TRINITY COLLEGE DUBLIN

Assignment: Final Assignment
Name: Frason Francis
CS7CS4/CSU44061 – Machine Learning
Student-ID: 23333313

Dataset:
This dataset DublinBikes API offers a granular view of bike availability and usage across various stands in the city, with data points collected in real-time. It encapsulates a wealth of information including timestamped records of bike stand occupancy, which serves as a reliable foundation for our temporal analysis. By analyzing this dataset, we've been able to trace the patterns of usage before, during, and after the pandemic, offering valuable insights into how public health measures and changes in commuter behavior have affected the dynamics of city-bike utilization. The DublinBikes API dataset is thus instrumental in informing the Dublin City Council's strategies for optimizing the bike-sharing system in the wake of the pandemic's unprecedented impact.

Part - 1

Data Preprocessing:

- *Time Series Segmentation:* The approach starts by segmenting the data into three distinct phases - pre-pandemic, during the pandemic, and post-pandemic. This segmentation is vital as it acknowledges the potential impact of the COVID-19 pandemic on bike usage patterns. By treating these as separate entities, the model can better adapt to the unique characteristics and trends of each period.
- *Data Cleaning and Renaming:* In the initial stages, you focus on ensuring data consistency. This involves renaming columns for uniformity across different data files and removing any duplicate entries. Consistent naming conventions improve readability and prevent errors during data manipulation. Removing duplicates is crucial to avoid skewed analysis or bias in the model.
- *Time Parsing and Rounding:* The TIME column in your data is parsed into datetime objects, ensuring that time-related data is in a format that's conducive to analysis. Furthermore, you've implemented a rounding mechanism to align time records to the nearest 5 minutes or to a daily level. This step reduces the granularity of your data, making it more manageable and suited for capturing broader trends, which are more relevant in this context than minute-by-minute changes.
- *Aggregation of Data:* Post the rounding of time data, there's an aggregation step where data points are summarized (either summed or averaged) over the newly defined time intervals. This aggregation is a form of dimensionality reduction, simplifying the dataset while retaining essential information. It helps in understanding the overall trends in bike stand availability across different time periods without getting lost in the minutiae.

Feature Engineering:

- *Lagged Feature Creation:* One of the most crucial steps in your model is the creation of lagged features. These features are essentially past values of the target variable (bike stand availability) shifted by various time intervals - monthly, weekly, daily, and immediate past trends. In time series forecasting, such lagged features are pivotal as they allow the model to learn from past patterns and make informed predictions about the future.
- *Incorporation of Multiple Time Scales:* Your model uniquely combines features derived from different time scales. By doing so, it captures various cyclical patterns that occur on monthly, weekly, and daily bases. This multiscale feature engineering is particularly beneficial in capturing complex time series behaviors, as it accounts for both short-term fluctuations (like daily changes) and long-term trends (like seasonal variations).
- *Short-Term Trend Analysis:* By incorporating features that represent the immediate past, the model is equipped to understand and react to short-term trends. This is especially valuable in scenarios where there are abrupt changes in patterns, such as those brought on by sudden lockdown measures during the pandemic. It enables the model to quickly adapt to such changes, improving its predictive accuracy for near-term forecasts.

