D214 Data Analytics Graduate Capstone Project Assessment 2:

Univariate Time Series Analysis of Airline Arrival Delays

Nathanael Ellis

Western Governors University

D214: Data Analytics Graduate Capstone

Dr. Daniel Smith

September 11, 2022

**Research Question**

Over the past few months, airline flight delays in the United States have received a large amount of media attention. An article by Leslie Josephs states, “Already this year, the rate of flight cancellations and delays in June was higher than before the pandemic as a result of bad weather and staffing shortages.” (2022). Other articles similar to this highlight this current issue. Some even offer potential explanations for these delays. Understanding delay information is crucial to many entities including airlines, air traffic controllers, airfield managers, and travelers.

This problem provides a number of potential questions suitable for Data Analysis. The research question I pose for this project views the problem at a basic, yet core level. Can we create an ARIMA model that can predict the average number of delayed airline flights at major airports to within a 95% confidence interval? Creating a time series model with the ability to predict this data could assist in forecasting future delays. Airlines could use such a model to improve staffing, airport managers could better accommodate delayed travelers, and air traffic control could better prepare for upcoming changes.

The corresponding hypotheses based off the research question are the following:

Ho: The number of delayed airline flights at major airports cannot be tracked by an ARIMA model to the 95% confidence interval.

Ha: We can track the number of delayed airline flights at major airports within a 95% confidence interval using an ARIMA model.

To test these hypotheses, we will need to construct an ARIMA time series model of airline flight delay data. Aggregate airline delay count was chosen as the dependent variable so that we can view the trend across all airlines. The same reasoning was used for the choice of all major airports. Discussion of the ARIMA model selection occurs later on in the “Analysis” section of this report. Once a model is constructed, we can view performance using summary statistics. We can then also compare these statistics to a baseline or another model to help judge the ARIMA model efficacy.

**Data Collection**

The data required for this project consists of flight summary information with anticipated and actual flight arrival times. Fortunately, this data was publicly available through the Federal Aviation Administration (FAA) and the Bureau of Transportation Statistics (2022). 14 Code of Federal Regulations (CFR) 234 and 241 require airlines to report this information (as well as other flight summary statistics) to the BTS. It is then downloadable from their website in the form of compressed .csv files. I chose to gather data from January 2017 all the way through May 2022. I did this as five years of information would yield plenty of data for the ARIMA model to use to detect any seasonality.

The accessibility of this data was a huge advantage made possible by federal regulation. One disadvantage was that data retrieval from the website was limited by the website’s interface. I had to download each file month by month for the chosen time period (5 years). Another disadvantage of this configuration is that any future analysis using data from this website will require manual downloads. Finally, data is limited to the information currently available to the BTS. I was only able to retrieve up to May 2022, over two months older than the current date when the data was first retrieved on August 14, 2022.

Once downloaded, I could begin to review the data and chose next steps to prepare it for the extraction process. As mentioned, the data was stored in separate monthly .csv files. Their decompressed sizes ranged anywhere from about 50 MB all the way up to almost 200MB per file. Each file contained 109 unique features of individual flight information. Altogether, there were 32,819,892 different flights

To organize the data, I tried two different approaches. The first was to use a local PostgreSQL server to store the data. Using this approach would allow me to retrieve only the relevant features I needed without bogging down my machine’s memory. It also would have allowed me to easily retrieve other data from the original set if I determined I needed other information later on. Unfortunately, the size of the data when uncompressed took up a large amount of local hard drive space at 11GB. Due to that reason, I opted to try another approach.

The second method, and the one I ultimately ended up using during this project, was to decompress the data and leave it in the monthly .csv files it came in. The main advantage to this approach was its simplicity. It did not require creating or extracting from a separate server. I could also easily read the data directly into my coding environment. The disadvantages of this method, similar to before, revolved around the overall size of the data. I had to read, clean, and extract the relevant data from each file separately so as not to use too much memory or storage space. The versatility of this method is limited as any future gathering of other features from the dataset would require going through the entire process of loading the full sets one by one all over again. In the end, as this would be a univariate analysis requiring extraction of only a few features, this disadvantage was deemed acceptable for this project. If we plan to use this data for other future analysis, then the need to migrate towards the PostgreSQL approach would become more necessary.

**Data Extraction and Preparation**

For the data extraction, cleaning, modeling, and analysis portions of this project, I chose to use the python programming language with Jupyter Notebook as my IDE. Python was chosen due to the availability of its many useful libraries. The NumPy (Harris, 2020) and Pandas (Reback, 2021) libraries contain objects to help import, contain, and transform data. Statsmodels (Seabold, 2010) aided in seasonal decomposition, Autocorrelation Function (ACF) plotting, and the Augmented Dickey-Fuller (ADF) tool. PMDArima (Smith et al., 2017) was used to create the ARIMA models. Functions from Kats (Jiang, 2022) and Prophet (Taylor, 2017) were used to create alternative models for comparisons. Finally, I used Matplotlib (Caswell, 2021) to help create visualizations in the programming environment.

As alluded to earlier, the data extraction process was accomplished by importing the data directly from csv files. A key step in data preparation is exploration. To better understand the data, I first loaded a single file. I checked the top five rows of the DataFrame to view the available features and see examples of the format of each column (See Figure 1). An added benefit to loading a single file to explore before loading everything was that I was able to create and test the functions I would use to check for and address duplicate and missing values. Once I had completed these steps, I began to read in the data in its entirety.

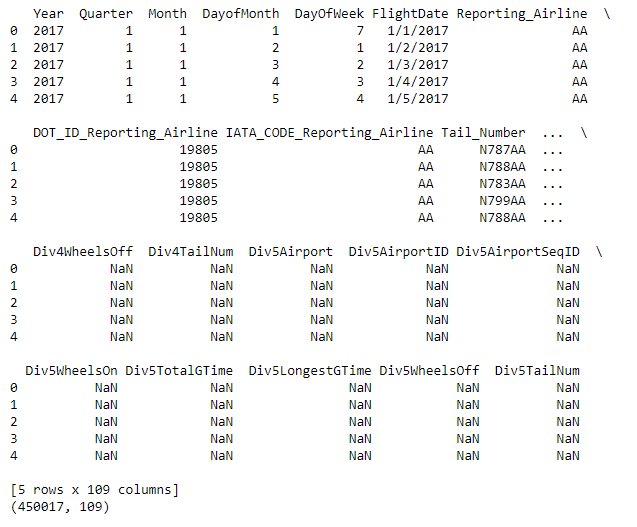


Figure 1: Exploratory DataFrame

Due to the size of the data, each file was read one by one. The advantage to extracting the data in this manner was that it saved system memory. A disadvantage was that it meant several functions had to be repeated multiple times increasing the overall process length. Once a file was read in, I inspected it for duplicate values. None were found in any of the files. The duplicate check process was accomplished by using a Pandas function to check that at least one feature in each row was different from other rows (i.e., no two rows were completely identical). The advantage to this method is that it ensures no duplicates values make it through to the analysis. The disadvantage is that if there were duplicate rows that were supposed to be there and not a mistake, they would be removed.

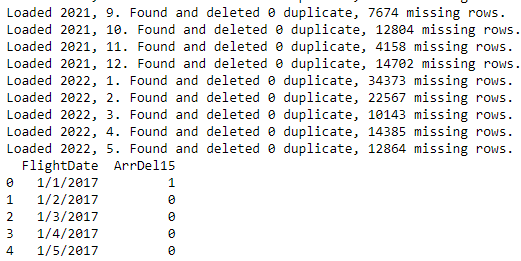


Figure 2: File loading and cleaning

I then proceeded to reduce the data to just three features and check those for missing values. Overall, 2.6% of the data was found to have missing values for the chosen features. Any rows with missing values were deleted. I used a pandas function to check for any null values in the data. The advantage to checking missing values this way is that it ensures no missing values make it though and cause issues during the analysis phase. A disadvantage is that missing values may be missing on purpose. An example of this would be the rows that the flight was cancelled. The reduced data set for each of the files were all combined together to make one aggregate set. This now contained 31,966,929 flights and the three chosen features.



