Assessing the Accuracy of Time-Series Forecasting Models on Meteorological Datasets

Forecast Focus

Accurate Weather Forecasting

Student 1 Name (ID): Kevin Patrick Boyle (19731615)

Student 2 Name (ID): Mark Reid (19414892)

Supervisor: Mark Roantree

Date Completed: 5 / 5 2023

1. Introduction:

The accuracy of forecasting models is crucial for any business that relies on predicting future trends. In this technical guide, we present our system “Forecast Focus’ that was designed to test the accuracy of five different time-series forecasting models on 10 different datasets. The models used in our system include *ARIMA*, *SARIMAX*, *XGBoost*, *LSTM*, and *Holt Winters Exponential Smoothing*.

The datasets used were collected from several different areas in the Subotica region in Serbia spanning over 9 years from 2013 to 2021. Our system provides an easy-to-use interface for users to test the accuracy of these models on their own datasets and make informed decisions based on the results. The primary goal of our system is to help businesses make accurate predictions and understand which type of model is most appropriate based on the characteristics of the data collected. In this guide, we will walk you through the steps required to use our system and analyse the results obtained from testing the different models on your own datasets.

To gain a deeper understanding of each forecasting model used in our system, we conducted extensive research using online resources such as Google Scholar. By leveraging these services, we were able to find detailed information regarding each model's underlying principles, techniques, and limitations.

1.2Glossary:

Term:	Definition:
Time-series:	A set of data points collected over time.
Autoregression:	A model that uses a time-series' past values to predict future values.
ARIMA:	Autoregressive integrated moving average, a popular time-series forecasting model.
SARIMAX:	Seasonal autoregressive integrated moving average with exogenous variables, a time-series forecasting model that takes into account seasonal patterns and external factors.
XGBoost:	Extreme gradient boosting, a machine learning algorithm used for time-series forecasting.
LSTM:	Long short-term memory, a recurrent neural network architecture used for time-series forecasting.
Holt Winters Time-Series Forecasting:	A time-series forecasting method that takes into account seasonality, trend, and level.
Hyperparameters:	Parameters that are set before the model training and affect the model's performance and behaviour.
Forecasting:	The process of making predictions about future events based on historical data.
Exogenous variables:	Variables that are external to the time-series being forecasted and can affect the

	forecasting accuracy.
Scikit-learn	Python machine learning library with various algorithms for classification, regression, and clustering.
TensorFlow	an open-source library for deep learning, used in industry and academia for AI applications.
Decision Trees	A tree-shaped model used to make predictions based on decisions made at each branch point.
Gradient Boosting	A technique which combines multiple weak learning models into a single strong model

1.3 Motivation:

Our project aims to address a crucial issue in the field of meteorology, which is the accurate prediction of missing values in weather datasets. Inaccurate or incomplete data can have serious consequences, particularly in weather forecasting, where even small errors can have a significant impact.

There are several reasons why gaps may occur in weather forecasting data, resulting in incomplete or missing information. One common cause is maintenance being conducted on the weather station equipment, which on average occurs annually. During this time, the equipment may be offline, leading to missing data. Additionally, extreme weather conditions, such as hurricanes or severe storms, can disrupt power to the weather stations or cause damage to the equipment, resulting in gaps in the data. Missing values can also occur randomly which can be due to technical malfunctions, software errors, or communication problems. Regardless of the cause, missing data can have significant implications for decision-making and forecasting accuracy.

By testing the accuracy of various time-series forecasting models on 10 different meteorological datasets, we hope to provide valuable insights into which model is best suited for filling in missing values. Our motivation for this project was that it provides us with the opportunity to enhance our skills in machine learning and data analysis, which are in high demand in today's job market.