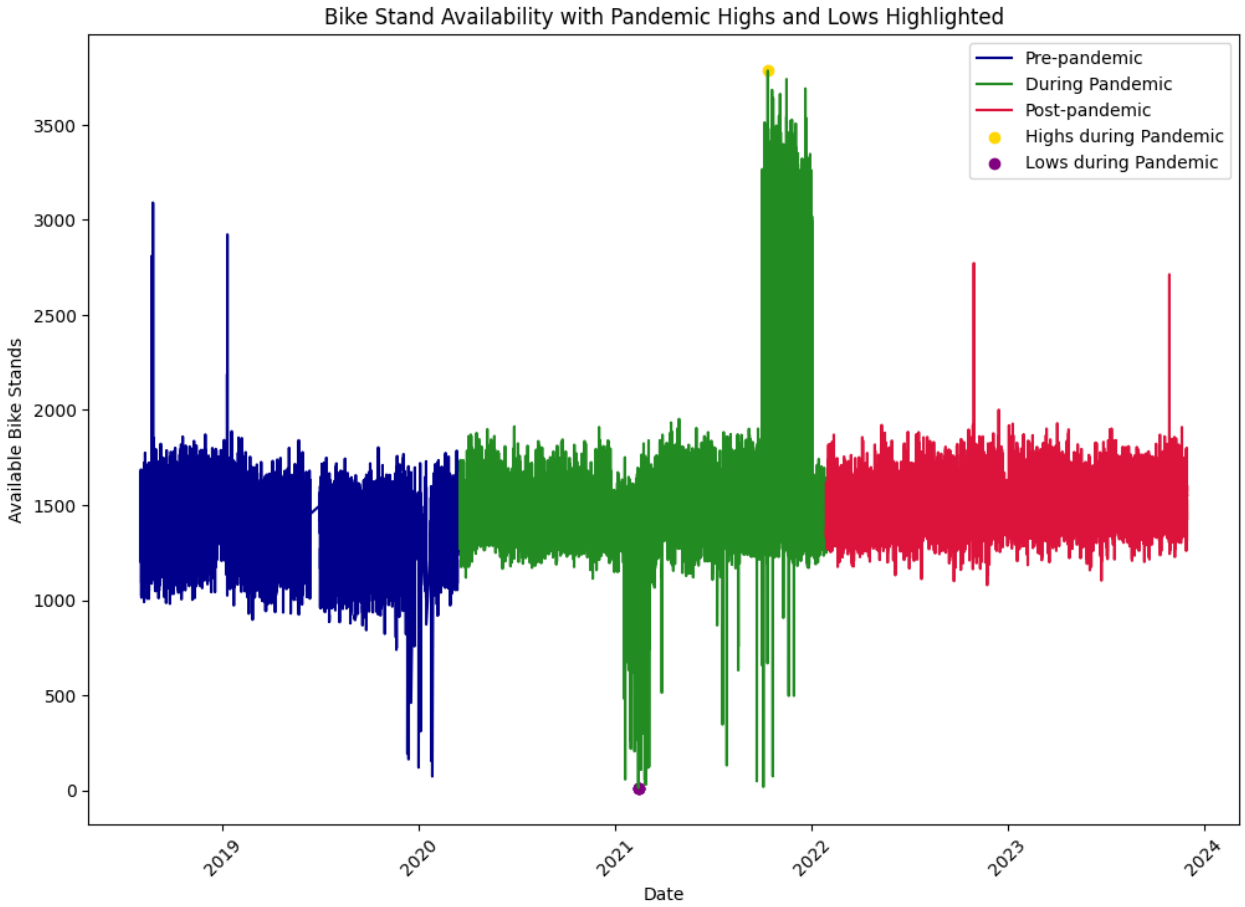

Figure. 1

The fig.1 "Bike Stand Availability with Pandemic Highs and Lows Highlighted" visualizes the fluctuations in bike stand availability in Dublin from 2019 to 2024, segmented into pre-pandemic (blue), pandemic (green), and post-pandemic (red) periods. The stability in availability during the pre-pandemic times contrasts sharply with the pandemic period, which exhibits significant volatility with notable peaks (yellow) and troughs (purple), reflecting the impact of COVID-19 on urban mobility. Post-pandemic data suggests a return to stability, indicating an adjustment to new normalcy in city-bike usage patterns.