Figure 3: Import and cleaning totals

The next step was to get the data into the correct format. The ‘FlightDate’ feature was changed to a datetime object and moved to the index. I then grouped the data by the date and summed together the other columns. This effectively created a row for each day with the total number of delays across all airlines and airports all together. 1977 days were included in this overall set spanning between January 2017 to May 2022. The mean number of delays per day was 2827.7 with a maximum value of 8217 and a minimum of 130. The median value for delays was 2703. To clarify, a delay is counted as any arrival time with an actual arrival time of 15 minutes later than the scheduled arrival time. Grouping data in this manner has the advantage that it is simple to interpret. A disadvantage of grouping in aggregate is that we lose insight from smaller grouped levels, such as if one airline was responsible for most of the delays.

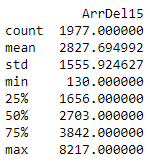


Figure 4: Summary statistics

The last step in my exploration process was to graph the time series and visually inspect it (See Figure 5). It quickly became noticeable that the variance and trend had some interesting characteristics. Around March through well into 2021, the number of delays drops well below the apparent trend, while the variance also shrinks. This was likely due to the effect of the Covid-19 pandemic on the US airline industry. Since this would likely affect any potential modeling using this time period, I elected to check if using percentage of flights delayed would provide a more reliable modeling metric. The same characteristics noticed in the count data were also seen in the percentage data. In the end, I chose to stay with the count data as stated in the hypotheses. An advantage to using visual inspections of a graph is that it provides the individual a unique perspective in order to better understand the underlying data. A disadvantage is that it is not a precise way to make decisions, as graph interpretation can be subjective and is dependent on how the graph was created. Scales can sometimes lead to false conclusions.

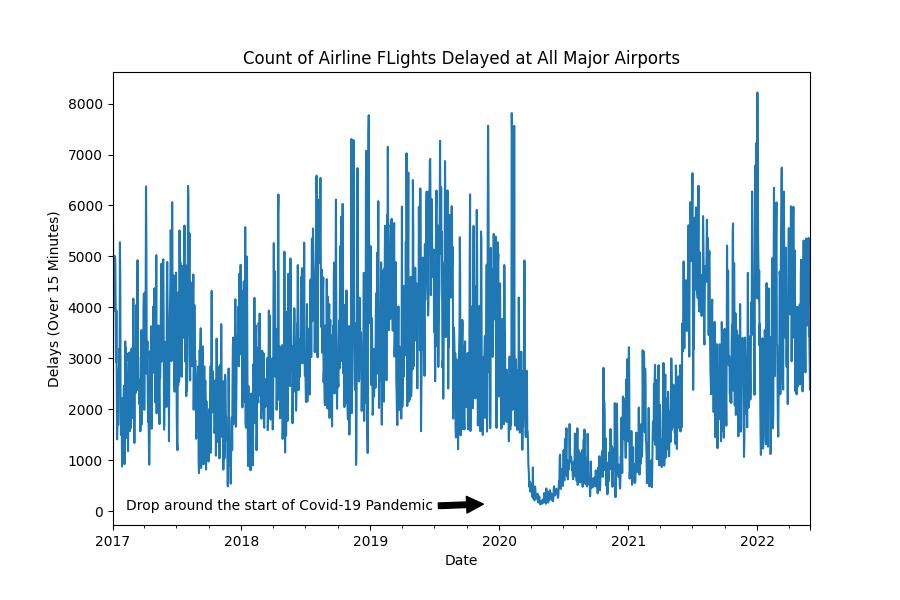


Figure 5: Time Series Exploratory Line plot

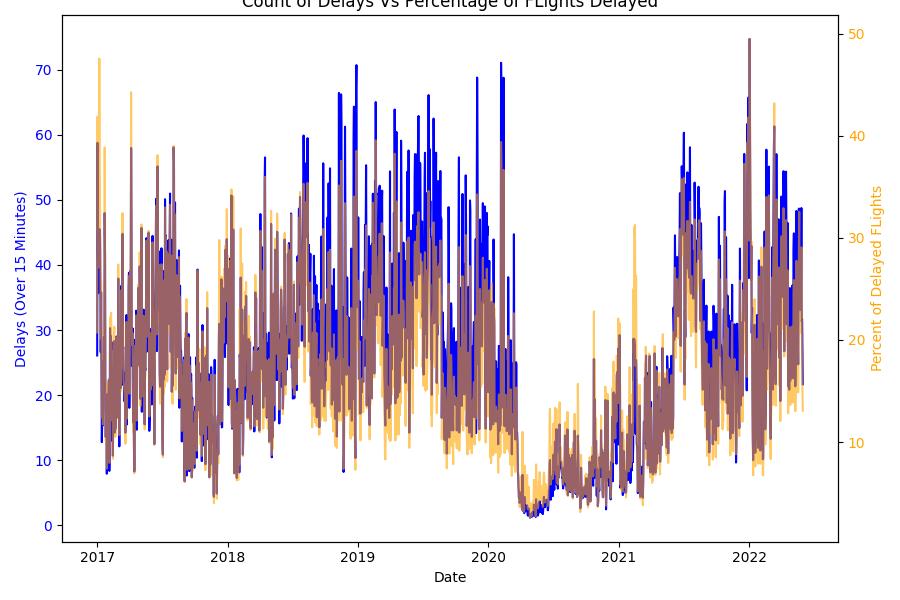


Figure 6: Delay Count vs Delay Percentage

The last steps of the data cleaning process involved handling outliers and ensuring the time series was continuous. Multiple outliers were detected using the Kats ‘outlierdetector’ function. This detector is excellent for timeseries, as it checks individual values against rolling means instead of aggregate. I decided to leave the outliers in place, as I viewed their information important to the series and the model I would create. The advantage being that the model has extra information to use for its creation. A disadvantage to leaving the outliers is that it can skew the model. As for continuity, the series was found to be continuous with no missing days.



Figure 7: Continuity Check

The final preparation step was to create separate Train/Test sets. The final 5% of the time series data was reserved for testing. This number was chosen as it left almost 100 days for testing (about three months). Predictive capabilities of the ARIMA model start to decay over time, as is evident in the expanding confidence interval bands created by the model. It also helps by including more data in the training set further past the Covid-19 pandemic trough. Advantages to splitting a data set in this manner is that it allows us to zero in on a specific prediction interval for the test portion of the data. Continuous splitting is also required for time series data, as randomized splitting would cause the data to lose the important relationships each datapoint shares with the data that precedes it. A disadvantage is that variations away from the overall trend (especially near the split) can have a large effect on the over prediction capabilities of the model.



Figure 8: Train/Test Split

**Analysis**

For this project, I chose to use an ARIMA based modeling approach to the time series problem. ARIMA models incorporate multiple terms for autoregressive elements (or lags) and moving averages (or windows). When ARIMA values are properly selected and fitted to time-series data, it can provide accurate and flexible predictions. An advantage to ARIMA models is that it can work well for short term forecasts with highly seasonal data. One disadvantage to the ARIMA model is that if new information becomes available, it cannot be added to this model. The model has to be recreated each time we want to include the latest information. Before creating the ARIMA model, we first need to review the data for stationarity and seasonality.

I used a combination of the ADF test as well as the Kwiatkowski–Phillips–Schmidt–Shin (KPSS) test to aid in determining time series stationarity and see if any differencing of the data would be required. Both tests indicated that differencing was not needed for the data set. Specifically, the ADF test returned a value of 0.01 which is high enough to reject the null hypothesis for the ADF test (that the series has a unit root). The advantage to using both of these tests in conjunction is that when they agree we can be fairly certain that no further differencing is needed.

