PROJECT REPORT ML FINAL ASSIGNMENT

JANUARY 3, 2024

MACHINE LEARNING: CS7CS4
BY: ALOK SINGH
STUDENT ID:23336803

PART 1 FINAL ASSESMENT

1. INTRODUCTION

The global pandemic COVID-19 had a very devastating impact on life of humans across the world which included change in mobility and transportation pattern in different areas of world. There was a need to restructure the complexities and upgrade infrastructure and services post-pandemic, therefore there was a need to study the changes that this pandemic brought in transportation pattern. Here our focussed area is Ireland so we will be analysing the usage of bike by people across the country. The aim of this project is to understand and analyse the impact of COVID-19 pandemic on usage of Dublin bikes network.
The primary objective of this project is to evaluate deviations in city-bike usage particularly during two periods: the pandemic period and post-pandemic phase. This project can be very useful in planning and optimizing the city-bike-system, to align it more with altered behaviour resulting from pandemic.
To achieve this objective, I used the following approach, initially data from Dublin bikes was downloaded from 2018 Q3 to 2023 Q4 and was pre-processed, which included handling of missing data and making standardised data format across all the files. Then analysis was done on temporal trends and spatial dynamics that might have altered due to pandemic's influence. To analyse these both descriptive statistics and advanced machine learning methodologies were used which included ARIMA time series forecasting, prophet.
This project not only explore descriptive aspects of bike usage but also delves into predictive analytics. By using the machine learning models, we tried to explore how would the bike usage would have been if there was no pandemic.
The subsequent section of this report elaborates all the above-mentioned steps.

2. Data Pre-processing and Feature Engineering

2.1 Introduction

The Data pre-processing and Feature Engineering is essential in preparing Dublin bikes dataset for further analysis. This phase included transforming raw data into a uniform structured format, which included cleansing, feature augmentation and temporal segmentation.

2.2 Dataset Overview

I downloaded Dublin bikes dataset from Dublinbikes API, from 2018 Q3 to December 2023.it included important columns like STATION ID, TIME, LAST UPDATED, NAME, BIKE_STANDS, AVAILABLE_BIKE_STANDS, AVAILABLE_BIKES, STATUS, ADDRESS, LATITUDE, and LONGITUDE. These column gives information about availability of bikes and location of station.

2.3 Data Cleaning

Here I started with a folder (ML final assignment\before cleaning) where I placed all the files that were to be processed for cleaning and gave a directory where cleaned files were to be saved. The code then iterated all the files in directory ML final assignment\before cleaning the condition to iterate was if the filename starts with dublinbikes_ or dublinbike-historical-data-.

File processing included following steps:

Filename parsing: Here the year was extracted from filename using regular expression.

File Reading: Files was read using pd.read_csv .

Handling Missing Values: Here the missing values (NaN) in all columns were replaced by NA.

2.4 Feature Engineering

Renaming columns and Date time conversion:

Date Time Conversion: In this step columns ('TIME','LAST UPDATED') were converted to appropriate Date time format using pd.to_datetime.
Then these files were then saved to cleaned directory ML final assignment\after cleaning.

Renaming columns: After this step we did further cleaning by making column names standard in all the files as the columns were not consistent in all the files, we did this by performing following steps:
Method used was df.rename here we renamed 'BIKE STANDS' with 'BIKE_STANDS' ,'AVAILABLE BIKE STANDS' with 'AVAILABLE_BIKE_STANDS','AVAILABLE BIKES' with 'AVAILABLE_BIKES' which ensured consistent naming convention. This process helps in avoiding any confusion and makes data aggregation and analysis easier by making data uniform.
After making data uniform then I removed Duplicate entry from the data by specifying the columns from which key is to be made and generating key column in data frame using lambda. Once I got the key, I used drop_duplicates function to remove duplicate entry from data set.

Deriving New Feature: Bike_usage was calculated from available data. The formula used to compute bike usage was:

$$bike_{usage} = \frac{BIKE_{STANDS} - AVAILIBLE_{BIKES}}{bIKE_{STANDS}}$$

Using this formula bike_usage for each entry was calculated this value was between 0 and 1 representing the usage of bike at a particular timestamp. Mean daily bike usage was calculated by taking the average of all
bike_usage on a specific day.

Dataset splitting:

Temporal segmentation: The dataset was then splitted into three temporal periods namely- pre-pandemic ,pandemic and post-pandemic on basis of specific dates (pandemic start date '2020-03-13' and pandemic end date '2021-07-01'). This helped in detailed analysis of bike usage patterns during different phases of pandemic.

Handling Temporal Data:

Temporal Analysis: Then I analysed temporal pattern across different phases of pandemic by employing different statistical techniques which in turn helped in computation of mean daily bike usage for each phase of pandemic.

Temporal Trends in Aggregated Daily Bike Usage:

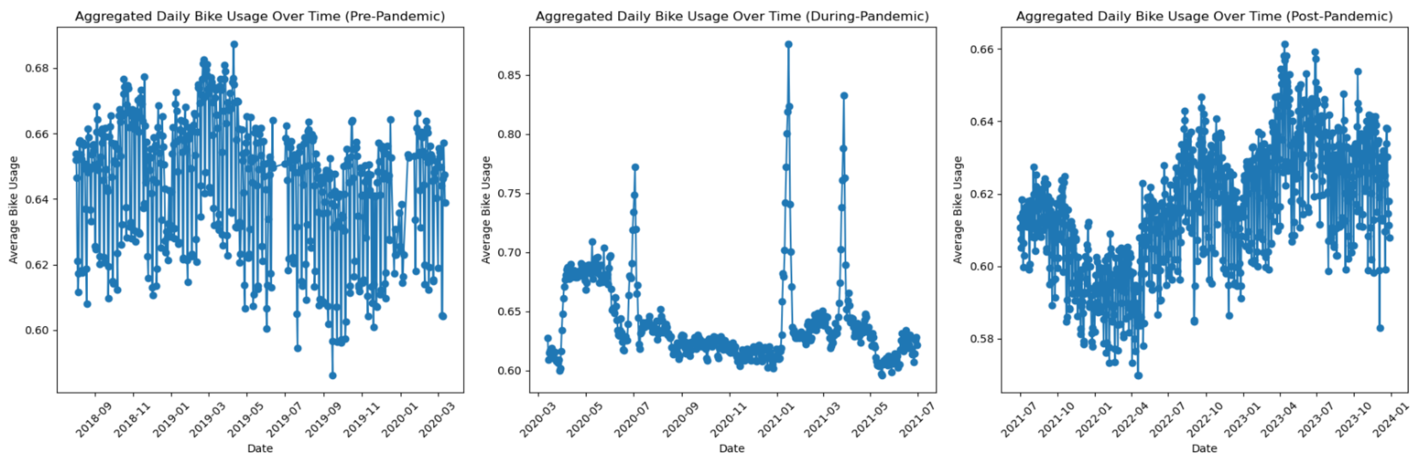

Figure 1 Aggregated Daily Bike Usage During various phases of pandemic

Pre-Pandemic Trends: From figure 1 bike usage pattern highlights the consistent and predictable daily usage of bikes. The data shows a stable trend characterized by regular peaks and troughs which represents regular commuting patterns throughout the week. This leads to a conclusion that before COVID there was a consistent and a habitual utilization of bike service.

During Pandemic Trends: The most significant trend was the reduction in overall bike usage (figure 1.), which was lower than pre pandemic period. Though the overall trend indicates decreased mobility, but we can see there are three significant spike in the plot surpassing the usual values. These spikes are for very short duration which may be due to specific events or changes in restrictions which might have allowed a small period of increased mobility.

Post-Pandemic Trends:

From Figure 1 we can see the bike usage has returned to the level similar to pre pandemic period with similar regular fluctuations with peaks and trough but there are still some variations from pre pandemic period the amplitude of oscillation seem to have reduced a bit as compared to pre pandemic period. Which indicates shift in bike usage patterns due to impact of COVID on behaviour of public.

Statistical Analysis Summary:

Statistics	Pre-Pandemic Bike usage	During-Pandemic Bike usage	Post-Pandemic Bike usage
Mean	0.6442	0.6396	0.6156
Median	0.6490	0.6289	0.6160
Standard Deviation	0.0199	0.0364	0.0176
Minimum	0.5861	0.5956	0.5698
Maximum	0.6874	0.8763	0.6614

Interpretation of statistical analysis:

This analysis shows how bike usage pattern has changed over time.