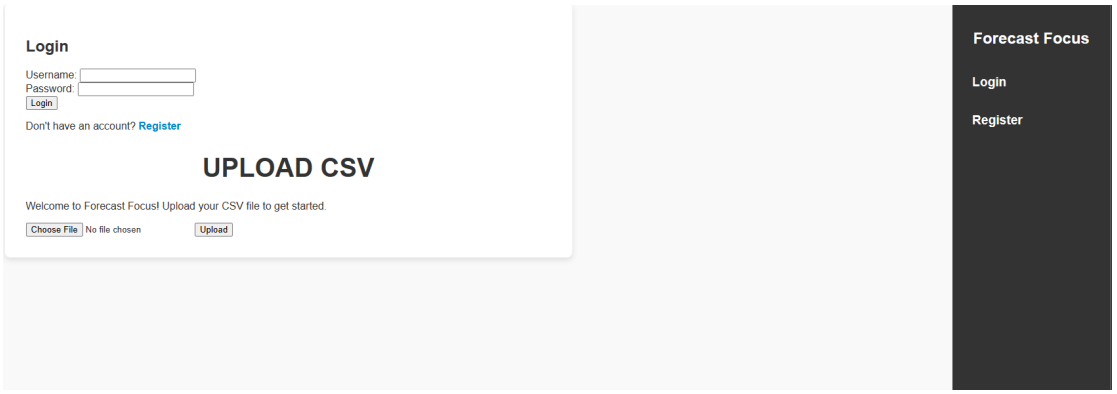
2. System Architecture:

3. Design:

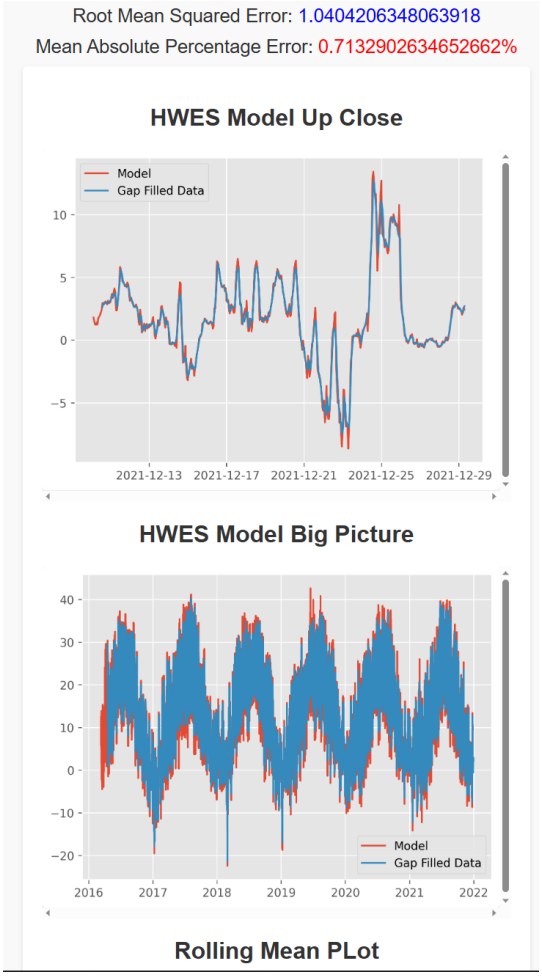
Users Database:

We implemented a user profile database using SQLAlchemy Database. This allowed users to create their own account and login to the web application, where they could upload their own time-series datasets and choose from the five available forecasting models: *ARIMA*, *SARIMAX*, *XGBoost*, *LSTM*, and *Holt Winters Exponential Smoothing*. Once the users had uploaded their time-series dataset to the web application and selected their desired forecasting model, they had the option to insert gaps into their dataset and forecast the missing values. Furthermore, users could view a history of their previously uploaded datasets and the associated forecasting results, which allowed for easy comparison and analysis. The aim of the user profile database was to add a level of personalization and customization to our web application. Overall, the implementation of the user profile database was a key feature of our web application, and it significantly improved the user experience and functionality of our time-series forecasting system.

Front End:



User Login Page.