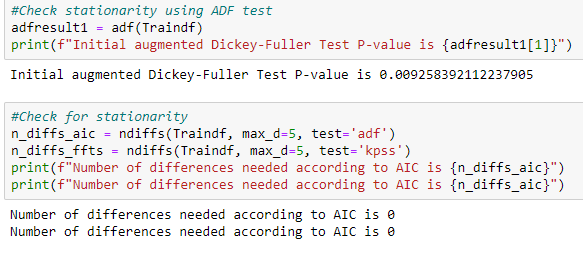


Figure 9: Stationarity check

The next step was to check for seasonality, which I again turned towards using two different methods. The first was to use ACF and PACF plots. As well as indicating possible starting values for the AR and MA terms in the ARIMA model, reoccurring cycles in one of the graphs can indicate seasonality. The PACF graph does appear to have several strong negative correlations occurring roughly every 7-8 days that are outside of the significance bounds. Kats provides a useful Fast Fourier Transformation tool to automatically detect any underlying seasonality. In this case, it also detected seasonality, but at a different interval of 3.5. These differences could be caused due to the different methods each processes uses to obtain the values. Both contribute towards the use of a weekly 7-day season.

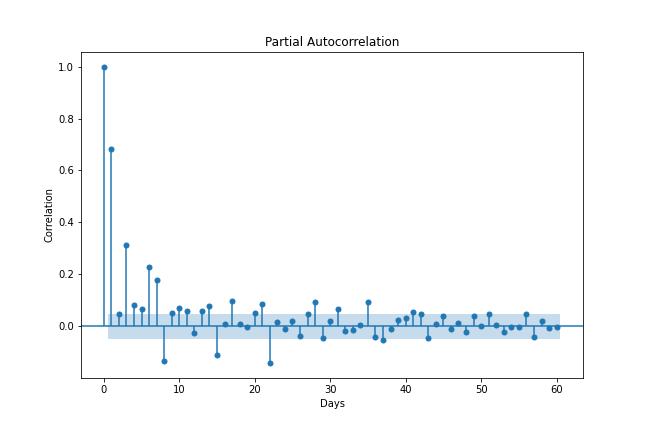


Figure 10: ACF Plot

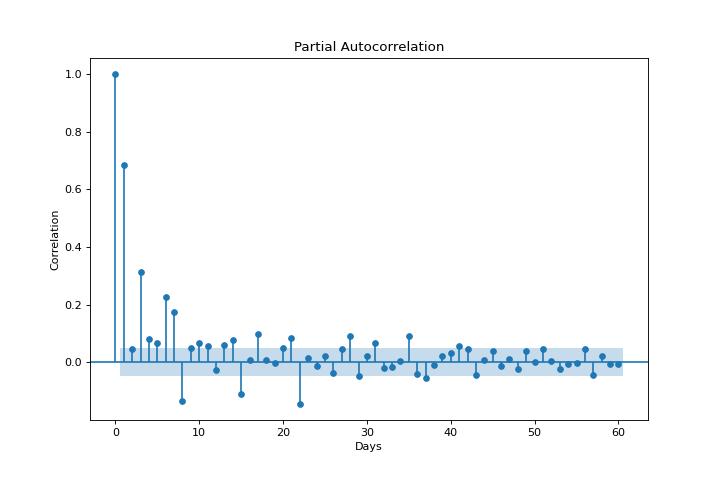


Figure 11: PACF Plot



Figure 12: FFT Detector

Another tool that I used was the seasonal decomposition tool. It breaks down the series into the underlying trend, season, and residual components in order better understand each part. The decomposition on our data reveals a couple interesting traits. The residuals have differing variance, especially around the 2020 time period noted before. The residuals also have a fairly uniform distribution with a very slight rightward skew. With seasonal and residual data removed, the trend displays a clearer representation of the diminished delay counts for 2020 through 2021.

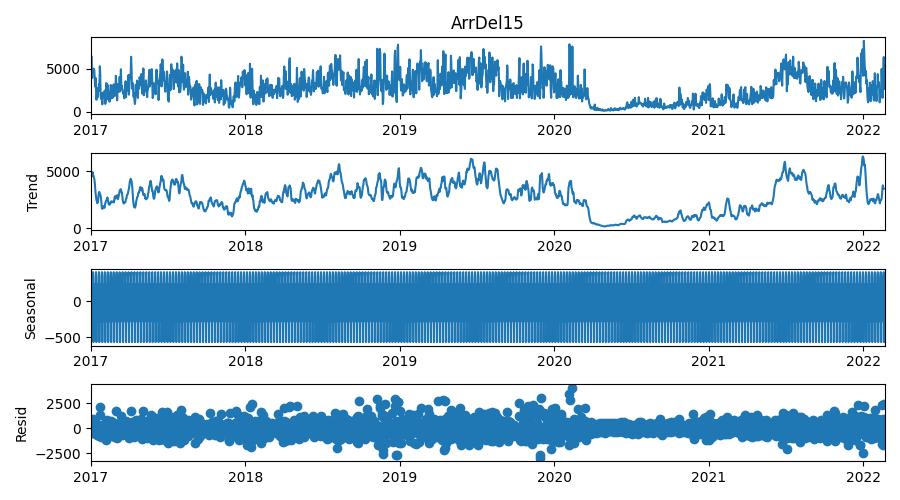


Figure 13: Seasonal Decompose

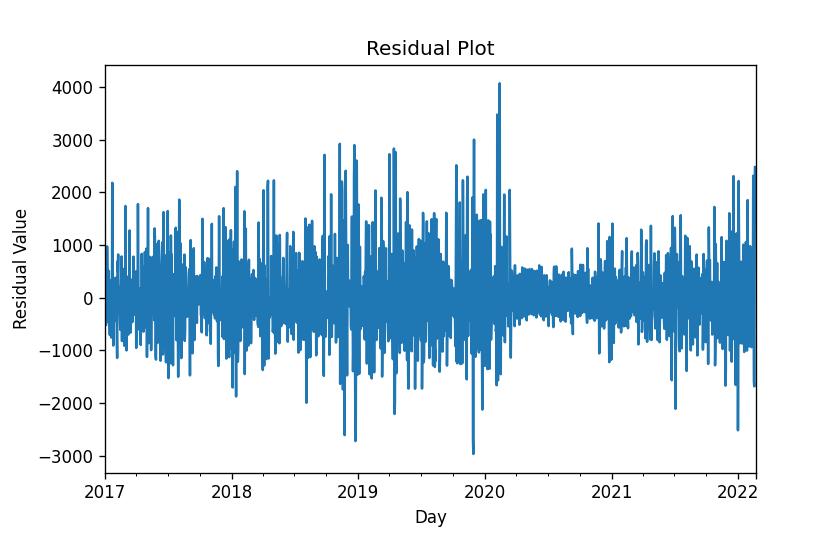


Figure 14: Residuals

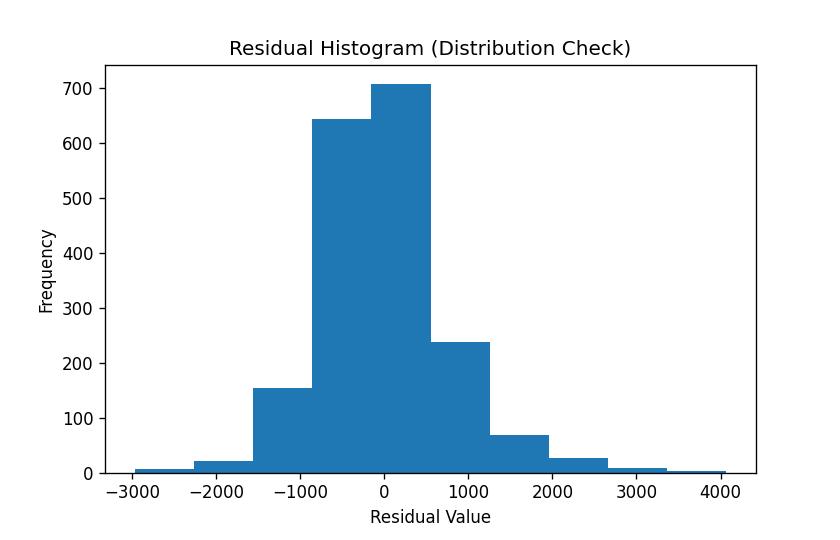


Figure 15: Residual Histogram

With the understanding that the series had a seasonal element and no differencing terms, I created an ARIMA based model using the auto-ARIMA tool. This tool has the advantage of checking multiple models to find the best performing one based off of a specified metric. I chose to use the ‘AIC’ metric as it has built in features to penalizing overly complex (and often overfitting) models.