If we compare mean bike usage over the three periods, we can say that bike usage before the pandemic was 64.42%, during pandemic was 63.96 % which means there is a slight drop in mean usage during pandemic whereas after pandemic it further reduced to 61.56% which indicates the bike usage declined after pandemic period.

If we compare Standard deviation then in pre pandemic period it was 0.0199 which means the bike usage was relatively consistent during this period, during pandemic it increased to 0.0364 which implies that during this phase bike usage showed higher fluctuations as compared to pre pandemic period, in post pandemic period it was again 0.0176 which means the bike usage became more consistent after pandemic.

These statistical analysis sheds light on the fluctuation in bike usage pattern over time. The analysis tells us that there is a decline in bike utilization from pre to post pandemic period. Also, during pandemic phase, it showed both decrease in usage and increase in standard deviation which indicates fluctuations in bike usage trends during this phase.

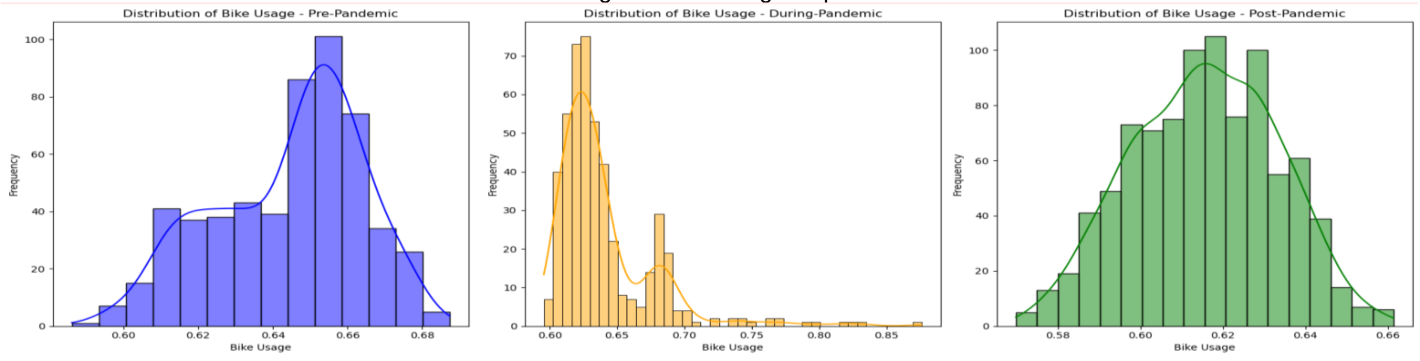

Figure 2: Distribution of bike usage across different periods

3. Machine Learning Methodologies

In this project I explored 2 Time series forecasting methods namely ARIMA (Autoregressive Integrated Moving Average) and Facebook Prophet both had their own advantages, and both gave me a reasonable result but in our scenario, Prophet has an upper hand as it incorporates seasonality modelling, Holiday effect and handles missing data. Below is detailed analysis of both the models:

ASPECT	ARIMA	Prophet
Handling Seasonality	ARIMA models require explicit consideration and inclusion of seasonal components (not explicitly in a formula).	Prophet automatically captures seasonality using Fourier series with seasonalities specified by users.
Handling Trends	ARIMA models need explicit consideration and inclusion of trend components (not explicitly in a formula).	Prophet captures trend using a piecewise linear/logistic growth model with changepoints.
Handling External Regressors	ARIMA doesn't inherently incorporate external factors.	Prophet allows inclusion of additional regressors (e.g., weather data) through additional input columns.
Outliers and Missing Data	ARIMA needs pre-processing or special treatment for outliers and missing values.	Prophet is robust to missing data and outliers due to its built-in imputation method.
Complexity	ARIMA models involve differencing, autoregression, and moving average components, making them complex.	Prophet is relatively simpler, involving trend, seasonality, holidays, and additional regressors.
Interpretability	ARIMA's components (AR, I, MA) are interpretable: AR (autoregression), I (differencing), MA (moving average).	Prophet's components (trend, seasonality, holidays) offer interpretability but may be less intuitive.
Forecast Accuracy	ARIMA's forecast accuracy depends on correctly identifying patterns.	Prophet's accuracy depends on capturing trends, seasonality, and additional regressors effectively.
Performance on Short Time Series	ARIMA may struggle with shorter time series due to the need for a larger number of observations.	Prophet handles shorter time series well due to its imputation and handling of missing values.

Formulas Used in both the time series forecasting methods are given below:
ARIMA:
1) AR(Auto Regression) component: This part models the relationship between current observation and past observation.Φ represents the coefficient of every lagged term. C is the constant term or intercept.
Formula: $y_t = c + \Phi_1 * y_{t-1} + \Phi_2 * y_{t-2} + \cdots + \Phi_p * y_{t-p} + \varepsilon_t$
2) I (Integrated) Component: This represents difference to make time series stationary by eliminating trends and seasonality.
Formula: $\Delta y_t = y_t - y_{t-1}$
3) MA (Moving Average) Component: This represents the relationship of model between current observation and past forecast errors. θ represents coefficient for each lagged forecast error. **εt** represents the error term, **y_t** represents the current value in time series model.
Formula:$y_t = c + \theta_1 * \varepsilon_{t-1} + \theta_2 * \varepsilon_{t-2} + \cdots + \theta_p * \varepsilon_{t-p} + \varepsilon_t$

Prophet: This model was developed by facebook ,it models time series data using an additive regression model which includes multiple components. The formula used by prophet for forecasting time series Y(t) is:
Formula: $y(t) = g(t) + s(t) + h(t) + \epsilon t$
g(t) represents the trend component which captures overall trend in time series, which allows a flexible modelling of both linear and non-linear trends.
s(t) represents the seasonal component: This component incorporates periodic changes or seasonality present in the data, capturing daily, weekly, and yearly patterns.
h(t) represents holiday component: This component includes the effect of holidays and events that significantly impacts time series.
ϵt represents the error term: This accounts for any irregularity or noise that is not captured by the trend, seasonality, or holiday component.
Due to seasonality and holiday component prophet works better in our scenario as compared to ARIMA.

Plots Generated by both the models:

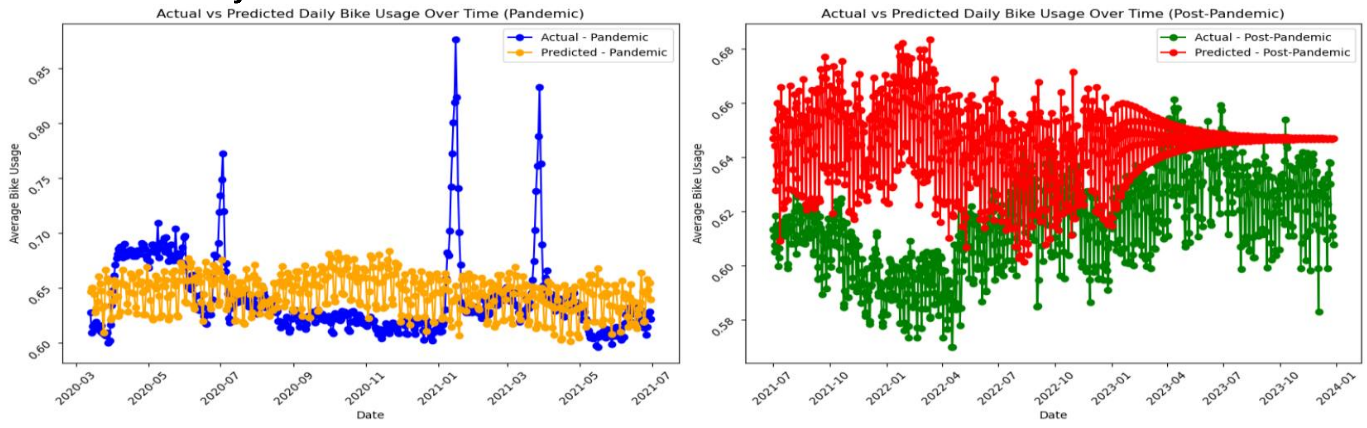

Figure 3: Plots of Actual vs Predicted Daily Bike usage over time (ARIMA Model)