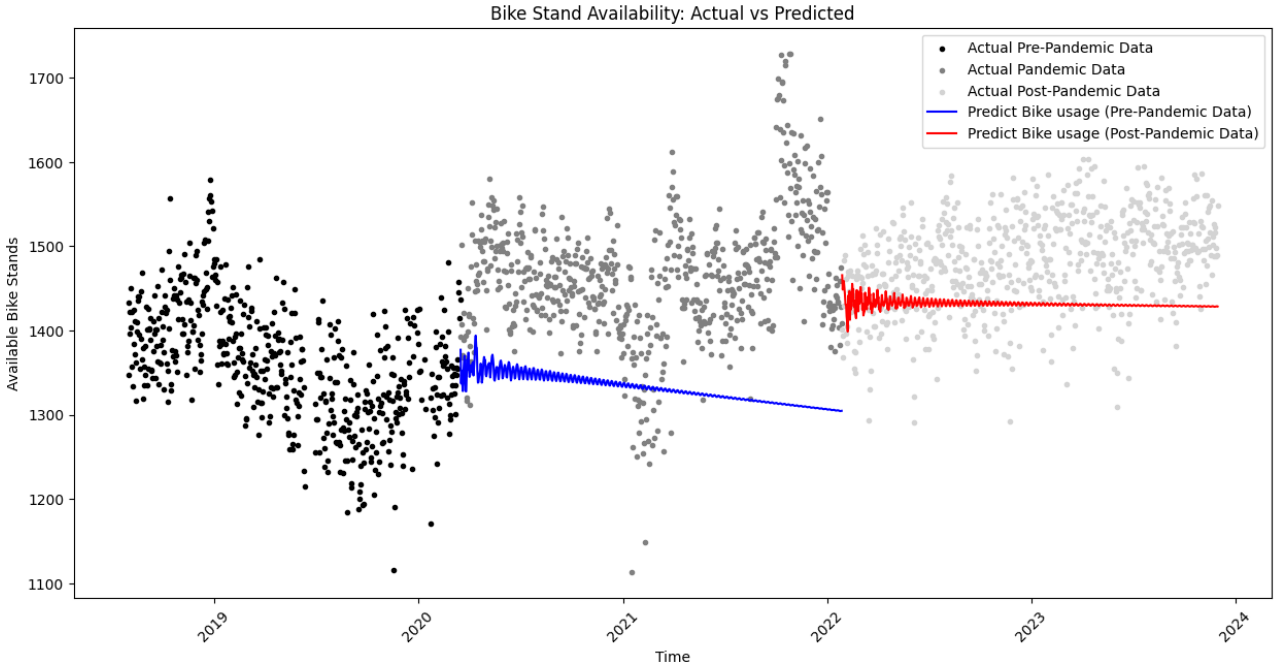

Figure. 2

Machine Learning Techniques:

Ridge Regression:

- Ridge Regression is a variant of linear regression that includes L2 regularization. This technique is particularly useful when there is a risk of overfitting or when the dataset has multicollinearity, meaning that the independent variables are highly correlated.
- The regularization term added to the cost function penalizes large coefficients, effectively shrinking them and reducing model complexity. This leads to more generalized models that perform better on unseen data.
- The alpha parameter in the Ridge Regression controls the strength of this regularization. In the context of the code, alpha is an adjustable hyperparameter, allowing for model tuning.

Time Series Analysis:

- The code is focused on time series forecasting, which involves using historical data points to predict future values. This is a common approach in scenarios where data points are collected or indexed in time order.
- Time series forecasting is essential for understanding and predicting trends, cycles, and seasonal variations over time, which are expected in data such as bike stand usage.

Lagged Features:

- To capture temporal dependencies, the code constructs lagged features. These are past values of the target variable shifted back by a certain number of time steps, which serve as input features for the model.
- Lagged features are a cornerstone of time series modeling as they allow the model to learn from the past patterns to predict future outcomes.

Multi-Step Forecasting:

- Multi-step forecasting refers to predicting more than one future time step during each prediction phase. In the code, this is done iteratively, where each new prediction becomes a part of the input features for the subsequent prediction.
- This approach is vital for planning and decision-making processes that require future estimates over a range of time, not just the immediate next step.

Cross-Validation with TimeSeriesSplit:

- To evaluate the model's performance, the code uses cross-validation with a TimeSeriesSplit. This splitter is specifically designed for time series data, ensuring that the validation process respects the chronological order of observations.
- This method is preferable to random splitting because it prevents future data from leaking into the training process, which could give an unrealistically optimistic measure of the model's predictive power.

Impact During the Pandemic Period:

The analysis revealed a notable deviation in city-bike usage during the pandemic period compared to pre-pandemic forecasts. By employing time series forecasting models, specifically Ridge Regression augmented with lagged features to account for temporal dependencies, we observed changes in usage patterns. The model's predictions, characterized by a Mean Squared Error (MSE) of ***4959.44***, suggested an alteration in the behavior of city-bike users during the pandemic. This was likely influenced by public health guidelines, lockdown measures, and the populace's shifting stance on public transport usage. The higher cross-validation error during the pandemic period, as gauged by our models, was indicative of these disruptions. The results underscored the need for a dynamic response to bike-sharing system management, as traditional usage patterns were temporarily upended.