Figure 16: Auto ARIMA

Using the newly created ARIMA model, I could then graph the results. The plot diagnostic show residuals with a close to normal but slightly skewed distribution. The correlogram shows that that ARIMA modes used account for any lag correlations. When viewing the ARIMA model with 95% confidence intervals against the training data (see Figure 18), the bands appear to be fairly wide but capture most of the variance. A few outliers appear to escape past the bounds. In a similar fashion, when viewing the graph of the model predictions against the reserved test data (see Figure 19), we can see that the trend appears to be traveling in the correct direction, wide confidence interval bands capture most of the points, and 9 values (9% of the test data) escape past the confidence interval.

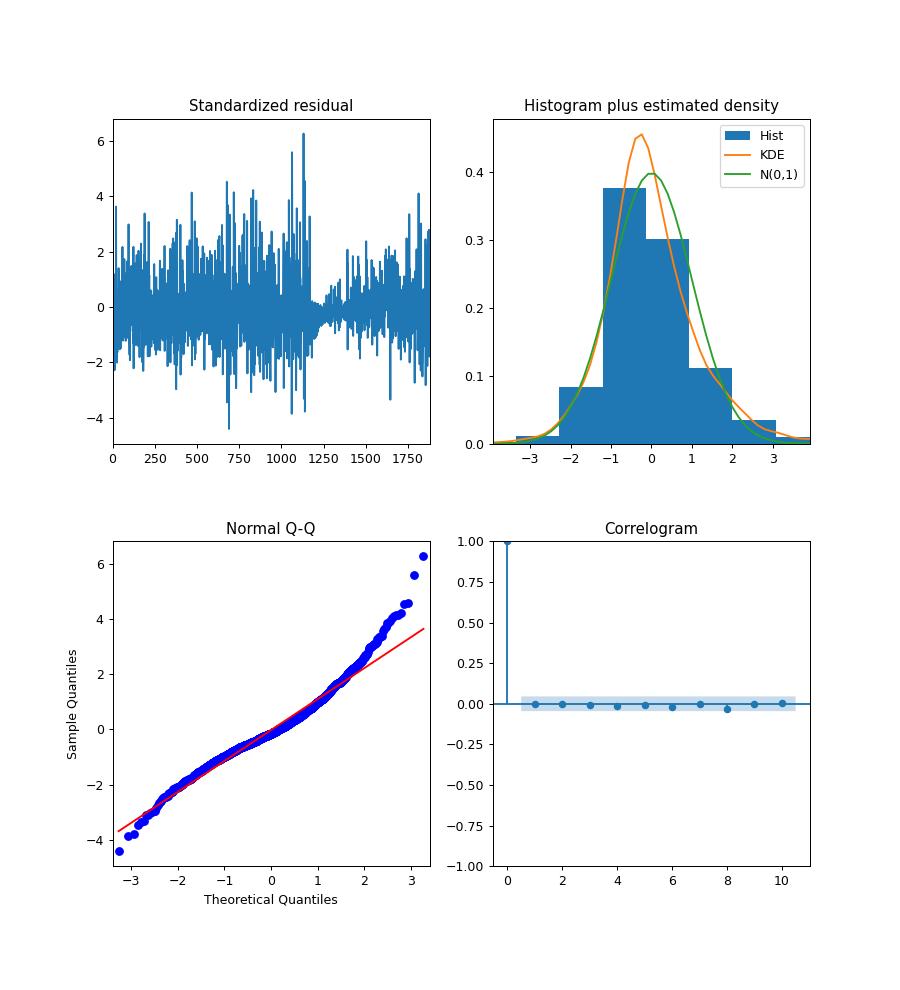


Figure 17: Plot Diagnostics

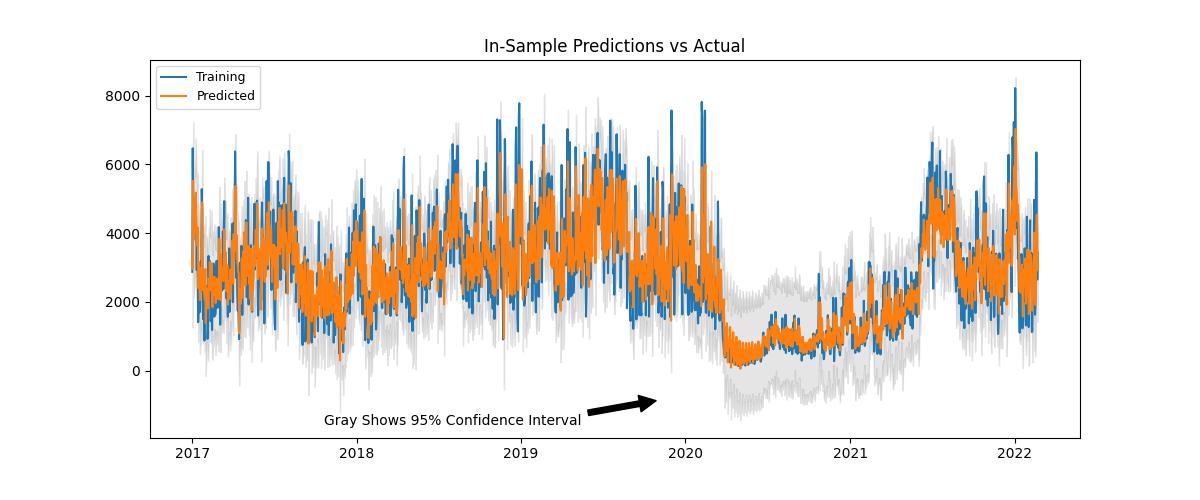


Figure 18: Train values vs model

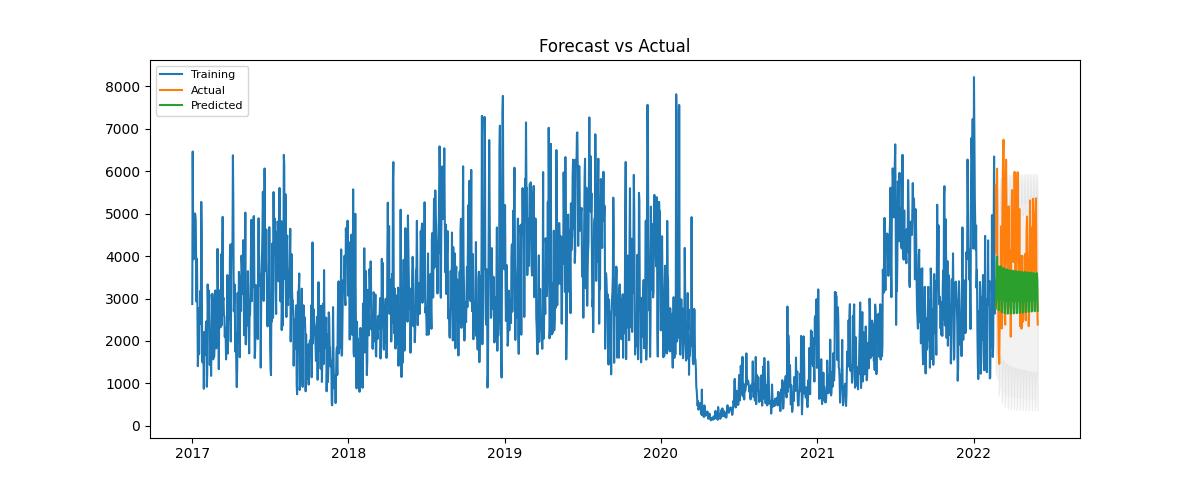


Figure 19: Test values and predictions



Figure 20: Number of values out of 95% Confidence Interval

ARIMA model performance was also analyzed using MAE and the model summary. The MAE value for the test data was 976. Analyzing the plot summary held interesting results. Most of the created Terms were statistically significant with the exception of the ar.L4 and ar.L5 terms, but most of them had fairly small coefficients at under 1. The Ljung-box metrics tell us we cannot reject the null hypothesis that residuals are white noise. That being said, the Heteroskedasticity shows that there is variance in the residuals, and the Jarque-Bera test shows the residuals are not normally distributed.