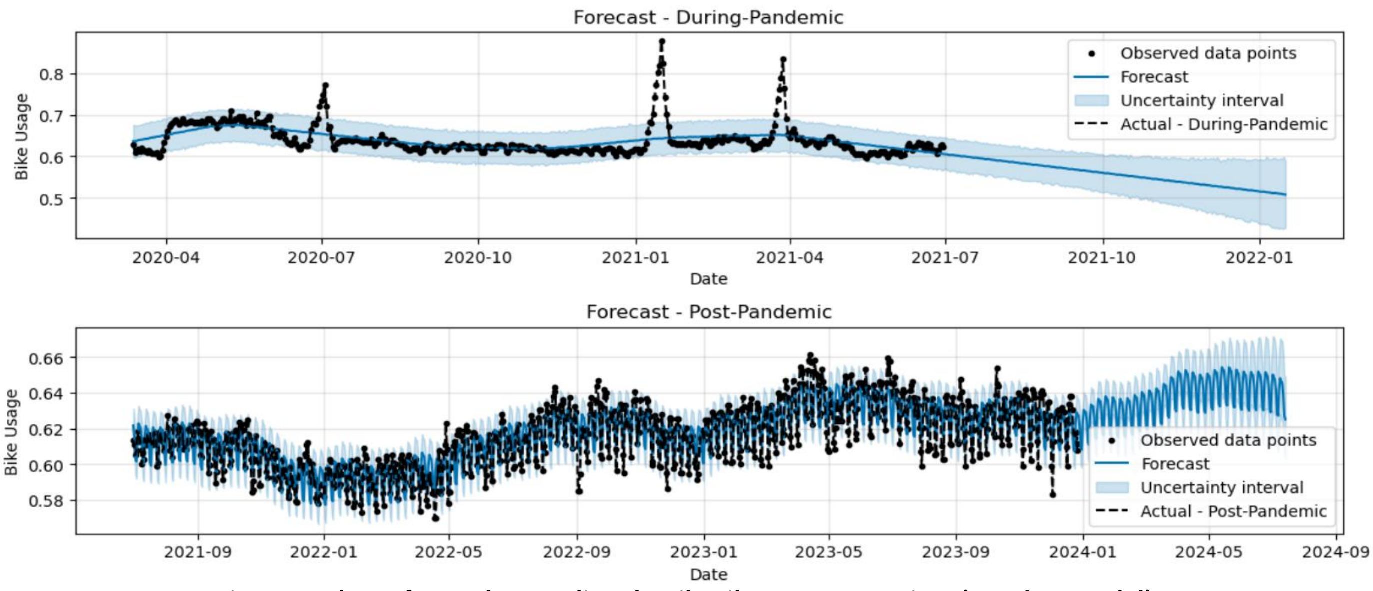

Figure 4: Plots of Actual vs Predicted Daily Bike usage over time (Prophet Model)

1) Plots of Actual vs Predicted Daily Bike usage over time (ARIMA Model): (Figure 3)

ARIMA model was trained on Pre pandemic data to obtain the predictions for Pandemic and post pandemic period i.e., usage of bikes if there was no pandemic.

Pandemic Period Graph: In this graph the blue line represents the actual bike usage during pandemic period.Actual graph shows very high fluctuations which might have been due to uncertainity and varying conditions of pandemic.
The orange line signifies the predicted bike usage based on ARIMA model trained on pre pandemic data . The predictions seem to be consistently less than the actual usage of bikes , which means the model had underestimated the bike usage durinh pandemic period.The reason behind it could be due to change in pattern of bike usage that was absent in pre pandemic data for training.

Post Pandemic Graph: The green line shows the actual bike usage post pandemic,which is more steady as compared to pandemic period.This means that after pandemic the bike usage pattern had returned to more predictable pattern.
The red line shows the predicted values which seems to be over estimated .Though the discrepancy actual between actual and predicted values appear to be less when compared to pandemic period. This indicates that the post pandemic bike usage pattern is somewhat more similar to pre pandemic pattern on which model was trained.

Observations from both the plots:

Model's prediction does not capture the trends accurately, which indicates that model lacks the complexity required to understand the fine details of actual data during these periods. The predictions are smoother than the actual bike usage lines, as ARIMA is known for capturing long term trends rather than short term fluctuations. Model's parameter (order = 5,0,5) are chosen by a function which gives the best parameter for given training dataset (pre pandemic data) based on lowest AIC value. (AIC measures how good the statistical model has been fit lower the value better the fit of model. Order = (p,d,q) where p= AR(Auto Regression) component,
d= I (Integrated) Components, q= MA (Moving Average) Component)

2) Plots of Actual vs Predicted Daily Bike usage over time (Prophet Model): (Figure 4)

During pandemic Forecast graph: The first graph displays forecast of data during pandemic in this plot the black dotted line represents the actual bike usage which exhibits a relative stable trend with some noticable peaks .The blue line the forecast that prophet model did which is closely follows the actual usage which means the model has learned the trend well . The light blue shaded region is uncertainty interval,which seems to be very narrow, indicating model's confidence in its predictions.The forecast is extended beyond the actual data points into future,which shows models prediction for future days.

Post Pandemic Forecast graph: In this graph the actual bike usage is shown by black dotted line ,which shows more fluctuations as compared to during pandemic,which means after pandemic the bike usage returned to its normality with varying bike usage patterns. The forecast line captures these fluctuations to some extent but with visible deviations,especially where the actual usage has sharp high and low points. The uncertainity interval in this plot is broader as compared to during pandemic period , especially towards future predictions,which implies that the variations in the post pandemic data may be greater,which makes it hard for the model to predict with high confidence.

Observations from both the plots:

- The estimates that follow the path of the actual data and the central tendency of the bike usage is captured by the model.
- With the increasing prediction uncertainty, the uncertainty intervals grow broader, and the forecasts extends to the future.
- The post-pandemic period's higher variability proves to be difficult to provide precise future predictions.

Comparison of ARIMA and Prophet plots:

- The ARIMA model underestimated the bike usage during pandemic due to models dependence on pre pandemic data thereby lacking adaptability to sudden changes in usage patterns Whereas Prophet model showed better adaptation during pandemic by closely following the actual trends, which indicated higher capacity to learn from abrupt changes in usage pattern.
- The ARIM model overestimates the post pandemic bike usage predictions which inturn means adjustment issue to returning pattern whereas Prophet captured the fluctuations well in post pandemic period but showed visible deviations at sharp lows and highs but still maintained relatively more accurate trends.
- In usage patterns during and post-pandemic period, the prophet model gives better adaptability to unpredictable changes.
- The unpredictability introduced is superiorly performed by prophet model as compared to ARIMA.
- Overall suitability with its flexibility to sudden changes is handled well by Prophet model.

4. Evaluations

In evaluating the models performance we will be using the following evaluation metrics:

- **MAPE (Mean Absolute Percentage Error):**This measures the average percentage difference between predicted predicted and actual values.
  Interpretation: A lower MAPE value indicates small average error between predicted and actual values.It represents model's overall accuracy in predicting the target variable.
  Formula : $\frac{1}{n}\sum_{i=1}^{n}\left|\frac{A_i-F_i}{A_i}\right| X\ 100$ where Ai = Actual value at time I ,F_i =Predicted value at time I and n= Total number of observations
- **RMSE(Root Mean Squared Error):**It measures the square root of average of squared difference between predicted and actual values.
  Interpretation: It provides an estimate of model's prediction error in same units as target variable.
  Formula : $\sqrt{\frac{1}{n}\sum_{i=1}^{n}(A_i-F_i)^2}$ where Ai = Actual value at time I ,F_i =Predicted value at time I and n= Total number of observations
- **MAE(Mean Absolute Error):**This measures the absolute difference between predicted and actual values.
  Interpretation: This gives average magnitude of errors irrespective of their direction (positive or negative).
  Formula : $\frac{1}{n}\ \sum_{i=1}^{n}|A_i-F_i\ |$ where Ai = Actual value at time I ,F_i =Predicted value at time I and n= Total number of observations.