Impact During the Post-Pandemic Period:
Moving into the post-pandemic era, the analysis aimed to discern the lasting effects of the pandemic on city-bike usage. Interestingly, the post-pandemic period demonstrated a different trend, with the MSE from our predictive models being notably lower at ***4140.37*** compared to the pandemic period. This suggests a stabilization in usage patterns, potentially as a result of the city's adaptation to the 'new normal' and gradual resumption of daily routines. The findings indicate that while there was a significant impact on city-bike usage during the pandemic, the post-pandemic period is showing signs of recovery and a return to predictability, albeit with new patterns that differ from the original forecasts. The lower cross-validation error post-pandemic suggests a more stable and predictable usage pattern that the council can leverage for planning purposes.

Part -2

(i) What is a ROC curve? How can it be used to evaluate the performance of a classifier compared with a baseline classifier? Why would you use an ROC curve instead of a classification accuracy metric?

A ROC (Receiver Operating Characteristic) curve is a tool used in machine learning to assess the performance of a classification model by plotting the True Positive Rate (sensitivity) against the False Positive Rate (1-specificity) at various thresholds. It is particularly useful over simple accuracy metrics in scenarios with imbalanced datasets or when the costs of false positives and false negatives are significantly different. The ROC curve helps in comparing the classifier's performance against a baseline (like random guessing, represented by a diagonal line in the ROC plot) by showing how well the model distinguishes between classes. The Area Under the Curve (AUC) provides a single value summary, with higher values indicating better model performance. Thus, ROC curves offer a more comprehensive evaluation of a classifier's effectiveness in differentiating between classes than mere accuracy.

(ii) Give two examples of situations where a linear regression would give inaccurate predictions. Explain your reasoning and what possible solutions you would adopt in each situation.

- In situations where the data shows complex relationships that linear regression can't adequately capture, its predictions become inaccurate. For instance, consider a scenario where you're trying to predict crop yields based on various factors like rainfall, temperature, and soil quality. If the relationship between these factors and crop yield is non-linear (say, crop yield increases up to a certain level of rainfall but decreases beyond that due to flooding), linear regression would fail to model this accurately.
- Another example is in financial markets, where predicting stock prices based on past performance can be challenging due to the market's volatile nature; this volatility often leads to heteroscedasticity, where the error variance isn't constant across all data points. In these cases, solutions include using non-linear models, like polynomial regression for the crop yield example, to better fit the data's curvature, or employing techniques like ARCH

(Autoregressive Conditional Heteroskedasticity) models in financial forecasting to handle heteroscedasticity. The key is to align the modeling technique with the data's underlying structure and behavior.

(iii) The term 'kernel' has different meanings in SVM and CNN models. Explain the two different meanings. Discuss why and when the use of SVM kernels and CNN kernels is useful, as well as mentioning different types of kernels.

1. *SVM Kernels*:
    - Meaning: In SVMs, a kernel is a function used to take data as input and transform it into the required form. It enables SVMs to solve nonlinear problems by mapping the original nonlinear observations into a higher-dimensional space where they become linearly separable.
    - Usefulness: Kernels in SVM are crucial for dealing with complex datasets where the relationship between classes is not linear. By applying the kernel trick, SVMs can efficiently perform classification tasks without the need to explicitly map data to a higher dimension.
    - Types: Common types of SVM kernels include linear, polynomial, radial basis function (RBF or Gaussian), and sigmoid kernels.
2. *CNN Kernels:*
    - Meaning: In CNNs, a kernel (also known as a filter) is a small matrix used to perform convolution operations across the input data (like an image). It's essentially a feature extractor that slides over the input data to produce feature maps.
    - Usefulness: Kernels in CNNs are used for various tasks such as edge detection, blurring, and sharpening in image processing. They help in extracting high-level features from images, which are crucial for tasks like image classification, object detection, and more.
    - Types: In CNNs, kernels are usually not predefined but learned during the training process. However, the architecture might specify the size of these kernels (e.g., 3x3, 5x5).