Figure 21: MAE

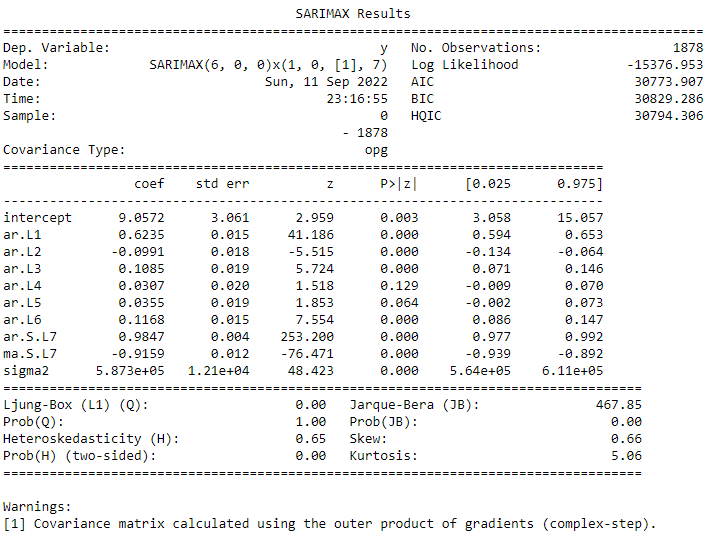


Figure 22: Model Summary

To better understand the ARIMA results, I needed to compare it to other models. The next model I decided to use was created from the time series analysis tool, Prophet. Created by researchers from Meta, profit is an easy-to-use alternative for time series modeling. The models it creates use additive or multiplicative terms similar to ARIMA. One large advantage to profit is that it more easily accounts for multiple seasonality and holiday factors in its models. Figure 23 shows the forecast of the Prophet model. Figure 24 shows the data split into the multiple seasonalities and holidays. The Prophet MAE was found to be 866

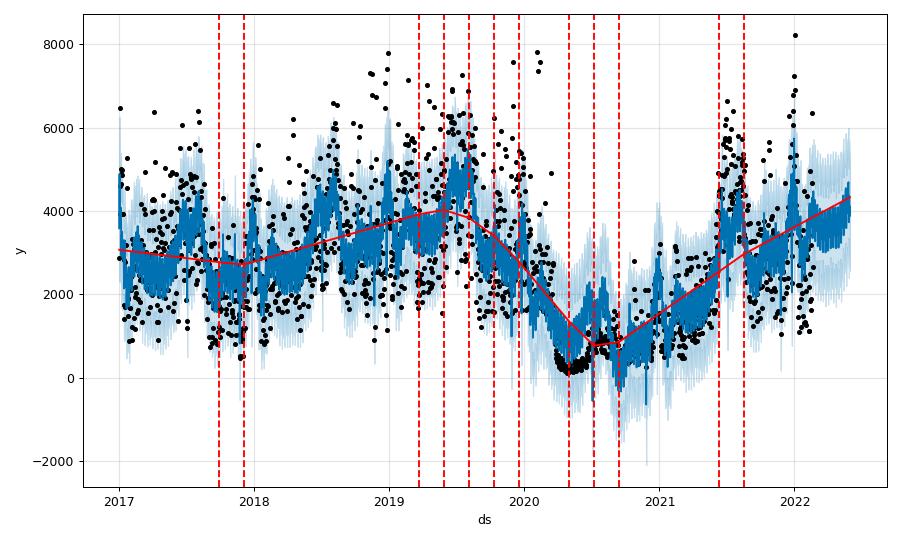


Figure 23: Prophet Prediction

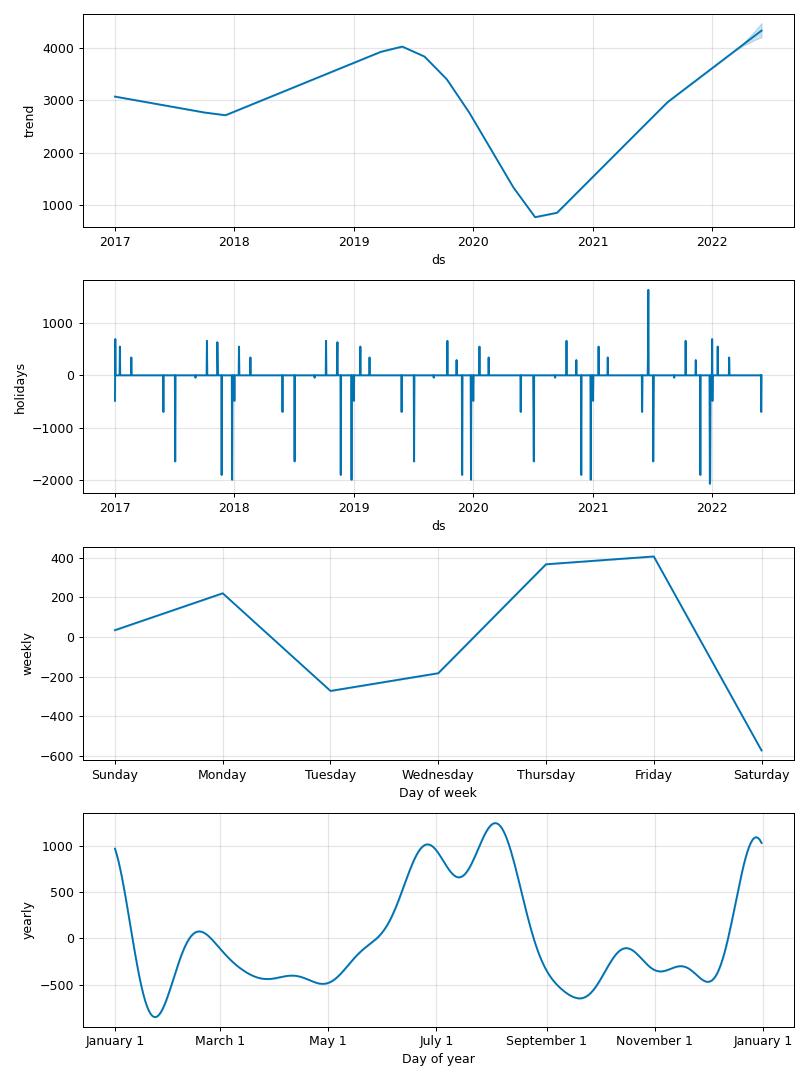


Figure 24: Prophet Components

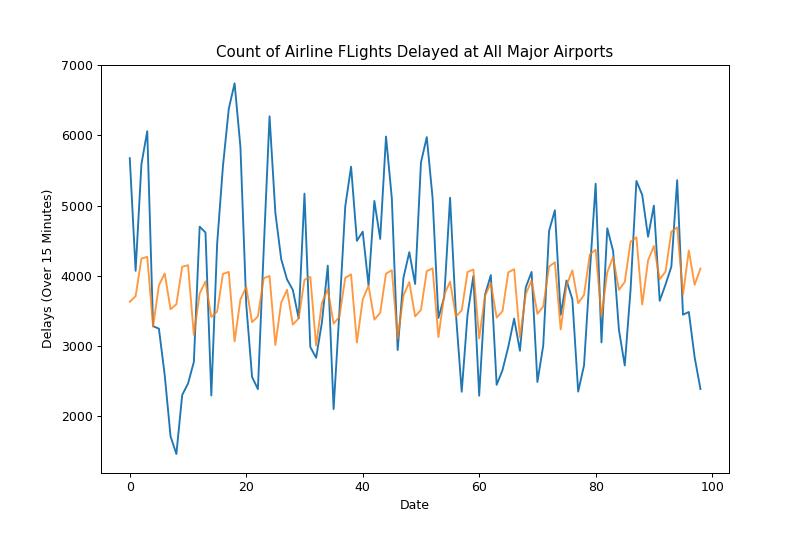


Figure 25: Prophet Forecast vs Actual



Figure 26: Prophet MAE

The last model I used for comparison is a baseline model where all predictions are equal to the median of the training data. Any model MAE would need to be significantly smaller in order for it to be considered a better forecaster than the alternative of just using the median of the data. The Baseline MAE was found to be 1424.