Results page for Holt Winters Forecasting

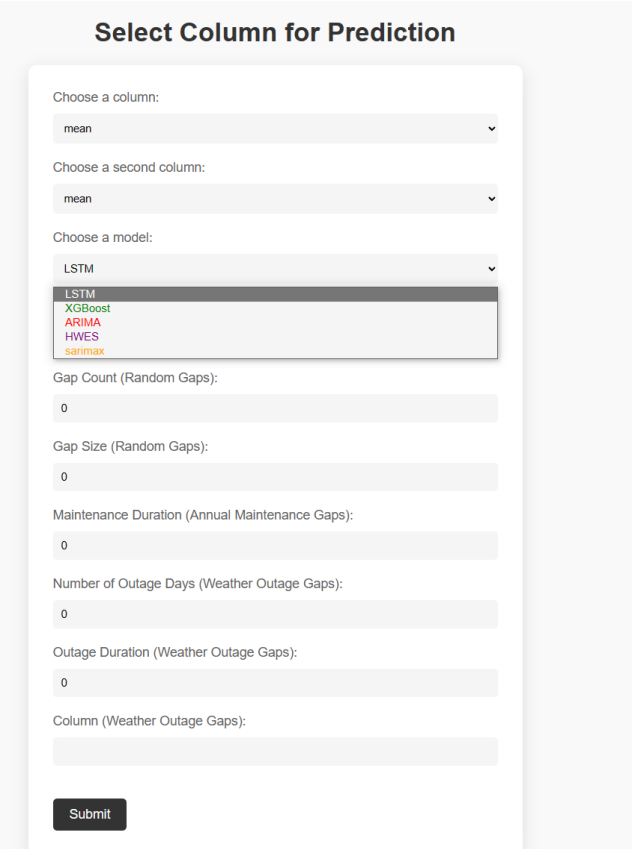
user123's Profile

Previously Uploaded Files

- DAD.csv - 2023-05-06 21:21:41
- E94.csv - 2023-05-06 21:22:05

Logout

User Upload History



Page to Upload Dataset, Select model and Gap-Insertion Method

The front-end design of our time-series forecasting web application was designed to ensure a user-friendly and intuitive experience. The website features both a registration page for new users to create an account and a login page for existing users to access the system. The registration page allows users to enter their details and create a new user profile, login credentials.

Once logged in, the application allows users to upload time-series datasets in the form of a CSV file, where users can select their dataset, choose the forecasting model, and insert gaps into their time-series dataset as needed. This feature allows users to simulate a variety of scenarios, including gaps caused by extreme weather conditions, annual maintenance, or random gaps. Users can specify the quantity of gaps and the granularity of the gaps. The quantity and granularity of the gaps specified by the user can have a significant impact on the accuracy of the forecasting results. The application also includes a page where users can view a history of their previously uploaded datasets, specifically for their user account. Finally, the application includes a page where forecasting results are displayed after gap filling, providing users with valuable insights into the accuracy of their chosen forecasting model.

Overall, we believe the front-end design of our time-series forecasting web application is user-friendly, and optimised for ease of use, allowing users to focus on the task of forecasting with minimal distractions.

4.Research:

4.1 Time Series Forecasting:

The core focus of our project is time-series forecasting, which involves predicting future values based on historical data. In order to ensure the accuracy and reliability of our time-series forecasting system, we conducted extensive research into several time-series forecasting models. This research involved an in-depth analysis of each model's strengths and weaknesses, as well as a review of their underlying algorithms and methods.

We chose to implement the five time-series forecasting models (*ARIMA*, *SARIMAX*, *XGBoost*, *LSTM*, and *Holt Winters Exponential Smoothing*) based on their ability to generate accurate and reliable forecasting results for a wide range of time-series datasets. Additionally, each of these models uses a unique approach to generate forecasting results, providing users with a variety of options to choose from based on their specific needs and requirements.

4.2 Time Series Models: ARIMA & SARIMA:

ARIMA is a statistical model that uses past values and errors to predict future values. It is particularly useful for modelling stationary time series data. SARIMAX is an extension of ARIMA that includes seasonal components and external variables, making it more suitable for modelling time series data with seasonality and external factors. We discovered that the time-series data we were using had seasonal patterns that made it hard to predict future values with certain forecasting models like ARIMA. ARIMA works well for data that doesn't change much over time, but it performed worse than others due to the seasonal changes in our data.

