	0.025	0.98015	
wind_speed.11	-9.011e-01	1.150e-01	-7.839	8.58e-15	***
temp.12	-2.165e-01	3.488e-02	-6.207	6.99e-10	***
humidity.12	2.526e-02	1.925e-02	1.312	0.18955	
pressure.12	1.558e-02	1.021e-02	1.526	0.12724	
wind_speed.12	4.001e-01	1.278e-01	3.131	0.00178	**
temp.13	1.018e-01	3.562e-02	2.858	0.00433	**
humidity.13	-3.711e-02	1.954e-02	-1.899	0.05778	.
pressure.13	-2.048e-02	1.198e-02	-1.709	0.08762	.
wind_speed.13	-1.924e-01	1.282e-01	-1.500	0.13371	
temp.14	1.062e-02	3.566e-02	0.298	0.76577	
humidity.14	3.357e-02	1.960e-02	1.713	0.08685	.
pressure.14	3.037e-02	1.286e-02	2.361	0.01834	*
wind_speed.14	-2.088e-02	1.281e-01	-0.163	0.87051	
temp.15	5.714e-02	3.558e-02	1.606	0.10853	
humidity.15	4.703e-03	1.952e-02	0.241	0.80961	
pressure.15	-1.934e-02	1.326e-02	-1.459	0.14491	
wind_speed.15	-2.047e-01	1.281e-01	-1.598	0.11034	

As one would expect, the p-values generally decrease as the lag increases. In addition, the coefficient values seem to match up surprisingly well with the values from the basic linear model. However, the p-values for the VAR model seem to be smaller.

After we created our four models, we calculated RMSE for each using the 2017 data. The results of this comparison were as follows:

Model	RMSE
Basic Linear	5.912
Holt Seasonal	16.32
SARIMA	5.46
VAR	5.784

The best performing model on 2017 data was the SARIMA model. The worst performing model was the Holt Seasonal model. Most of the models, including the Basic Linear model which we used as a baseline, had an RMSE of below 6. This implies ~10% of errors in our models, which is a fairly good predictive value.

Conclusion

As we look to the future, the need for more precise weather predictions grows, particularly in the face of climate change and it remains a vital and dynamically evolving discipline that significantly impacts both daily lives and long-term planning across various sectors. The focus of our project was on the curation, visualization, and forecasting of meteorological data. We began by importing the data into a MySQL database, providing a structured platform for storage and management. For visualization, we leveraged Tableau's robust analytics capabilities to create insightful and interactive representations of the weather data. Lastly, to project future weather trends, we employed time series analytical models, which enabled us to forecast upcoming conditions with a degree of precision.