(**iv)In k-fold cross-validation, a dataset is resampled multiple times. What is the idea behind this resampling i.e. why does resampling allow us to evaluate the generalization performance of a machine learning model. Give a small example to illustrate. Discuss when it is and it is not appropriate to use k-fold cross-validation.**

- In k-fold cross-validation, the dataset is split into 'k' smaller groups or 'folds'.
- Each fold gets a turn to be the test set, with the rest of the folds combined as the training set.
- This resampling lets every data point be in both the training and test set, giving a more complete check on how the model performs.

Example:
Consider a dataset of 50 students' grades. If you use 5-fold cross-validation, you divide these students into 5 groups of 10. In each of the 5 rounds, one group of 10 students is used for testing and the other 40 for training. By the end, each student's grade has been in the test set once, ensuring all data is used effectively.
When to Use It:
Good to Use: When you don't have a lot of data, k-fold helps use every bit of it for both training and testing. It's also useful for datasets that are pretty evenly spread out.

Not Ideal: It might not work well for very big datasets because it can be slow. Also, if your data has groups that are very different from each other, k-fold might not give accurate results. And for time-based data, where the order of data points matters, k-fold might not be the best choice.
In essence, k-fold cross-validation is about making sure the model is tested on various parts of your data to see how well it can adapt to new, unseen data.

Code Appendix:

PREPROCESSING FILE

```
# Import necessary libraries
import os
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from datetime import datetime
from tqdm import tqdm
import matplotlib.dates as mdates

# Constants for column names
COLUMNS_OLD = ["TIME", "BIKE STANDS", "AVAILABLE BIKE STANDS", "AVAILABLE BIKES"]
COLUMNS_NEW = ["TIME", "BIKE_STANDS", "AVAILABLE_BIKE_STANDS", "AVAILABLE_BIKES"]

# Function to split a file into two dataframes based on a regex pattern
def split_data_by_regex(filename: str, split_regex: str) -> (pd.DataFrame, pd.DataFrame):
    if "Q" in filename:
        df = pd.read_csv(filename, usecols=COLUMNS_OLD)
        df.rename(columns={"BIKE STANDS": "BIKE_STANDS", 'AVAILABLE BIKE STANDS':
'AVAILABLE_BIKE_STANDS', 'AVAILABLE BIKES': 'AVAILABLE_BIKES'}, inplace=True)
    else:
        df = pd.read_csv(filename, usecols=COLUMNS_NEW)

    index = df[df["TIME"].str.contains(split_regex)].index[0]
    return df[:index], df[index:]

# Function to read and organize data from different time periods
def load_data_by_periods(folder: str) -> (pd.DataFrame, pd.DataFrame, pd.DataFrame):
    dfs = {}

    pre_pandemic_mar, pandemic_mar = split_data_by_regex(f"{folder}/start-pandemic/2020Q1.csv", "2020-03-17 .*")
    pandemic_jan, post_pandemic_jan = split_data_by_regex(f"{folder}/end-pandemic/2022January.csv", "2022-01-28 .*")

    for time_period in ["pre-pandemic", "pandemic", "post-pandemic"]:
        period_df = []
        for subdir, _, files in os.walk(f"{folder}/{time_period}"):
            for file in tqdm(files, desc=f"Reading {subdir}"):
                if "Q" in file:
                    df = pd.read_csv(os.path.join(subdir, file), usecols=COLUMNS_OLD)
                    df.rename(columns={"BIKE STANDS": "BIKE_STANDS", 'AVAILABLE BIKE STANDS':
'AVAILABLE_BIKE_STANDS',
                                       'AVAILABLE BIKES': 'AVAILABLE_BIKES'}, inplace=True)
                else:
                    df = pd.read_csv(os.path.join(subdir, file), usecols=COLUMNS_NEW)
                period_df.append(df)
                del df

        if time_period == "pre-pandemic":
            period_df.append(pre_pandemic_mar)
        elif time_period == "pandemic":
            period_df.insert(0, pandemic_mar)
            period_df.append(pandemic_jan)
        elif time_period == "post-pandemic":
            period_df.insert(0, post_pandemic_jan)

        dfs[time_period] = pd.concat(period_df, ignore_index=True)
```

```
    return dfs["pre-pandemic"], dfs["pandemic"], dfs["post-pandemic"]

# Function to round time to the nearest 5 minutes
def round_time_to_nearest_five(time: str) -> datetime:
    dt_format = "%Y-%m-%d %H:%M:%S"
    dt = datetime.strptime(time, dt_format)
    rounded_dt = datetime(dt.year, dt.month, dt.day, dt.hour, (dt.minute // 5) * 5)
    return rounded_dt

# Function to aggregate data by rounded times
def aggregate_data_by_time(df: pd.DataFrame, period: str) -> pd.DataFrame:
    df = df.drop_duplicates()
    tqdm.pandas(desc=f"Rounding {period} times to closest 5 minutes.")
    df['TIME'] = df['TIME'].progress_apply(round_time_to_nearest_five)
    return df.groupby(df['TIME'], as_index=False).aggregate({'BIKE_STANDS': 'sum', 'AVAILABLE_BIKE_STANDS': 'sum',
'AVAILABLE_BIKES': 'sum'})

# Function to save aggregated data
def save_aggregated_data(df: pd.DataFrame, period_name: str):
    df = aggregate_data_by_time(df, period_name)
    df.to_csv(f"data/rounded_{period_name}.csv")

# Function to process and save data for all periods
def process_and_save_all_periods(df1: pd.DataFrame, df2: pd.DataFrame, df3: pd.DataFrame):
    save_aggregated_data(df1, "pre-pandemic")
    save_aggregated_data(df2, "pandemic")
    save_aggregated_data(df3, "post-pandemic")

# Function to read pre-processed data
def load_processed_data() -> (pd.DataFrame, pd.DataFrame, pd.DataFrame):
    pre_pandemic = pd.read_csv("data/rounded_pre-pandemic.csv", parse_dates=['TIME'])
    pandemic = pd.read_csv("data/rounded_pandemic.csv", parse_dates=['TIME'])
    post_pandemic = pd.read_csv("data/rounded_post-pandemic.csv", parse_dates=['TIME'])
    return pre_pandemic, pandemic, post_pandemic

# Function to average data over days
def average_data_daily(df: pd.DataFrame, period: str) -> pd.DataFrame:
    tqdm.pandas(desc=f"Rounding {period} times to day.")
    df['TIME'] = df['TIME'].progress_apply(lambda x: datetime(x.year, x.month, x.day))
    return df.groupby(df['TIME'], as_index=False).aggregate({'BIKE_STANDS': 'mean', 'AVAILABLE_BIKE_STANDS':
'mean', 'AVAILABLE_BIKES': 'mean'})

# Function to save daily averaged data
def save_daily_data(df: pd.DataFrame, period_name: str):
    df = average_data_daily(df, period_name)
    df.to_csv(f"data/daily_{period_name}.csv")

# Function to process and save daily data for all periods
def process_and_save_daily_data(df1: pd.DataFrame, df2: pd.DataFrame, df3: pd.DataFrame):
    save_daily_data(df1, "pre-pandemic")
    save_daily_data(df2, "pandemic")
    save_daily_data(df3, "post-pandemic")


# Function to plot bike stand availability with trend lines
def plot_bike_stand_availability_with_trend_lines(pre_pandemic: pd.DataFrame, pandemic: pd.DataFrame,
                                                  post_pandemic: pd.DataFrame):
    plt.figure(figsize=(12, 8))

    # Plotting each period with trend lines
```

```
    for df, color, label in zip(
            [pre_pandemic, pandemic, post_pandemic],
            ['darkblue', 'forestgreen', 'crimson'],
            ['Pre-pandemic', 'During Pandemic', 'Post-pandemic']
    ):
        plt.plot(df['TIME'], df['AVAILABLE_BIKE_STANDS'], c=color, label=label)

        # Fitting and plotting a polynomial trend line
        z = np.polyfit(mdates.date2num(df['TIME']), df['AVAILABLE_BIKE_STANDS'], 2)
        p = np.poly1d(z)
        plt.plot(df['TIME'], p(mdates.date2num(df['TIME'])), c=color, linestyle='--', label=f'{label} Trend')

    plt.xlabel("Date")
    plt.ylabel("Available Bike Stands")
    plt.title("Bike Stand Availability with Trend Lines")
    plt.xticks(rotation=45)
    plt.legend()
    plt.tight_layout()
    plt.show()

    plt.figure(figsize=(12, 8))

    # Plotting each period
    plt.plot(pre_pandemic['TIME'], pre_pandemic['AVAILABLE_BIKE_STANDS'], c='darkblue', label="Pre-pandemic")
    plt.plot(pandemic['TIME'], pandemic['AVAILABLE_BIKE_STANDS'], c='forestgreen', label="During Pandemic")
    plt.plot(post_pandemic['TIME'], post_pandemic['AVAILABLE_BIKE_STANDS'], c='crimson', label="Post-pandemic")

    # Highlighting highs and lows during the pandemic
    max_availability = pandemic['AVAILABLE_BIKE_STANDS'].max()
    min_availability = pandemic['AVAILABLE_BIKE_STANDS'].min()
    high_dates = pandemic[pandemic['AVAILABLE_BIKE_STANDS'] == max_availability]['TIME']
    low_dates = pandemic[pandemic['AVAILABLE_BIKE_STANDS'] == min_availability]['TIME']

    plt.scatter(high_dates, [max_availability] * len(high_dates), c='gold', label='Highs during Pandemic')
    plt.scatter(low_dates, [min_availability] * len(low_dates), c='purple', label='Lows during Pandemic')

    # Adding labels, title, and legend
    plt.xlabel("Date")
    plt.ylabel("Available Bike Stands")
    plt.title("Bike Stand Availability with Pandemic Highs and Lows Highlighted")
    plt.xticks(rotation=45)
    plt.legend()
    plt.tight_layout()
    plt.show()


# Main function
def main():
    pre_pandemic, pandemic, post_pandemic = load_processed_data()
    plot_bike_stand_availability_with_trend_lines(pre_pandemic, pandemic, post_pandemic)

if __name__ == "__main__":
    main()
```