Figure 27: Baseline MAE

**Data Summary and Implications**

On comparing the MAEs one thing of importance to note is that all models were using the same scale. The lowest performing/highest value MAE came from the Baseline model using the training data median value for forecasting at 1424. The next was the ARIMA model with a MAE of 976. The best performing model was the Prophet model with an MAE of 866.

With the results in mind, the recommended course of action would be to disregard the ARIMA model for delay predictions. While it did outperform the baseline, it did not capture all test values within our 95% confidence interval missing 9% of the test predictions. The better MAE alone is not a good enough predictor to reject our original null hypothesis for the ARIMA model that was created. The prophet model created for comparison produces better results than the ARIMA model, with an MAE value of less than 2/3rds the baseline model value. This model better captured the multiple seasonalities and changepoints in the underlying data.

A large limitation to this analysis was hardware related. Even with 16GB of memory available on the computer used for processing, it often became a challenge to balance analysis tasks and available memory without having the program crash. Time became another factor as computing times often increased as model complexity, seasonal length, or parameters validated grew. A more powerful hardware environment could lead to more elaborate and better fitting models that compute in a shorter amount of time.

There are multiple avenues we could pursue to further analyze this particular data set. First, we could attempt improvements in the current ARIMA model through various methods. The data analyzed could be shortened so that the low period during 2020 and 2021 does not confound the model. We could also add in exogenous variables and modify the SARIMA model into a SARIMAX. Another option is to further explore the Prophet model and analyze it for viability of use.

Second, other time series methods may produce better results than the ARIMA or prophet models utilized for this project. We could attempt a multivariate analysis approach to incorporate other independent variables that may allow us to predict delays more accurately. We could use a neural network model with LSTM layers.

Finally, we could turn our analysis towards other features within the dataset. Airline, airport, flight number, and delay category are all features that we could form a number of other delay related research questions around.

**Code**

The code for this project was created using a Jupyter Notebook IDE. The associated .ipynb file containing the code for this project has been included with this report. Below is a copy of the code used for the project.

# # Time Series Analysis of the Airline Delay Data

# ## Data Extraction and Preparation

#Import Libraries

import random

import sys

import numpy as np

import pandas as pd

import gc

import matplotlib.pyplot as plt

from scipy.stats import boxcox

from statsmodels.tsa.stattools import adfuller as adf

from statsmodels.graphics.tsaplots import plot\_acf, plot\_pacf, plot\_predict

from statsmodels.tsa.seasonal import seasonal\_decompose, STL

import itertools

from pmdarima.arima import auto\_arima, ndiffs

from kats.consts import TimeSeriesData

from kats.detectors.seasonality import FFTDetector

from kats.detectors.outlier import OutlierDetector

from prophet import Prophet

from prophet.plot import add\_changepoints\_to\_plot

from prophet.diagnostics import cross\_validation, performance\_metrics

from sklearn.metrics import mean\_absolute\_error

# A line plot of the data helps establish a baseline understanding of the time series data. Put another way, it is the first chance to visually see the data and begin analysing the variance, trend, and seasonality.

#Create Ranges for iterating over .csv files

yearRange = np.arange(2017,2023,1)

monthRange = np.arange(1,13,1)

#Exploratory analysis

exploreDF = pd.read\_csv(r"C:\Users\Nathan\Desktop\College\WGU\11. Capstone\Data\Uncompressed Monthly\Temp\On\_Time\_Reporting\_Carrier\_On\_Time\_Performance\_(1987\_present)\_2017\_1.csv", low\_memory=False)

print(exploreDF.head())

print(exploreDF.shape)

#Select columns for analysis

columnsOnTimeTSA = ["FlightDate", 'ArrDel15']

#Check for Duplicates

tsaDF = exploreDF[columnsOnTimeTSA]

tsaDF.info()

#Check for duplicates. Takes in dataframe and returns the cleaned dataframe with count of duplicates

def duplicateCheck(inputdf):

anyDups = inputdf.duplicated().sum()

dupsDeleted=0

if anyDups > 0:

originalLength = len(inputdf)

inputdf.drop\_duplicates(inplace=True)

newLength = len(inputdf)

dupsDeleted= originalLength-newLength

return(inputdf, dupsDeleted)

#Function to check for missing values. Takes dataframe in and returns the dataframe and count of missing

def missingCheck(inputdf):

anyMissing = inputdf.isnull().sum().sum()

missDeleted=0

if anyMissing > 0:

originalLength = len(inputdf)

inputdf.dropna(inplace=True)

newLength = len(inputdf)

missDeleted= originalLength-newLength

return(inputdf, missDeleted)

#Clean exploratory DataFrames from memory

del exploreDF

del tsaDF

gc.collect()

#Create dictionary of datatypes for easier import

datatypes = {'ActualElapsedTime': 'float64',

'AirTime': 'float64',

'ArrDel15': 'float64',

'ArrDelay': 'float64',

'ArrDelayMinutes': 'float64',

'ArrTime': 'float64',

'ArrivalDelayGroups': 'float64',

'CRSElapsedTime': 'float64',

'CancellationCode': 'object',

'DepDel15': 'float64',

'DepDelay': 'float64',

'DepDelayMinutes': 'float64',

'DepTime': 'float64',

'DepartureDelayGroups': 'float64',

'TaxiIn': 'float64',

'TaxiOut': 'float64',

'WheelsOff': 'float64',

'WheelsOn': 'float64'}

#Loop over year and month ranges. Import the files and clean each one. Output is the combined DataFrame with just the

#values needed for the analysis

count = 0

totalRows = 0

totalDuplicates = 0

totalMissing = 0

for year in yearRange:

for month in monthRange:

newDF = pd.read\_csv(fr"C:\Users\Nathan\Desktop\College\WGU\11. Capstone\Data\Uncompressed Monthly\Temp\On\_Time\_Reporting\_Carrier\_On\_Time\_Performance\_(1987\_present)\_{year}\_{month}.csv", low\_memory=False,

dtype=datatypes)

totalRows += len(newDF)

newDF, anyDups = duplicateCheck(newDF)

totalDuplicates += anyDups

reducedDF = newDF[columnsOnTimeTSA].copy()

shortDF, anyMiss = missingCheck(reducedDF)

totalMissing += anyMiss

shortDF['ArrDel15'] = shortDF['ArrDel15'].astype('int')

if count == 0:

superDF = shortDF

else:

superDF = superDF.append(shortDF)

count+=1

print(f"Loaded {year}, {month}. Found and deleted {anyDups} duplicate, {anyMiss} missing rows.")

if (year == 2022 and month == 5):

break

print(superDF.head())

#Clean unused Data

del newDF

del reducedDF

del shortDF

gc.collect()

#Print totals from the import and cleaning process

print(f"Totals rows process: {totalRows}. Total duplicates: {totalDuplicates}. Total missing: {totalMissing}.")

print(f"Precent missing: {(totalMissing/totalRows)\*100}%")

print(f"Final number of rows is {len(superDF)}.")

superDF.to\_csv(r"C:\Users\Nathan\Desktop\College\WGU\11. Capstone\Upload\TEMPDONOTUPLOAD.csv")

#Print the shape

print(f"The shape of the DataFrame is {superDF.shape}")

#Change data index to timeseries

superDF['FlightDate'] = pd.to\_datetime(superDF['FlightDate'])

gDF = superDF.copy().groupby(['FlightDate'], axis=0).sum()

gDF.index.freq = pd.tseries.frequencies.to\_offset("D")

#View the new data shape and first five rows

print(gDF.shape)

print(gDF.head())

#Summary statistics

print(gDF.describe())

#Create a series

gDFDelay = gDF['ArrDel15']

print(gDFDelay.head())