To address this issue, we then decided to use the SARIMAX time-series forecasting model, which was better suited to handle time-series data with seasonal components. By including exogenous variables into the forecasting model, SARIMAX was able to provide a greater accuracy and precision in predicting the gap values for each of our datasets. We tested each column in the dataset for its correlation with the others. This helped us identify which variables would be best suited for use as exogenous variables in our forecasting models. By selecting variables that were highly correlated with the target variable, we were able to improve the accuracy and precision of our forecasts.

4.3 Time Series Models: XGBoost:

XGBoost, on the other hand, is a machine learning algorithm that uses decision trees and gradient boosting to make predictions. Unlike ARIMA and SARIMAX, XGBoost is not a time series-specific model, but it can still be used for time series forecasting. The algorithm works by building a combination of decision trees each predicting the target variable based on a set of inputs.

The algorithm then iteratively improves the combination by adding new decision trees that correct the errors of the previous models. This process continues until the algorithm converges on a set of models that minimise the overall error. We found XGBoost’s biggest advantage was its ability to handle missing values in the input data.

4.4 Time Series Models: LSTM (Long Short-Term Memory)

LSTM is a type of neural network that is particularly useful for capturing long-term dependencies in time series data. Unlike other neural network architectures, LSTM networks have the ability to capture long-term dependencies in the data. LSTM networks are well-suited to time series data that contains complex patterns, like trends and seasonal changes. This made them a great fit for the datasets we were working with, since they exhibited these types of patterns.

4.5 Time Series Models: Holt Winters Exponential Smoothing

Holt Winters Exponential Smoothing is particularly useful for modelling and forecasting data that has a clear pattern of repeating highs and lows over time, which was especially useful for the Mean, Minimum and Maximum Precipitation columns in our dataset. The algorithm uses an exponential smoothing technique to model the underlying trend, seasonality, and level of the time series data. By incorporating these factors into the forecasting model, Holt Winters Exponential Smoothing is able to provide accurate and reliable predictions for future values in the time series.

5. Implementation:

5.1 Model: LSTM

The implementation of the LSTM Model imports necessary libraries including tensorflow, pandas, numpy, and matplotlib. The 'lstm_forecast' function takes three parameters and loads and preprocesses data from the files. It then defines a function to split data into input and output sequences for LSTM training. The function builds and compiles an LSTM model with a ModelCheckpoint callback to save the best model based on validation loss. The trained model is used to make predictions on the combined data, and the results are plotted using matplotlib. Finally, the function returns the root mean squared error and validation root mean squared error values, along with base64 encoded strings of the two plots generated during the function call.

```
X_training, Y_training = X[:60000], y[:60000]
X_Validation, Y_Validation = X[60000:65000], y[60000:65000]
X_testing, y_testing = X[65000:], y[65000:]

model = Sequential([
    InputLayer((5, 1)),
    LSTM(64),
    Dense(8, activation='relu'),
    Dense(1, activation='linear')
])

model.summary()

checkpoint = ModelCheckpoint('model/', save_best_only=True)

model.compile(loss=MeanSquaredError(), optimizer=Adam(learning_rate=0.0001), metrics=[RootMeanSquaredError()])
history = model.fit(X_training, Y_training, validation_data=(X_Validation, Y_Validation), epochs=10, callbacks=[checkpoint])

rmse = history.history['root_mean_squared_error'][-1]
vrmse = history.history['val_root_mean_squared_error'][-1]
```

5.2 Model: XGBoost

The implementation of the XGboost model imports necessary libraries including pandas, numpy, matplotlib, seaborn, and XGBoost. The 'xgboost_forecast' function takes three parameters and loads and preprocesses data from the files. It splits the data into training and testing sets based on the date, extracts time series elements from the data, and defines time series elements to use for prediction. The function initialises an XGBoost model with specified hyperparameters and trains the model using the training set. It evaluates the model on the training and testing sets and makes predictions on the testing set. The function plots the actual data with gaps and the predicted values using matplotlib and saves it as a base64 encoded string. Finally, the function returns the trained XGBoost model, the plot, and evaluation results.

```
x_training = train[elements]
y_training = train[column_name]
x_testing = test[elements]
y_testing = test[column_name]

# This initialises XGBoost model with parameters
reg = xgb.XGBRegressor(base_score=0.5, booster='gbtree',
                       n_estimators=1000,
                       early_stopping_rounds=50,
                       objective='reg:linear',
                       max_depth=3,
                       learning_rate=0.1)


# This trains the XGBoost model using the training set
reg.fit(x_training, y_training,
        eval_set=[(x_training, y_training), (x_testing, y_testing)],# This evaluates the model on training and testing sets
        verbose=100)
# This evaluates the model on training and testing sets
```