Evaluation metrics for ARIMA and Prophet model:

Model	Phase	MAPE	RMSE	MAE
ARIMA	During Pandemic	4.73%	0.04	0.03
ARIMA	Post Pandemic	5.15%	0.04	0.03
ARIMA	Combined	4.94%	0.04	0.03
Prophet	During Pandemic	2.59%	0.03	0.017
Prophet	Post Pandemic	0.95%	0.0076	0.0058
Prophet	Combined	1.51%	0.019	0.0098

ARIMA MODEL:

- **During Pandemic Evaluation:** The Arima model showed a MAPE value of 4.73% and RMSE of 0.04 and MAE of 0.03 which implies a fairly accurate forecast but with a moderate error rate.
- **Post Pandemic Evaluation:** Model displayed a slightly higher MAPE value of 5.15 % and maintained a consistent RMSE of 0.04 and MAE of 0.03.
- **Combined Period Evaluation:** Model showed a MAPE of 4.94 % and a consistent RMSE of 0.04 and MAE of 0.03. it displayed stable error rates in both the pandemic phases.

Prophet Model:

- **During Pandemic Evaluation:** This model showed a lower MAPE of 2.59% , an RMSE of 0.03 and MAE of 0.017 during pandemic phase which indicates a higher accuracy and precision as compared to ARIMA.
- **Post Pandemic Evaluation :** This phase showed a lower MAPE of 0.95% and a significantly less RMSE of 0.0076 and MAE of 0.0058 demonstrating a superior predictive accuracy and precision.

- **Combined Period Evaluation:** Model showed a MAPE of 1.51%, an RMSE of 0.019 and MAE of 0.0098 which signifies a consistent and robust forecast capabilities across both pandemic phase.

Comparative analysis of ARIMA and Prophet Model:
The prophet model outperformed ARIMA model in both the phases of pandemic in forecasting, showcasing lower error rates (MAPE,RMSE,MAE).Prophet's lower error Rates showcases its forecasting ability and reliability as compared to ARIMA for this dataset and time frame.

PART 2 FINAL ASSESMENT

Question (i) ROC CURVE

ROC Curves refers to Reciever operating Characterstic Curve. It is a graphical representation that illustrates the performance of a classification model by plotting the True positive Rate (Sensitivity) vs the False positive at various threshold settings.
Evaluation compared to Baseline Classifier :ROC curves helps in evaluating the classifier by comparing its performance against a baseline model.The baseline model might be a random classifier.If the ROC curve of the model is above baseline curve, this signifies a better predictive power.
ROC Curves are preferred over classification accuracy metrics in cases where classes are imbalanced,where the distribution of classes in dataset is unequal.In such cases Classification accuracy metrics can be misleading because it does not take class imbalance into account,whereas ROC curve provides a comprehensive understanding of model performance across various threshold,providing a more in-depth evaluation .
Example : Lets consider we have a machine learning model which has a purpose of classifying if an email is spam or not based on its content. Suppose we have a dataset of 1000 emails,from which 50 emails are actually spam (Class A) and rest 950 are non spam (Class B).
ROC curve: In this case ROC curve will show the model's performance by plotting True Positive rate against False Positive rate at different classification threshold levels.

- True positive Rate(TPR): The ratio of accurately identifying spam email to the total number of actual spam emails.
- False positive Rate(FPR): The ratio of inaccurately identified non spam email to total number of non spam emails.

Evaluation Compared to Baseline Classifier: If we compare the model's ROC curve to a baseline model ,which is represented as random classifier.If ROC curve of model is above the baseline curve ,it means the model has better predictive power than random guessing.Which means model performs better when it tries to distinguish between spam and non spam email than a purely random guess.
ROC curve vs Classification Accuracy metrics :
In this scenario of imbalanced classes where we have only 5% of emails which are spam,if we rely only on classification accuracy it can be misleading.For example if model predicts every mail as non spam,it achieves an accuracy of 95% but in reality it does not identify any mail as spam and saying the models accuracy is 95% is not correct .

Question (ii)The two situation where linear regression gives inaccurate results are given below:

Non linear relationships:
Situation: Linear regression assumes that dependent and independent variables are linearly related to each other. But in real world scenarios variables might not be linearly related to each other. For instance, when the change in the dependent variable does not match the steady rate of change in the independent variable. In this case the linear regression would not give accurate predictions.
Reasoning: lets take an example for this suppose a scenario where the relationship between the amount of fertilizer used by a farmer on a crop does not follow a linear trend for amount of fertilizer vs crop yield but follows a diminishing trend. Let's assume initially as fertilizer increases crop yield increased sharply but beyond a certain point the increase might slowed down or even pleatue.
Solution: To solve this we can use polynomial regression. By introducing higher order terms in model, so that it can capture curvature and better fit non linear patterns in data. Other approach that can be used to address this is using nonlinear models like decision tree to capture complex relationships between dependent and independent variables.
Outliers and influential points:
Situation: The datasets outliers can affect the result of linear regression. An outlier is a point which deviated from general pattern thereby impacting the slope and intercept of the regression line and in turn leading to inaccurate predictions.
Reasoning: imagine a car price prediction model where cost price of most of the cars lies in a certain range but there exist one car which is extremely expensive as compared to rest of the cars in the dataset. This datapoint can significantly affect the regression line thereby changing prediction for price of typical car.
Solution:
Outliers can be handled in different ways . If the outlier is due to error in entry removing or correcting them might fix this issue or we can use regress techniques like robust regression or weighted least square to minimise the influence of outliers on parameters of model thereby leading to accurate predictions.

Question (iii) In SVM and CNN the kernel has different meaning.
SVM Kernels:
In Support vector machines kernels signifies to a method that allows SVMs to operate in a higher dimensional space where it does not explicitly calculates the data points coordinates in that space.Input data is transformed to higher dimensional space by using it to make data more linearly seperable when data is not linearly seperable in its original feature space.
SVM kernels are useful in following situations:
1) **Non-linearity in data** :when data is not linearly seperable then SVM kernel helps to map the data to a higher dimensional space where it might become seperable.
2) **Feature Extraction**: Kernels also helps in learning the complex decision boundaries in dataset.
3) **Avoiding overfitting**: Overfitting can be prevented by proper selection of kernel which inturn controls smoothness of decision boundaries.

Different types of SVM Kernels:
1) **Linear Kernel**: used when data is linearly separable.
2) **Polynomial Kernel**: Uses Polynomial function to map data to higher dimensional space.
3) **Radial Basis Function (RDB) Kernel**: It is often the default choice which is capable of modelling complex decision boundaries.

CNN Kernels:
In Convolution Neural Networks (CNN), Kernel refers to a smaller sized matrix (filter) which is utilized to perform operations like convolution over an input image or feature map.These kernel slides over the input data to extrtact features like edges , textures,patterns etc.
CNN kernels are useful in following situations:
1) **Feature Extraction:** CNN kernel captures the spatial hierarchies and features like shape , edges and patterns etc.
2) **Dimensionality reduction:** The CNN kernels helps in reducing the number of parameters in network thereby making it more efficient .
3) **Translation Invariance:** CNNs uses shared weight(kernels) over input data , which makes them effective in identifying pattern irrespective of their location in image.

Different Types of CNN kernels:
1) **Edge Detection Kernels**: like SOBET, Prewitt or Scharr kernels.
2) **Convolution kernels**: Filters such as Gaussian blur ,sharpen or emboss.
3) **Custom kernels**: These kernels are designed filters to extract unique features.

Use Cases:
- When dataset are relatively small to medium sized and we are dealing with a classification problem then SVM kernel is suitable.
- Whenever we have a task related to image processing , computer vision or task where spatial relationships are important there CNN kernels stands out in these tasks.

Thereby SVM kernels helps in changing data to facilitate better decision boundary, whereas CNN kernel performs Better when feature extraction and learning representations from input data is important specifically in context of images and spatial information. Both the SVM and CNN kernel plays an important role in Machine learning based on the scenario that we are analysing and nature of data.

Question (iv)
K-fold cross-validation is a resampling strategy that divides a dataset into several subsets, or folds, in order to regularly assess how well a machine learning model generalises. This resampling's main goal is to guarantee solid and trustworthy model evaluation by utilising the full dataset and preventing overfitting.
Here's a clarification and illustration:
The main idea behind resampling is to mimic the model's behaviour on hypothetical data. We accomplish several important goals by dividing the dataset into various groups and utilising each subset successively as training and validation data:
Optimising Data Utilisation: This makes certain that each and every data point has an opportunity to be included in the validation set. This contributes to the evaluation's increased representation of the whole dataset.

Minimising Overfitting:
By training and validating the model on distinct subsets, overfitting is avoided and the model is forced to improve its generalisation to new data by not becoming too used to the training set.