#Plot the line graph timeseries data for EDA

plt.rcParams.update({'figure.figsize':(9,6), 'figure.dpi':100})

gDF['ArrDel15'].plot(legend=False)

plt.ylabel('Delays (Over 15 Minutes)')

plt.xlabel('Date')

plt.title('Count of Airline FLights Delayed at All Major Airports')

plt.annotate('Drop around the start of Covid-19 Pandemic', xy=(.54,.16), xycoords='figure fraction', xytext=(.14, .15),

arrowprops=dict(facecolor='black', shrink=0.05))

plt.savefig(r'C:\Users\Nathan\Desktop\College\WGU\11. Capstone\Figures\LineplotEDA1.jpeg')

plt.show()

#Create percent data for comparison

gDFprcnt = superDF.groupby(['FlightDate'], axis=0).agg({'ArrDel15': ['sum','count']})

gDFprcnt.index.freq = pd.tseries.frequencies.to\_offset("D")

gDFprcnt['PrcntDel'] = round((gDFprcnt['ArrDel15', 'sum'] / gDFprcnt['ArrDel15', 'count']) \* 100, 2)

gDFprcnt.drop(['ArrDel15'], axis=1, inplace=True)

print(gDFprcnt.head())

#Plot the line graph of the percent data and delay data for EDA

plt.rcParams.update({'figure.figsize':(9,6), 'figure.dpi':100})

fig, ax1 = plt.subplots()

ax1.set\_xlabel('Date')

ax1.set\_ylabel('Delays (Over 15 Minutes)', color='b')

ax1.plot(gDF.index, (gDF['ArrDel15']/110), label='Delay Count', color='b')

ax1.tick\_params(axis='y', labelcolor='b')

ax2 = ax1.twinx()

ax2.set\_ylabel('Percent of Delayed FLights', color='orange') # we already handled the x-label with ax1

ax2.plot(gDFprcnt.index, gDFprcnt['PrcntDel'], alpha=0.6, label='Percent of Flights Delayed', color='orange')

ax2.tick\_params(axis='y', labelcolor='orange')

fig.tight\_layout() # otherwise the right y-label is slightly clipped

#plt.plot(gDF.index, (gDF['ArrDel15']/110), label='Delay Count')

#plt.plot(gDFprcnt.index, gDFprcnt['PrcntDel'], alpha=0.8, label='Percent of Flights Delayed')

plt.title('Count of Delays Vs Percentage of FLights Delayed')

plt.savefig(r'C:\Users\Nathan\Desktop\College\WGU\11. Capstone\Figures\TwinLineplotEDA.jpeg')

plt.show()

gc.collect()

#Ensure time series is continous

ContinousIndChk = gDFDelay.index.to\_series().diff().ne('1 days').cumsum().sub(1).sum()

print(f"There is/are {ContinousIndChk} non-continous days.")

DataLength = len(gDFDelay)

print(f"There are {DataLength} days worth of data.")

proDF = pd.DataFrame(data=gDFDelay)

proDF['y'] = proDF['ArrDel15']

proDF['ds'] = gDF.index

proDF = proDF[['ds', 'y']]

proDF.reset\_index(drop=True, inplace=True)

print(proDF.head())

#Create the train/test split for the Delay Count Data

#Export Train and Test data to csv.

TrainTestSplitProp = 0.05

TestSize = int(round(DataLength\*TrainTestSplitProp, 0))

Traindf = gDFDelay.iloc[:(DataLength-TestSize)]

Testdf = gDFDelay.iloc[(DataLength-TestSize):]

print(f"Train size is {len(Traindf)}.")

print(f"Test size is {len(Testdf)}.")

Traindf.to\_csv(r"C:\Users\Nathan\Desktop\College\WGU\11. Capstone\Upload\TrainDataset.csv")

Testdf.to\_csv(r"C:\Users\Nathan\Desktop\College\WGU\11. Capstone\Upload\TestDataset.csv")

#Create the train/test split for the data to be used in the Prophet models. Since

#Export Train and Test data to csv.

Trainprodf = proDF.iloc[:(DataLength-TestSize)]

Testprodf = proDF.iloc[(DataLength-TestSize):]

print(f"Train size is {len(Traindf)}.")

print(f"Test size is {len(Testdf)}.")

Trainprodf.to\_csv(r"C:\Users\Nathan\Desktop\College\WGU\11. Capstone\Upload\TrainProDataset.csv")

Testprodf.to\_csv(r"C:\Users\Nathan\Desktop\College\WGU\11. Capstone\Upload\TestProDataset.csv")

# ## Analysis

# The following section is pre model analysis, model preperation, and model creation

#Check stationarity using ADF test

adfresult1 = adf(Traindf)

print(f"Initial augmented Dickey-Fuller Test P-value is {adfresult1[1]}")

#Check for stationarity

n\_diffs\_aic = ndiffs(Traindf, max\_d=5, test='adf')

n\_diffs\_ffts = ndiffs(Traindf, max\_d=5, test='kpss')

print(f"Number of differences needed according to AIC is {n\_diffs\_aic}")

print(f"Number of differences needed according to AIC is {n\_diffs\_aic}")

#Use autocorrelation to check for seasonality

seasonalCheck = Traindf - Traindf.rolling(220).mean()

seasonalCheck = seasonalCheck.dropna()

#Plot the autocorrelation

plot\_acf(seasonalCheck, alpha=0.05, lags=700)

plt.ylabel('Pearson Correlation')

plt.xlabel('Days in term')

plt.savefig(r'C:\Users\Nathan\Desktop\College\WGU\11. Capstone\Figures\SeasonalACF.jpeg')

plt.show()

#Plot the partial autocorrelation

plt.rcParams.update({'figure.figsize':(9,6), 'figure.dpi':80})

plot\_pacf(seasonalCheck, alpha=0.05, lags=60)

plt.ylabel('Correlation')

plt.xlabel('Days')

plt.savefig(r'C:\Users\Nathan\Desktop\College\WGU\11. Capstone\Figures\SeasonalPACF.jpeg')

plt.show()

#Use Fast Forier Transformation to detect seasonality periods

gDFts = TimeSeriesData(Trainprodf, time\_col\_name='ds')

fft\_detector = FFTDetector(gDFts)

fft\_detector.detector()

# Detect outliers

ts\_outlierDetection = OutlierDetector(gDFts, "additive")

ts\_outlierDetection.detector()

# Print outliers

print(f"All outliers: {ts\_outlierDetection.outliers}")

# Graphing the seasonal decomposition help us better understand the underlying components of the series. It more clearly highlights the trend as well as the residuals.

#Seasonal Decomposition

plt.rcParams.update({'figure.figsize':(9,5), 'figure.dpi':100})

result = seasonal\_decompose(Traindf, model='additive', period=7)

result.plot()

plt.savefig(r'C:\Users\Nathan\Desktop\College\WGU\11. Capstone\Figures\SeasonalDecompose.jpeg')

plt.show()

#Identify the overall trend in the data for the given 7 day period

plt.rcParams.update({'figure.figsize':(7,4.5), 'figure.dpi':120})

result.trend.plot(kind='line')

plt.ylabel('Delays (Over 15 Minutes)')

plt.xlabel('Date')

plt.title('Decomposed Delay Trend')

plt.savefig(r'C:\Users\Nathan\Desktop\College\WGU\11. Capstone\Figures\Trend.jpeg')

plt.show()

#Confirm lack of trend in rediuals of differenced data. No trend, but variance difference remains

result.resid.plot(kind='line')

plt.ylabel('Residual Value')

plt.xlabel('Day')

plt.title('Residual Plot')

plt.savefig(r'C:\Users\Nathan\Desktop\College\WGU\11. Capstone\Figures\ResidPlt.jpeg')

plt.show()

#Check distribution of residuals for gaussian distribution

result.resid.plot(kind='hist')

plt.ylabel('Frequency')

plt.xlabel('Residual Value')

plt.title('Residual Histogram (Distribution Check)')

plt.savefig(r'C:\Users\Nathan\Desktop\College\WGU\11. Capstone\Figures\DistCheck.jpeg')

plt.show()

#Free up memory

gc.collect()

#Create model using auto arima

autoar\_model = auto\_arima(Traindf,

test='adf',

max\_p=11, max\_q=2,

m=7,

seasonal=True,

trace=True,

error\_action='ignore',

suppress\_warnings=True,

stepwise=True, parallel=True, approximation=True)