5.3 Model: HWES (Holt-Winters Exponential Smoothing)

The 'hwes_forecast' function takes three parameters and loads and preprocesses data from the files. It creates a Holt-Winters Exponential Smoothing model, fits the model to the data, and forecasts the values for the entire dataset. The function calculates the root mean squared error and mean absolute error of the model. The function plots the forecasted values and the actual data with gaps and saves it as a base64 encoded string. It also plots the rolling average against the real data in the dataset and saves it as a base64 encoded string. Finally, the function returns the three plots, the root mean squared error, and mean absolute error.

```
# Create a Holt-Winters Exponential Smoothing model
model = ExponentialSmoothing(data, seasonal_periods=12, trend='add', seasonal='add', damped_trend=True)

# Fit the model to your data
model_fit = model.fit()

# Forecast the values for the entire dataset
forecast = model_fit.fittedvalues

# Calculate the RMSE and MAPE
rmse = np.sqrt(mean_squared_error(data, forecast))
mae = mean_absolute_error(data, forecast)


plt.plot(forecast[50:500], label='Model')
plt.plot(df_gaps[column_name][50:500], label='Gap Filled Data')
plt.legend()
```

5.4 Model: Sarimax

The sarima_forecast function takes in a dataset and two column names, column1 and column2. It reads the dataset and drops any fully null columns and rows, and then reads another dataset (df_gaps) and sets its datetime column as the index.

The keep_two_columns function takes the dataset and the two column names and returns only those two columns.

The split_dataset function splits the dataset into 70% training data and 30% temporary data (test + validation), and then splits the temporary data into 50% test data and 50% validation data (15% of the original dataset each).

The sarimax_forecast function takes the training and test data and the two column names, and fits a SARIMAX model to the training data using the first column as the endogenous variable and the second column as the exogenous variable. It then forecasts the values for the test data and plots the historical precipitation, future precipitation prediction, and actual precipitation data for the test data. The function returns a base64 encoded PNG image of the plot.

Finally, the sarima_forecast function calls the keep_two_columns and split_dataset functions, and then calls sarimax_forecast with the resulting training and test data and column names. It returns the resulting plot as a base64 encoded PNG image.

```
def sarimax_forecast(train_data, test_data, columns):
    model = SARIMAX(
        endog=train_data[columns[0]],
        exog=train_data[columns[1]],
        order=(3, 0, 1),
        trend=(0, 0),
        seasonal_order=(0, 1, 0, 12)
    ).fit()

    forecast = model.forecast(steps=test_data.shape[0], exog=test_data[columns[1]])

    start_date = train_data.index.min()
    end_date = test_data.index.max()
```

5.5 Model: Arima

This code defines a function called arima_forecast that takes three arguments: file, file_gaps, and column_name. It first reads two CSV files (file and file_gaps) into pandas data frames and sets the Datetime column as the index. It then performs some data analysis, including computing rolling means and standard deviations, checking for stationarity using the augmented Dickey-Fuller (ADF) test, and plotting autocorrelation and partial autocorrelation functions.