Performance Estimation:
We get k performance estimates by iterating through this process k times, where k is the number of folds. The generalisation performance of the model is then evaluated by averaging these estimates, which yields a more

consistent and trustworthy result.

Example:
For a binary classification problem, let us use a dataset of 100 samples, of which 70 belong to class A and 30 to class B. The k-fold cross-validation with k=5 (5-fold cross-validation) is what we wish to employ.
Five subsets (folds) of the dataset were randomly selected for the first iteration.
One of the folds serves as the validation set for each iteration, while the other four are used for training. For example, fold 1 serves as the validation set in the first iteration, whereas folds 2, 3, 4, and 5 are joined for training.
To provide every fold an opportunity to serve as the validation set, we repeat this procedure for all 5 folds.
Following the completion of every iteration, we have five performance metrics (for example, accuracy) from each fold. To get an indication of our model's overall performance, we take the average of these scores.
When It Is Appropriate:
Huge Enough Datasets: When you have a big enough dataset to guarantee that every fold is representative, K-fold cross-validation works really well.
Balanced Datasets: This method performs best when the dataset is spread approximately equally throughout the classes.
Model Comparison: This is helpful for comparing the outcomes of several models or for fine-tuning hyperparameters.
Performance assessment: Excellent for giving a reliable approximation of a model's performance.
When It Is Not Appropriate:
Particularly Small Datasets: If the dataset is really small, there may not be sufficient data in each fold to support useful training and assessment.
Datasets with a high degree of imbalance: Stratified k-fold cross-validation should be applied to datasets with a high degree of imbalance to guarantee that the proportions of each class distribution are maintained at each fold.
Time-Series Data: The temporal order is crucial to understanding when working with time-series data files. Time-series cross-validation and other time-based splits are better suited.
Models Requiring a Lot of Computing Power: K-fold cross-validation might not be feasible if training a model requires a lot of computing power.

REFERENCES:

- For the paper "Forecasting at Scale" by Taylor, S. J., & Letham, B. (2018):

  Taylor, S. J., & Letham, B. (2018). Forecasting at Scale. Journal of Computational and Graphical Statistics, 27(3), 509-526.
  DOI Link: https://doi.org/10.1080/10618600.2017.1384734

  Prophet's GitHub repository: https://github.com/topics/prophet?o=desc&s=updated

- For the book "Time Series Analysis: Forecasting and Control" by Box, G. E. P., Jenkins, G. M., & Reinsel, G. C. (2015):

  Box, G. E. P., Jenkins, G. M., & Reinsel, G. C. (2015). Time Series Analysis: Forecasting and Control. John Wiley & Sons.
  Link: https://www.scirp.org/reference/ReferencesPapers?ReferenceID=2132630

- For the book "Introduction to Time Series and Forecasting" by Brockwell, P. J., & Davis, R. A. (2016):

  Brockwell, P. J., & Davis, R. A. (2016). Introduction to Time Series and Forecasting. Springer.
  Link: https://warin.ca/ressources/books/2016_Book_IntroductionToTimeSeriesAndFor.pdf

- For the book "Forecasting: Principles and Practice (2nd ed.)" by Hyndman, R. J., & Athanasopoulos, G. (2018):

  Hyndman, R. J., & Athanasopoulos, G. (2018). Forecasting: Principles and Practice (2nd ed.). OTexts.
  Link: https://www.scirp.org/reference/referencespapers?referenceid=2849375

- For the paper "Support-Vector Networks" by Cortes, C., & Vapnik, V. (1995):

  Cortes, C., & Vapnik, V. (1995). Support-Vector Networks. Machine Learning, 20(3), 273–297.
  Link to the Paper: https://www.scirp.org/reference/referencespapers?referenceid=1150668