#Free up memory

gc.collect()

# ## Model Performance Analysis

# The following are the results of the model creation. Also included is the prophet model and baseline model creations

#Plot Residuals from Auto-Arima Module

plt.rcParams.update({'figure.dpi':90})

autoar\_model.plot\_diagnostics(figsize=(10,11))

plt.tight\_layout

plt.subplots\_adjust(hspace=0.3)

plt.savefig(r'C:\Users\Nathan\Desktop\College\WGU\11. Capstone\Figures\ModelPlotDiagnostics.jpeg')

plt.show()

#Predict in sample values

AAInPredicted, InConfidenceInt = autoar\_model.predict\_in\_sample(return\_conf\_int=True, alpha=0.05)

AAInPredictedDF = pd.DataFrame(AAInPredicted, index=Traindf.index)

InConfidenceIntDF = pd.DataFrame(InConfidenceInt)

#Plot in-sample predicted values and actual values

plt.figure(figsize=(12,5), dpi=100)

plt.plot(Traindf, label='Training')

plt.plot(AAInPredictedDF, label='Predicted')

plt.fill\_between(AAInPredictedDF.index[1:], InConfidenceIntDF.iloc[1:,0], InConfidenceIntDF.iloc[1:,1],

color='k', alpha=0.1)

plt.title('In-Sample Predictions vs Actual')

plt.legend(loc='upper left', fontsize=9)

plt.annotate('Gray Shows 95% Confidence Interval', xy=(.55,.2), xycoords='figure fraction', xytext=(.27, .15),

arrowprops=dict(facecolor='black', shrink=0.05))

plt.savefig(r'C:\Users\Nathan\Desktop\College\WGU\11. Capstone\Figures\InSample.jpeg')

plt.show()

#Forecast the arima model

ForecastDaysPastCurrent = 0

CombinedForecast = len(Testdf) + ForecastDaysPastCurrent

OneDayPastEnd = Testdf.index[-1] + pd.DateOffset(day=2)

AddFutureDaysToIndex = pd.Series(pd.date\_range(OneDayPastEnd, freq="D", periods=ForecastDaysPastCurrent))

CombinedIndex = pd.concat([pd.Series(Testdf.index), (AddFutureDaysToIndex)])

AAPredict, ConfInt = autoar\_model.predict(n\_periods=CombinedForecast, return\_conf\_int=True, alpha=0.05)

AAPredictDF = pd.DataFrame(AAPredict, index=CombinedIndex)

ConfIntDF = pd.DataFrame(ConfInt)

#Graph the Forecast, confidence intervals, and test data for reference

plt.figure(figsize=(12,5), dpi=100)

plt.plot(Traindf, label='Training')

plt.plot(Testdf, label='Actual')

plt.plot(AAPredictDF, label='Predicted')

#plt.plot(ForCas.predicted\_mean, label='Forecast')

plt.fill\_between(AAPredictDF.index, ConfIntDF[0], ConfIntDF[1],

color='k', alpha=.05)

plt.title('Forecast vs Actual')

plt.savefig(r'C:\Users\Nathan\Desktop\College\WGU\11. Capstone\Figures\Forecast.jpeg')

plt.show()

countOutOfInterval=0

counter=0

for x in Testdf:

if ((x < ConfIntDF.iloc[counter,0]) or (x > ConfIntDF.iloc[counter,1])):

countOutOfInterval+=1

counter+=1

print(f"Number of values out of bounds: {countOutOfInterval}")

print(f"The percent of values outside of the prediction interval is: {round(countOutOfInterval/len(Testdf)\*100, 2)}%")

#Print the Mean Absolute Error

MAE = mean\_absolute\_error(Testdf, AAPredictDF)

print(f"The Mean Absolute Error of the model is: {MAE}")

#Print Model Summary

print(autoar\_model.summary())

#Save the predictions to a CSV

output = AAPredictDF

output.to\_csv(r"C:\Users\Nathan\Desktop\College\WGU\11. Capstone\Upload\PredictedOutput.csv")

# ## Meta Prophet Model

#Create the prophet model

Proph\_m1 = Prophet(changepoint\_range=0.9)

Proph\_m1.add\_country\_holidays(country\_name='US')

Proph\_m1.fit(Trainprodf)

#Create future distribution

future = Proph\_m1.make\_future\_dataframe(periods=len(Testprodf))

future.tail()

# Create forecast based on prophet model

forecast = Proph\_m1.predict(future)

print(forecast[['ds', 'yhat', 'yhat\_lower', 'yhat\_upper']].tail())

#Plot the forecast

fig1 = Proph\_m1.plot(forecast)

a = add\_changepoints\_to\_plot(fig1.gca(), Proph\_m1, forecast)

plt.savefig(r'C:\Users\Nathan\Desktop\College\WGU\11. Capstone\Figures\Prophetforecast.jpg')

#Plot components

fig2 = Proph\_m1.plot\_components(forecast)

plt.savefig(r'C:\Users\Nathan\Desktop\College\WGU\11. Capstone\Figures\Prophetcomponents.jpeg')

#Create MAE value

MergedDF = pd.merge(Testprodf, forecast[['ds','yhat\_lower','yhat\_upper','yhat']],on='ds')

MergedDF = MergedDF[['ds','yhat\_lower','yhat\_upper','yhat','y']]

y\_true = MergedDF['y'].values

y\_pred = MergedDF['yhat'].values

ProphMAE1 = mean\_absolute\_error(y\_true, y\_pred)

print(f"Prophet MAE is: {ProphMAE1}")

#Model predictions vs actual

plt.rcParams.update({'figure.figsize':(9,6)})

plt.plot(y\_true)

plt.plot(y\_pred, alpha=0.8)

plt.ylabel('Delays (Over 15 Minutes)')

plt.xlabel('Date')

plt.title('Count of Airline FLights Delayed at All Major Airports')

plt.savefig(r'C:\Users\Nathan\Desktop\College\WGU\11. Capstone\Figures\ProphetPredvsActual.jpeg')

plt.show()

Basemean = pd.DataFrame(np.full((len(Testdf), 1), Traindf.median(), dtype='int'))

BaselineMAE = mean\_absolute\_error(Testdf, Basemean)

print(f"Baseline MAE is: {BaselineMAE}")

**References**

*Airline On-Time Performance Data*. (2022, May). BTS. Retrieved August 14, 2022, from https://www.transtats.bts.gov/Tables.asp?QO\_VQ=EFD&QO\_anzr=Nv4yv0r%FDb0-gvzr%FDcr4s14zn0pr%FDQn6n&QO\_fu146\_anzr=b0-gvzr

Caswell et al. (2021, March 31). matplotlib/matplotlib: REL: v3.4.1 (Version v3.4.1). Zenodo. http://doi.org/10.5281/zenodo.4649959

Harris et al. (2020). Array programming with NumPy. *Natur*e, *585,* 357–362. DOI: 0.1038/s41586-020-2649-2

Jiang, X. (2022). *Kats* (Version 0.0.0). https://github.com/facebookresearch/Kats

Josephs, L. (2022, July 1). More than 12,000 flights delayed, hundred canceled during busy July Fourth weekend. CNBC. Retrieved August 14, 2022, from <https://www.cnbc.com/2022/07/01/airline-travel-of-july-weekend-puts-airlines-and-travelers-to-the-test-after-difficult-spring.html>

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., … Grisel, O. (2011). Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research, 12*(Oct), 2825–2830.

Reback et al. (2021, April 12). pandas-dev/pandas: Pandas 1.2.4 (Version v1.2.4). Zenodo. <http://doi.org/10.5281/zenodo.4681666>

Seabold, Skipper, and Josef Perktold. (2010). statsmodels: Econometric and statistical modeling with python. *Proceedings of the 9th Python in Science Conference*.

Smith et al. (2017). pmdarima: ARIMA estimators for Python. Retrieved June 20, 2022, from <http://www.alkaline-ml.com/pmdarima>.

Taylor, S., Letham B. (2017). *Forecasting at scale*. PeerJ Preprints 5:e3190v2 https://doi.org/10.7287/peerj.preprints.3190v2