MODEL FILE

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.linear_model import Ridge
```

```
import matplotlib.pyplot as plt
import math
from sklearn.model_selection import TimeSeriesSplit


def one_step_ahead(y: pd.Series, lag: int, dt: float, alpha: float, trends: [str], stride: int) -> float:
    q = 1

    # Initialise sample numbers for different periods
    hy = math.floor(27 * 7 * 24 * 60 * 60 / dt)  # Number of samples in a half year
    m = math.floor(4 * 7 * 24 * 60 * 60 / dt)  # Number of samples in a month
    w = math.floor(7 * 24 * 60 * 60 / dt)  # Number of samples in a week
    d = math.floor(24 * 60 * 60 / dt)  # Number of samples in a day

    # Determine the length based on specified trends
    x = hy if "half-year" in trends else (m if "month" in trends else (w if "week" in trends else d))
    length = y.size - x - (lag * x) - q

    # Early return conditions
    if length <= 0 or not trends:
        return np.nan

    # Initialize model features
    XX = y[q:q + length:stride].values.reshape(-1, 1)

    # Add lagged features based on specified trends
    for trend in trends:
        for i in range(lag):
            if trend == "half-year":
                X = y[i * hy + q:i * hy + q + length:stride].values.reshape(-1, 1)
            elif trend == "month":
                X = y[i * m + q:i * m + q + length:stride].values.reshape(-1, 1)
            elif trend == "week":
                X = y[i * w + q:i * w + q + length:stride].values.reshape(-1, 1)
            elif trend == "short-term":
                X = y[i:i + length:stride].values.reshape(-1, 1)
            XX = np.column_stack((XX, X))