Next, the function fits an ARIMA model to the time series data using the ARIMA class from the statsmodels.tsa.arima.model module. It then makes predictions on the training data and computes mean squared error and mean absolute error metrics. The function returns these metrics as strings, as well as three plots generated using Matplotlib: the original data with rolling mean and standard deviation, the ARIMA model's predictions, and a lag plot. These plots are first saved as in-memory PNG images using base64 encoding, which are then returned as strings.

```
from sklearn.metrics import mean_squared_error, mean_absolute_error
mse = mean_squared_error(train, predictions)
mae = mean_absolute_error(train, predictions)

resultsone = (f'Test MSE: {mse}')
resultstwo = (f'Test MSE: {mae}')


n = int(len(train) * 0.005)
plt.plot(predictions, color='blue')              #MAIN GRAPH
plt.plot(df_gaps[column_name])
plot2 = io.BytesIO()  # here we package the plot as a png image using base64 io, so it can be returned to our flask app and html later
plt.savefig(plot2, format='png', dpi=300, bbox_inches='tight')
plot2.seek(0)
plot2 = base64.b64encode(plot2.getvalue()).decode('utf-8')
```

5.6 Data Processing:

The provided code below is what we used mostly to process the given CSV Data file. IT reads in the CSV file using pandas and assigns it to a dataframe named 'df'. It then drops any columns that contain all null values and any rows that contain null values. It sets the 'Datetime' column as the index and converts it to a datetime object using the pd.to_datetime() method. Similarly, it reads in another CSV file, assigns it to a dataframe named 'df_gaps', and sets the 'Datetime' column as the index after converting it to a datetime object. The format of the datetime in the 'df_gaps' file is explicitly specified as '%Y-%m-%d %H:%M:%S'.

```
df = pd.read_csv(file_path, sep=',', skiprows=1, encoding='latin1')
df = df.dropna(axis='columns', how='all')
df = df.dropna()
df = df.set_index('Datetime')
df.index = pd.to_datetime(df.index)

df_gaps = pd.read_csv(file_gaps, sep=',', encoding='latin1')
df_gaps['Datetime'] = pd.to_datetime(df_gaps['Datetime'], format='%Y-%m-%d %H:%M:%S')
df_gaps = df_gaps.set_index('Datetime')
```

5.7 Front End(flask) :

Even with backend as the key focus of this project, we decided to implement a simple and user-friendly front end interface using Flask framework.

The front-end code begins by importing several modules such as Flask, os, Pandas, SQLAlchemy, and others. Flask is a web application framework that is used to build the front end of the application. Pandas is a data manipulation library, while SQLAlchemy is a database toolkit that allowed us to create a database to store user information.

The app variable is an instance of the Flask class. It configures the application by setting the 'UPLOAD_FOLDER' directory for file uploads and creates a secret key to ensure client-side security. SQLAlchemy is also used to create two classes: User and Upload, which will represent the database models for the user information and uploaded files.

Next, there are several routes that handle different actions of the application. The first route is the home route, which requires a user to be logged in to access it. It renders a base HTML template. The second route, /upload, handles file uploads. It first checks if the user is logged in before processing the file upload. If the file is a CSV and allowed to be uploaded, the file is saved to the UPLOAD_FOLDER and the file upload record with the user is saved in the database.

The /select_column/<filename> route is for selecting a column and a model to use for forecasting. When a column and model are selected, it first reads the uploaded CSV file using Pandas. Then it calls the appropriate gap insertion function based on the selected method. Once the gaps are filled, the gap-filled dataset is saved, and the user is redirected to the appropriate result page based on the selected model. The result pages include xgboost_result, lstm_result, arima_result, hwes_result, and sarimax_result, each of which displays the results of the respective model with plots and metrics.

Lastly, there are several routes for user authentication, including register, login, and logout. These routes allow users to create a new account, log in to their account, and log out of their account, respectively. These routes also make use of the User class and the database to store and verify user information. Finally, the application is run using the create_database() function and the app.run() method.

6 . Challenges We Faced: Machine Learning.

This project was a challenging but rewarding experience for both of us. Before starting, we did not have a lot of prior experience working with machine learning and due to this we had to do extensive research to understand the packages we could use to implement each forecasting model and the underlying algorithms behind each one. Understanding the mathematical concepts used in each model was particularly challenging. We spent a considerable amount of time following tutorials and watching videos to gain a better understanding of the topic and each model. It was difficult to balance this project with other modules and assignments.