APPENDIX:

```
import os
import pandas as pd
import re
#reading files where  all the dublin dataset is present
directory = r'\ML final assignment\before cleaning'
cleaned_directory = r'\ML final assignment\after cleaning'
if not os.path.exists(cleaned_directory):
    os.makedirs(cleaned_directory)
#finding year from name of file  and cleaning data  removing empty cells
for filename in os.listdir(directory):
    if filename.endswith(".csv") and (filename.startswith("dublinbikes_") or
filename.startswith("dublinbike-historical-data-")):
        print(f"Processing file: {filename}")
        file_path = os.path.join(directory, filename)

        year_match_1 = re.search(r'dublinbike-historical-data-(\d{4})-(\d{2})\.csv$', filename)
        year_match_2 = re.search(r'dublinbikes_(\d{8})_(\d{8})\.csv$', filename)

        if year_match_1:
            year = year_match_1.group(1)
            print(f"Year extracted using pattern 1: {year}")
        elif year_match_2:
            year = year_match_2.group(1)[:4]
            print(f"Year extracted using pattern 2: {year}")
        else:
            print(f"No year found in filename: {filename}")
            continue

        year_directory = os.path.join(cleaned_directory, year)


        if not os.path.exists(year_directory):
            os.makedirs(year_directory)
            print(f"Created directory for year {year}")

        cleaned_file_path = os.path.join(year_directory, f"cleaned_{filename}")


        df = pd.read_csv(file_path)


        print(f"Columns in '{filename}': {df.columns.tolist()}")

# filling empty  cells with NA
        columns_to_fill_na = df.columns.tolist()

        num_na_before_cleaning = df[columns_to_fill_na].isna().sum().sum()
        df[columns_to_fill_na] = df[columns_to_fill_na].fillna('NA')
        df['TIME'] = pd.to_datetime(df['TIME'])
        df['LAST UPDATED'] = pd.to_datetime(df['LAST UPDATED'])
        num_na_after_cleaning = df[columns_to_fill_na].isna().sum().sum()
 #saving file to  cleaned directory

        df.to_csv(cleaned_file_path, index=False)

        print(f"File '{filename}' cleaned and saved to '{cleaned_file_path}'")
        print(f"NA values added in '{filename}': {num_na_after_cleaning - num_na_before_cleaning}")


# In[8]:


import os
import pandas as pd

# data to segeregate data on basis of pre covid during covid and after covid
cleaned_directory = r'\ML final assignment\after cleaning'
saving_directory= r'\ML final assignment\concatenated data'
if not os.path.exists(saving_directory):
    os.makedirs(saving_directory)

def load_cleaned_data(folder_path):
    data_frames = []
    for filename in os.listdir(folder_path):
        if filename.endswith(".csv") and filename.startswith("cleaned_"):
            file_path = os.path.join(folder_path, filename)
            df = pd.read_csv(file_path)

            df.rename(columns={
                'BIKE_STANDS': 'BIKE STANDS',
                'AVAILABLE_BIKE_STANDS': 'AVAILABLE BIKE STANDS',
                'AVAILABLE_BIKES': 'AVAILABLE BIKES'
            }, inplace=True)

            data_frames.append(df)
    return pd.concat(data_frames, ignore_index=True)


all_data = load_cleaned_data(cleaned_directory)


all_data['TIME'] = pd.to_datetime(all_data['TIME'])
# defining period for covid dates
before_covid_start = pd.Timestamp('2018-01-01')
before_covid_end = pd.Timestamp('2020-02-29')
during_covid_start = pd.Timestamp('2020-03-01')
```

```
during_covid_end = pd.Timestamp('2021-07-31')
after_covid_start = pd.Timestamp('2021-08-01')
after_covid_end = pd.Timestamp('2023-12-31')


# Condition to separate data
before_covid_data = all_data[(all_data['TIME'] >= before_covid_start) & (all_data['TIME'] <=
before_covid_end)]
during_covid_data = all_data[(all_data['TIME'] >= during_covid_start) & (all_data['TIME'] <=
during_covid_end)]
after_covid_data = all_data[(all_data['TIME'] >= after_covid_start) & (all_data['TIME'] <=
after_covid_end)]


before_covid_data = before_covid_data.drop_duplicates()
during_covid_data = during_covid_data.drop_duplicates()
after_covid_data = after_covid_data.drop_duplicates()
# saving files to directory mentioned above

before_covid_data.to_csv(os.path.join(saving_directory, 'before_covid_data.csv'), index=False)
during_covid_data.to_csv(os.path.join(saving_directory, 'during_covid_data.csv'), index=False)
after_covid_data.to_csv(os.path.join(saving_directory, 'after_covid_data.csv'), index=False)



# In[9]:



#code to remove duplicates from file
directory = r'\ML final assignment\concatenated data'


output_directory = r'\ML final assignment\removed duplicates'
if not os.path.exists(output_directory):
    os.makedirs(output_directory)


def load_cleaned_data(folder_path):
    data_frames = []
    for filename in os.listdir(folder_path):
        if filename.endswith(".csv"):
            file_path = os.path.join(folder_path, filename)
            df = pd.read_csv(file_path)


            df.rename(columns={
                'BIKE STANDS': 'BIKE_STANDS',
                'AVAILABLE BIKE STANDS': 'AVAILABLE_BIKE_STANDS',
                'AVAILABLE BIKES': 'AVAILABLE_BIKES'
            }, inplace=True)

            data_frames.append(df)
    return pd.concat(data_frames, ignore_index=True)


all_data = load_cleaned_data(directory)
#making key to get compare and check for duplicates

key_columns = ['STATION ID', 'TIME', 'LAST UPDATED', 'NAME', 'BIKE_STANDS', 'AVAILABLE_BIKE_STANDS',
               'AVAILABLE_BIKES', 'STATUS', 'ADDRESS', 'LATITUDE', 'LONGITUDE']


all_data['Composite_Key'] = all_data[key_columns].apply(lambda row: '_'.join(row.values.astype(str)),
axis=1)

#Dropping duplicates
initial_rows = all_data.shape[0]
cleaned_data_without_duplicates = all_data.drop_duplicates(subset='Composite_Key')
rows_removed = initial_rows - cleaned_data_without_duplicates.shape[0]
print(f"{rows_removed} rows removed due to duplicates.")


cleaned_data_without_duplicates = cleaned_data_without_duplicates.drop(columns='Composite_Key')

#saving to  output directory
output_file = os.path.join(output_directory, 'cleaned_concatenated_data_without_duplicates.csv')
cleaned_data_without_duplicates.to_csv(output_file, index=False)

print(f"File saved: {output_file}")


#!/usr/bin/env python
# coding: utf-8

# In[111]:



import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score
import matplotlib.pyplot as plt
import seaborn as sns
import matplotlib.pyplot as plt
import itertools
import warnings
```

```
from statsmodels.tsa.arima.model import ARIMA
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
from prophet import Prophet
import matplotlib.pyplot as plt
from prophet.diagnostics import cross_validation,performance_metrics
from prophet.plot import plot_cross_validation_metric
from sklearn.metrics import mean_absolute_error, mean_squared_error
import numpy as np


# In[112]:


#data pre processing
data = pd.read_csv('\ML final assignment\removed
duplicates\cleaned_concatenated_data_without_duplicates.csv')
columns_to_drop = ['LATITUDE', 'LONGITUDE']
data=data.drop(columns=columns_to_drop)


# In[113]:


data['TIME'] = pd.to_datetime(data['TIME'])

data['TIME'] = pd.to_datetime(data['TIME'])


pandemic_start_date = '2020-03-13'
pandemic_end_date = '2021-07-01'


pre_pandemic_data = data[data['TIME'] < pandemic_start_date]
pandemic_data = data[(data['TIME'] >= pandemic_start_date) & (data['TIME'] <= pandemic_end_date)]
post_pandemic_data = data[data['TIME'] > pandemic_end_date]


# In[114]:


data['bike_usage'] = (data['BIKE_STANDS'] - data['AVAILABLE_BIKES']) / data['BIKE_STANDS']
daily_usage_1 = data.groupby(data['TIME'].dt.date)['bike_usage'].mean()
daily_usage_1


# In[115]:
```

```
pre_pandemic_data['bike_usage'] = (pre_pandemic_data['BIKE_STANDS'] -
pre_pandemic_data['AVAILABLE_BIKES']) / pre_pandemic_data['BIKE_STANDS']
daily_bike_usage_pre_pandemic_data =
pre_pandemic_data.groupby(pre_pandemic_data['TIME'].dt.date)['bike_usage'].mean()
daily_bike_usage_pre_pandemic_data


# In[116]:


pandemic_data['bike_usage'] = (pandemic_data['BIKE_STANDS'] - pandemic_data['AVAILABLE_BIKES']) /
pandemic_data['BIKE_STANDS']
daily_bike_usage_pandemic_data =
pandemic_data.groupby(pandemic_data['TIME'].dt.date)['bike_usage'].mean()
daily_bike_usage_pandemic_data


# In[117]:


post_pandemic_data['bike_usage'] = (post_pandemic_data['BIKE_STANDS'] -
post_pandemic_data['AVAILABLE_BIKES']) / post_pandemic_data['BIKE_STANDS']
daily_bike_usage_post_pandemic_data =
post_pandemic_data.groupby(post_pandemic_data['TIME'].dt.date)['bike_usage'].mean()
daily_bike_usage_post_pandemic_data


# In[118]:


pandemic_data['TIME'] = pd.to_datetime(pandemic_data['TIME'])


pandemic_data['DATE'] = pandemic_data['TIME'].dt.date