    # Format output feature
    yy = y[lag * x + x + q:lag * x + x + q + length:stride].values

    # Train the model
    model = Ridge(fit_intercept=False, alpha=alpha, random_state=42).fit(XX, yy)

    # Return the model's prediction for the next value
    return model.predict(XX)[-1]


# Ensure this function is defined before it's called
def read_daily_data() -> (pd.DataFrame, pd.DataFrame, pd.DataFrame):
    pre_pandemic = pd.read_csv("data/daily_pre-pandemic.csv", parse_dates=['TIME'])
    pandemic = pd.read_csv("data/daily_pandemic.csv", parse_dates=['TIME'])
    post_pandemic = pd.read_csv("data/daily_post-pandemic.csv", parse_dates=['TIME'])
    return pre_pandemic, pandemic, post_pandemic

def cross_validate_model(data: pd.DataFrame, dt: float, alpha: float, trends: [str], stride: int):
    tscv = TimeSeriesSplit()
    errors = []

    for train_index, test_index in tscv.split(data):
        train, test = data.iloc[train_index], data.iloc[test_index]
```

```
        y_train = train['AVAILABLE_BIKE_STANDS']
        y_test = test['AVAILABLE_BIKE_STANDS']

        predictions = []
        for _ in range(len(test)):
            pred = one_step_ahead(y_train, 3, dt, alpha, trends, stride)
            y_train[len(y_train)] = pred
            predictions.append(pred)

        error = np.mean((y_test - predictions) ** 2)  # Mean Squared Error
        errors.append(error)

    return np.mean(errors)

def multi_step_predict(data: pd.DataFrame, dt: float, steps: int, alpha: float, trends: [str], stride: int):
    y = data['AVAILABLE_BIKE_STANDS']
    predictions = []

    for _ in range(steps):
        new_y = one_step_ahead(y, 3, dt, alpha, trends, stride)
        y[len(y)] = new_y
        predictions.append(new_y)

    return predictions


def main():
    pre_pandemic, pandemic, post_pandemic = read_daily_data()
    t_full = pd.array(pd.DatetimeIndex(pre_pandemic.iloc[:, 1]).astype(np.int64)) / 1000000000
    dt = t_full[1] - t_full[0]

    alpha = 1.0
    trends = ["week", "short-term"]
    stride = 1

    # Cross-validation on pre-pandemic data
    cv_error = cross_validate_model(pre_pandemic, dt, alpha, trends, stride)
    print(f"Cross-validation error (pre-pandemic): {cv_error}")

    # Cross-validation on post-pandemic data
    cv_error_post_pandemic = cross_validate_model(post_pandemic, dt, alpha, trends, stride)
    print(f"Cross-validation error (post-pandemic): {cv_error_post_pandemic}")

    # Multi-step prediction for pandemic period
    pandemic_predictions = multi_step_predict(pre_pandemic, dt, len(pandemic), alpha, trends, stride)
    pandemic_predicted_series = pd.Series(pandemic_predictions, index=pandemic['TIME'])

    # Multi-step prediction for post-pandemic period
    combined_data = pd.concat([pre_pandemic, pandemic])
    post_pandemic_predictions = multi_step_predict(combined_data, dt, len(post_pandemic), alpha, trends, stride)
    post_pandemic_predicted_series = pd.Series(post_pandemic_predictions, index=post_pandemic['TIME'])


    # Plot the actual data and the predictions
    plt.figure(figsize=(15, 8))
    plt.scatter(pre_pandemic['TIME'], pre_pandemic['AVAILABLE_BIKE_STANDS'], color='black', marker='.', label="Actual
Pre-Pandemic Data")
    plt.scatter(pandemic['TIME'], pandemic['AVAILABLE_BIKE_STANDS'], color='gray', marker='.', label="Actual Pandemic
Data")
    plt.scatter(post_pandemic['TIME'], post_pandemic['AVAILABLE_BIKE_STANDS'], color='lightgray', marker='.',
label="Actual Post-Pandemic Data")
```

```
    plt.plot(pandemic_predicted_series.index, pandemic_predicted_series, color='blue', linestyle='-', label="Predict Bike usage (Pre-Pandemic Data)")
    plt.plot(post_pandemic_predicted_series.index, post_pandemic_predicted_series, color='red', linestyle='-', label="Predict Bike usage (Post-Pandemic Data)")
    plt.xlabel("Time")
    plt.ylabel("Available Bike Stands")
    plt.title("Bike Stand Availability: Actual vs Predicted")
    plt.legend()
    plt.xticks(rotation=45)
    plt.tight_layout()
    plt.show()


if __name__ == "__main__":
    main()
```