Additionally we found it challenging ensuring that our time-series datasets were stationary. We had to conduct several tests, such as the ADF test, to determine whether the data was stationary or not. If the data was found to be non-stationary, we had to take the necessary steps to transform it into a stationary series to ensure accurate forecasting results. We also had to research methods of testing the accuracy of our forecasting models. We had to use testing metrics such as Root Mean Squared Error,

Mean Absolute Error, and Mean Absolute Percentage Error to evaluate the performance of our models and determine which one performed the best. This required a deep understanding of the underlying algorithms and mathematics behind each model. Overall, ensuring the accuracy and reliability of our forecasting results was a major challenge throughout the project.

7. Workflow

7.1 Zoom

We made sure to communicate regularly using Zoom. This allowed us to have virtual face-to-face meetings, where we could share screens and discuss our progress on the tasks we had divided between each of us. During these meetings, we would discuss the deadlines we had to meet, how both of us were getting on with our own tasks, and if we had any questions that we needed to address. We were mindful to take note of questions that we could not answer, and we would arrange to ask our supervisor, Mark Roantree, during our meetings with him. Mark's provided us with several recommendations, such as using gaps with different granularities, which helped us improve the quality of our forecasting results.

Overall, this workflow allowed us to stay on top of our tasks, resolve any issues that arose, and ultimately deliver a successful project.

7.2 Regular Meetings

We met weekly to discuss what we intended to complete, and to update each other on the status of our respective tasks. As the project deadlines approached, these meetings became more frequent and lasted for longer durations, as we had to ensure that we were meeting all the requirements of the project. Additionally, as stated above we met with our supervisor regularly to update him on our progress and to ask any questions that we had.

7.2 Google Colab

In order to maintain our code and ensure that it was running smoothly, we utilised Google Colab. Due to the nature of our project, it was much easier and efficient to execute singular cells and see the results for just them rather than maintaining a complete Python script. This allowed us to quickly identify any errors or bugs in our code and make necessary changes in a timely manner. Google Colab also allowed for easy sharing of code among the pair of us.

9. **Testing: Metrics we used to measure the accuracy of our Time Series Forecasts.**

- *Root Mean Squared Error (RMSE)*

  RMSE is a measure of the differences between the actual values and the predicted values of a time series. It calculates the square root of the average of the squared differences between the predicted and actual values. RMSE is commonly used to measure the accuracy of a forecasting model, with lower RMSE values indicating higher accuracy.

- *Mean Absolute Error*

  Mean Absolute Error (MAE) is another metric used to evaluate the accuracy of time series forecasts. It calculates the average absolute differences between the predicted and actual values. Like RMSE, a lower MAE value indicates higher accuracy.

- *Mean Absolute Percentage Error*

  (MAPE) is a commonly used metric that calculates the percentage difference between the predicted and actual values of a time series. We used it to calculate the absolute differences between the predicted and actual values as a percentage of the actual values, and then took the average of those percentages. MAPE is commonly used to compare the accuracy of different forecasting models on the same data.

- *The Augmented Dickey-Fuller*

  When testing the stationarity of our time series dataset, we used the Augmented Dickey-Fuller (ADF) test. The ADF test is a statistical test that determines whether a time series is stationary or not.To use the ADF test, we first calculated the rolling mean and standard deviation of our dataset to check for any trends or patterns over time using the rolling function within Pandas.We then applied the ADF test to our dataset and interpreted the results. If the ADF test statistic was less than the critical value and the p-value was less than 0.05, we rejected the null hypothesis that the time series was non-stationary and concluded that the series was stationary. However, if the ADF test statistic was greater than the critical value and the p-value was greater than 0.05 we used several methods such as differencing / time shifting and took the log of the time series to make it stationary. We then reran the ADF Test to test for stationarity again.

10. Future Work

While we are happy with the work we have completed for this project there are still more things we would like to add to it. Unfortunately due to time constraints, other assignments and external problems we couldn't do everything. In this section, we will highlight features we would like to add and how they could be added.

During the development of our project, we researched and implemented five different time-series forecasting models to fill gaps in the dataset. However, we were interested in exploring other models that could potentially yield more accurate results. Unfortunately, due to time constraints, we were unable to implement additional models. If we were to continue with this project, we would have liked to explore other models such as Prophet and DeepAR. These models could provide users with even more options and potentially improve the accuracy of the forecasting results