numeric_columns = pandemic_data.select_dtypes(include=['float64', 'int64']).columns
daily_average_pandemic = pandemic_data.groupby('DATE')[numeric_columns].mean().reset_index()

daily_average_pandemic



# In[119]:


pre_pandemic_data['TIME'] = pd.to_datetime(pre_pandemic_data['TIME'])


pre_pandemic_data['DATE'] = pre_pandemic_data['TIME'].dt.date
```

```
numeric_columns = pre_pandemic_data.select_dtypes(include=['float64', 'int64']).columns
daily_average_pre = pre_pandemic_data.groupby('DATE')[numeric_columns].mean().reset_index()


daily_average_pre



# In[120]:


post_pandemic_data['TIME'] = pd.to_datetime(post_pandemic_data['TIME'])


post_pandemic_data['DATE'] = post_pandemic_data['TIME'].dt.date


numeric_columns = post_pandemic_data.select_dtypes(include=['float64', 'int64']).columns
daily_average_post = post_pandemic_data.groupby('DATE')[numeric_columns].mean().reset_index()


daily_average_post


# In[121]:


plt.figure(figsize=(12, 6))
plt.plot(daily_average_pre['DATE'], daily_average_pre['bike_usage'], label='Pre-Pandemic', color='blue')
plt.plot(daily_average_pandemic['DATE'], daily_average_pandemic['bike_usage'], label='During-Pandemic',
color='orange')
plt.plot(daily_average_post['DATE'], daily_average_post['bike_usage'], label='Post-Pandemic',
color='green')
plt.xlabel('Date')
plt.ylabel('Bike Usage')
plt.title('Average Daily Bike Usage Over Time')
plt.legend()
plt.show()


plt.figure(figsize=(10, 6))
periods = pd.concat([
    daily_average_pre.assign(Period='Pre-Pandemic'),
    daily_average_pandemic.assign(Period='During-Pandemic'),
    daily_average_post.assign(Period='Post-Pandemic')
])

# Plotting Distribution of Bike Usage for Pre-Pandemic
```

```
plt.figure(figsize=(8, 6))
sns.histplot(data=daily_average_pre, x='bike_usage', kde=True, color='blue')
plt.xlabel('Bike Usage')
plt.ylabel('Frequency')
plt.title('Distribution of Bike Usage - Pre-Pandemic')
plt.show()

# Plotting Distribution of Bike Usage for During-Pandemic
plt.figure(figsize=(8, 6))
sns.histplot(data=daily_average_pandemic, x='bike_usage', kde=True, color='orange')
plt.xlabel('Bike Usage')
plt.ylabel('Frequency')
plt.title('Distribution of Bike Usage - During-Pandemic')
plt.show()

# Plotting Distribution of Bike Usage for Post-Pandemic
plt.figure(figsize=(8, 6))
sns.histplot(data=daily_average_post, x='bike_usage', kde=True, color='green')
plt.xlabel('Bike Usage')
plt.ylabel('Frequency')
plt.title('Distribution of Bike Usage - Post-Pandemic')
plt.show()



# In[122]:


fig, axes = plt.subplots(1, 3, figsize=(18, 6))

# Plotting Distribution of Bike Usage for Pre-Pandemic
sns.histplot(data=daily_average_pre, x='bike_usage', kde=True, color='blue', ax=axes[0])
axes[0].set_xlabel('Bike Usage')
axes[0].set_ylabel('Frequency')
axes[0].set_title('Distribution of Bike Usage - Pre-Pandemic')

# Plotting Distribution of Bike Usage for During-Pandemic
sns.histplot(data=daily_average_pandemic, x='bike_usage', kde=True, color='orange', ax=axes[1])
axes[1].set_xlabel('Bike Usage')
axes[1].set_ylabel('Frequency')
axes[1].set_title('Distribution of Bike Usage - During-Pandemic')

# Plotting Distribution of Bike Usage for Post-Pandemic
sns.histplot(data=daily_average_post, x='bike_usage', kde=True, color='green', ax=axes[2])
axes[2].set_xlabel('Bike Usage')
axes[2].set_ylabel('Frequency')
axes[2].set_title('Distribution of Bike Usage - Post-Pandemic')

plt.tight_layout()
plt.show()


# In[123]:


import matplotlib.pyplot as plt

fig, axes = plt.subplots(1, 3, figsize=(18, 6))

# Plotting Aggregated Daily Bike Usage Over Time for Pre-Pandemic
axes[0].plot(daily_average_pre['DATE'], daily_average_pre['bike_usage'], marker='o', linestyle='-')
axes[0].set_xlabel('Date')
axes[0].set_ylabel('Average Bike Usage')
axes[0].set_title('Aggregated Daily Bike Usage Over Time (Pre-Pandemic)')
axes[0].tick_params(axis='x', rotation=45)

# Plotting Aggregated Daily Bike Usage Over Time for During-Pandemic
axes[1].plot(daily_average_pandemic['DATE'], daily_average_pandemic['bike_usage'], marker='o',
linestyle='-')
axes[1].set_xlabel('Date')
axes[1].set_ylabel('Average Bike Usage')
axes[1].set_title('Aggregated Daily Bike Usage Over Time (During-Pandemic)')
axes[1].tick_params(axis='x', rotation=45)

# Plotting Aggregated Daily Bike Usage Over Time for Post-Pandemic
axes[2].plot(daily_average_post['DATE'], daily_average_post['bike_usage'], marker='o', linestyle='-')
axes[2].set_xlabel('Date')
axes[2].set_ylabel('Average Bike Usage')
axes[2].set_title('Aggregated Daily Bike Usage Over Time (Post-Pandemic)')
axes[2].tick_params(axis='x', rotation=45)

plt.tight_layout()
plt.show()


# In[124]:


# Calculate statistical measures for bike usage
def calculate_statistics(data):
    mean_usage = data['bike_usage'].mean()
    median_usage = data['bike_usage'].median()
    std_dev_usage = data['bike_usage'].std()
    min_usage = data['bike_usage'].min()
    max_usage = data['bike_usage'].max()

    return {
        'Mean': mean_usage,
        'Median': median_usage,
        'Standard Deviation': std_dev_usage,
        'Minimum': min_usage,
        'Maximum': max_usage
    }

# Calculate statistics for pre-pandemic, during-pandemic, and post-pandemic data
statistics_pre_pandemic = calculate_statistics(daily_average_pre)
statistics_during_pandemic = calculate_statistics(daily_average_pandemic)
statistics_post_pandemic = calculate_statistics(daily_average_post)

# Print statistics
print("Statistics for Pre-Pandemic Bike Usage:")
print(statistics_pre_pandemic)

print("\nStatistics for During-Pandemic Bike Usage:")
print(statistics_during_pandemic)

print("\nStatistics for Post-Pandemic Bike Usage:")
print(statistics_post_pandemic)


# In[125]:


#function to calculate best value of order for ARIMA model (p,d,q)

def find_best_arima_parameters(data, p_range, d_range, q_range):
    best_aic = float("inf")
    best_order = None

    for p, d, q in itertools.product(p_range, d_range, q_range):
        order = (p, d, q)
        try:
            model = ARIMA(data, order=order)
            model_fit = model.fit()
            aic = model_fit.aic

            if aic < best_aic:
                best_aic = aic
                best_order = order

        except:
            continue

    return best_aic, best_order


p_values = range(0, 6)
```

```
d_values = range(0, 3)
q_values = range(0, 6)


best_aic, best_order = find_best_arima_parameters(train_data_pandemic_pre['bike_usage'], p_values,
d_values, q_values)
print(f"Best AIC: {best_aic}, Best Order: {best_order}")



# In[126]:


#forecasting and training ARIMA model from pre period data and forecasting for during pandemic and after
pandemic data
def fit_arima_train(data):
    data['bike_usage'] = pd.to_numeric(data['bike_usage'], errors='coerce')
    model = ARIMA(data['bike_usage'], order=(5,0, 5))
    model_fit = model.fit()
    return model_fit

def predict_arima(model, data):
    predictions = model.predict(start=0, end=len(data) - 1)
    return predictions


train_data_pandemic_pre = daily_average_pre
model_pandemic_pre = fit_arima_train(train_data_pandemic_pre)


test_data_pandemic = daily_average_pandemic
pandemic_predictions = predict_arima(model_pandemic_pre, test_data_pandemic)



test_data_post_pandemic = daily_average_post
pandemic_predictions_post = predict_arima(model_pandemic_pre, test_data_post_pandemic)


# In[127]:


fig, axes = plt.subplots(nrows=1, ncols=2, figsize=(16, 6))

# Plotting actual vs predicted pandemic bike usage
axes[0].plot(test_data_pandemic['DATE'], test_data_pandemic['bike_usage'], marker='o', linestyle='-',
label='Actual - Pandemic', color='blue')
axes[0].plot(test_data_pandemic['DATE'], pandemic_predictions, marker='o', linestyle='-',
label='Predicted - Pandemic', color='orange')
```

```
axes[0].set_xlabel('Date')
axes[0].set_ylabel('Average Bike Usage')
axes[0].set_title('Actual vs Predicted Daily Bike Usage Over Time (Pandemic)')
axes[0].tick_params(rotation=45)
axes[0].legend()

# Plotting actual vs predicted post-pandemic bike usage
axes[1].plot(test_data_post_pandemic['DATE'], test_data_post_pandemic['bike_usage'], marker='o',
linestyle='-', label='Actual - Post-Pandemic', color='green')
axes[1].plot(test_data_post_pandemic['DATE'], pandemic_predictions_post, marker='o', linestyle='-',
label='Predicted - Post-Pandemic', color='red')

axes[1].set_xlabel('Date')
axes[1].set_ylabel('Average Bike Usage')
axes[1].set_title('Actual vs Predicted Daily Bike Usage Over Time (Post-Pandemic)')
axes[1].tick_params(rotation=45)
axes[1].legend()

plt.tight_layout()
plt.show()


# In[128]:


#Evaluation of ARIMA model
def calculate_mape(y_true, y_pred):
    mask = y_true != 0
    y_true_masked = y_true[mask]
    y_pred_masked = y_pred[mask]
    return np.mean(np.abs((y_true_masked - y_pred_masked) / y_true_masked)) * 100
def calculate_mae(y_true, y_pred):
    return mean_absolute_error(y_true, y_pred)

# Calculate MAPE for during-pandemic predictions
mape_during_pandemic = calculate_mape(test_data_pandemic['bike_usage'].values, pandemic_predictions)
rmse_during_pandemic = np.sqrt(mean_squared_error(test_data_pandemic['bike_usage'].values,
pandemic_predictions))
accuracy_during_pandemic = 100 - mape_during_pandemic
mae_during_pandemic = calculate_mae(test_data_pandemic['bike_usage'].values, pandemic_predictions)
print("Evaluation Metrics - During-Pandemic ARIMA:")
print(f"MAPE : {mape_during_pandemic:.2f}%")
print(f"RMSE : {rmse_during_pandemic:.2f}")
print(f"MAE : {mae_during_pandemic:.2f}")
print("\n")
# Calculate MAPE for post-pandemic predictions
mape_post_pandemic = calculate_mape(test_data_post_pandemic['bike_usage'].values,
pandemic_predictions_post)
```

```python
rmse_post_pandemic = np.sqrt(mean_squared_error(test_data_post_pandemic['bike_usage'].values,
pandemic_predictions_post))
mae_post_pandemic = calculate_mae(test_data_post_pandemic['bike_usage'].values,
pandemic_predictions_post)
accuracy_post_pandemic = 100 - mape_post_pandemic
print("Evaluation Metrics -Post-Pandemi ARIMA:")
print(f"MAPE : {mape_post_pandemic:.2f}%")
print(f"RMSE : {rmse_post_pandemic:.2f}")
print(f"MAE : {mae_post_pandemic:.2f}")

print("\n")

# Calculate combined accuracy for both periods
combined_mae = (mae_during_pandemic + mae_post_pandemic) / 2
combined_mape = (mape_during_pandemic + mape_post_pandemic) / 2
combined_rmse = (rmse_during_pandemic + rmse_post_pandemic) / 2
combined_accuracy = (accuracy_during_pandemic + accuracy_post_pandemic) / 2
print("Evaluation Metrics - During and After Pandemic combined ARIMA:")
print(f"MAE: {combined_mae:.2f}")
print(f"MAPE: {combined_mape:.2f}%")
print(f"RMSE: {combined_rmse:.2f}")



# In[129]:


#Using Prophet model to forecast data duriong and post pandemic data
daily_average_pre = daily_average_pre.rename(columns={'DATE': 'ds', 'bike_usage': 'y'})
daily_average_pandemic = daily_average_pandemic.rename(columns={'DATE': 'ds', 'bike_usage': 'y'})
daily_average_post = daily_average_post.rename(columns={'DATE': 'ds', 'bike_usage': 'y'})


def forecast_prophet(data):
    model = Prophet()
    model.fit(data)
    future = model.make_future_dataframe(periods=200)
    forecast = model.predict(future)
    return model, forecast


model_pre, forecast_pre = forecast_prophet(daily_average_pre)
model_pandemic, forecast_pandemic = forecast_prophet(daily_average_pandemic)
model_post, forecast_post = forecast_prophet(daily_average_post)


fig, ax = plt.subplots(3, 1, figsize=(12, 8))
```

```
model_pre.plot(forecast_pre, ax=ax[0], xlabel='Date', ylabel='Bike Usage', plot_cap=False)
ax[0].plot(daily_average_pre['ds'], daily_average_pre['y'], color='black', label='Actual - Pre-Pandemic',
linestyle='--')
ax[0].set_title('Forecast - Pre-Pandemic')

model_pandemic.plot(forecast_pandemic, ax=ax[1], xlabel='Date', ylabel='Bike Usage', plot_cap=False)
ax[1].plot(daily_average_pandemic['ds'], daily_average_pandemic['y'], color='black', label='Actual -
During-Pandemic', linestyle='--')
ax[1].set_title('Forecast - During-Pandemic')

model_post.plot(forecast_post, ax=ax[2], xlabel='Date', ylabel='Bike Usage', plot_cap=False)
ax[2].plot(daily_average_post['ds'], daily_average_post['y'], color='black', label='Actual - Post-
Pandemic', linestyle='--')
ax[2].set_title('Forecast - Post-Pandemic')


ax[0].legend()
ax[1].legend()
ax[2].legend()

plt.tight_layout()
plt.show()


df_cv = cross_validation(model_pandemic, horizon='250 days', period='90 days', initial='200 days')
df_metrics = performance_metrics(df_cv)


metrics = ['mse', 'rmse', 'mae', 'mape', 'mdape', 'smape', 'coverage']


for metric in metrics:
    fig = plot_cross_validation_metric(df_cv, metric=metric)


# In[130]:


# evaluating Prophet model


def mean_absolute_percentage_error(y_true, y_pred):
    return np.mean(np.abs((y_true - y_pred) / y_true)) * 100


mae_pre = mean_absolute_error(daily_average_pre['y'], forecast_pre['yhat'][:len(daily_average_pre)])
mse_pre = mean_squared_error(daily_average_pre['y'], forecast_pre['yhat'][:len(daily_average_pre)])
mape_pre = mean_absolute_percentage_error(daily_average_pre['y'],
forecast_pre['yhat'][:len(daily_average_pre)])
```

```
rmse_pre = np.sqrt(mse_pre)


mae_pandemic = mean_absolute_error(daily_average_pandemic['y'],
forecast_pandemic['yhat'][:len(daily_average_pandemic)])
mse_pandemic = mean_squared_error(daily_average_pandemic['y'],
forecast_pandemic['yhat'][:len(daily_average_pandemic)])
mape_pandemic = mean_absolute_percentage_error(daily_average_pandemic['y'],
forecast_pandemic['yhat'][:len(daily_average_pandemic)])
rmse_pandemic = np.sqrt(mse_pandemic)

mae_post = mean_absolute_error(daily_average_post['y'], forecast_post['yhat'][:len(daily_average_post)])
mse_post = mean_squared_error(daily_average_post['y'], forecast_post['yhat'][:len(daily_average_post)])
mape_post = mean_absolute_percentage_error(daily_average_post['y'],
forecast_post['yhat'][:len(daily_average_post)])
rmse_post = np.sqrt(mse_post)

predicted_values_post = forecast_post['yhat'][:len(daily_average_post)]
actual_values_post = daily_average_post['y']
accuracy_ratio_post = np.mean(np.abs(actual_values_post - predicted_values_post) /
np.abs(actual_values_post))
accuracy_percentage_post = (1 - accuracy_ratio_post) * 100

predicted_values_pandemic = forecast_pandemic['yhat'][:len(daily_average_pandemic)]
actual_values_pandemic = daily_average_pandemic['y']
accuracy_ratio_pandemic = np.mean(np.abs(actual_values_pandemic - predicted_values_pandemic) /
np.abs(actual_values_pandemic))
accuracy_percentage_pandemic = (1 - accuracy_ratio_pandemic) * 100

print("Evaluation Metrics - During-Pandemic Prophet:")
print(f"MAE: {mae_pandemic}")
print(f"MSE: {mse_pandemic}")
print(f"RMSE: {rmse_pandemic}")
print(f"MAPE: {mape_pandemic:.2f}%")
print("\n")

print("Evaluation Metrics - Post-Pandemic Prophet:")
print(f"MAE: {mae_post}")
print(f"MSE: {mse_post}")
print(f"MAPE: {mape_post:.2f}%")
print(f"RMSE: {rmse_post}")

print("\n")


predicted_values_during_after = np.concatenate((forecast_pandemic['yhat'][:len(daily_average_pandemic)],
forecast_post['yhat'][:len(daily_average_post)]))
actual_values_during_after = np.concatenate((daily_average_pandemic['y'], daily_average_post['y']))
```

```
# Calculating MAE, MSE, and RMSE for during and after pandemic together
mae_during_after = mean_absolute_error(actual_values_during_after, predicted_values_during_after)
mse_during_after = mean_squared_error(actual_values_during_after, predicted_values_during_after)
rmse_during_after = np.sqrt(mse_during_after)

# Calculating accuracy percentage for during and after pandemic together
accuracy_ratio_during_after = np.mean(np.abs(actual_values_during_after - predicted_values_during_after)
/ np.abs(actual_values_during_after))
accuracy_percentage_during_after = (1 - accuracy_ratio_during_after) * 100

mape_during_after = mean_absolute_percentage_error(actual_values_during_after,
predicted_values_during_after)

print("Evaluation Metrics - During and After Pandemic combined Prophet:")
print(f"MAE: {mae_during_after}")
print(f"MSE: {mse_during_after}")
print(f"RMSE: {rmse_during_after}")
print(f"MAPE: {mape_during_after:.2f}%")




# In[131]:


predicted_values_during_after = np.concatenate((forecast_pandemic['yhat'][:len(daily_average_pandemic)],
forecast_post['yhat'][:len(daily_average_post)]))
actual_values_during_after = np.concatenate((daily_average_pandemic['y'], daily_average_post['y']))


# Calculate MAE, MSE, and RMSE for during and after pandemic together
mae_during_after = mean_absolute_error(actual_values_during_after, predicted_values_during_after)
mse_during_after = mean_squared_error(actual_values_during_after, predicted_values_during_after)
rmse_during_after = np.sqrt(mse_during_after)

# Calculate accuracy percentage for during and after pandemic together
accuracy_ratio_during_after = np.mean(np.abs(actual_values_during_after - predicted_values_during_after)
/ np.abs(actual_values_during_after))
accuracy_percentage_during_after = (1 - accuracy_ratio_during_after) * 100

mape_during_after = mean_absolute_percentage_error(actual_values_during_after,
predicted_values_during_after)

print("Evaluation Metrics - During and After Pandemic:")
print(f"MAE: {mae_during_after}")
print(f"MSE: {mse_during_after}")
print(f"RMSE: {rmse_during_after}")
print(f"MAPE: {mape_during_after:.2f}%")
print(f"Accuracy Percentage: {accuracy_percentage_during_after:.2f}%")
```