Politecnico di Torino

ICT for smart mobility
- Laboratory 2: ARIMA Model -
ICT for Smart Societies

Gavazzi Federica 306032
Pergolizzi Alessandro 318945
Sulas Claudio 318949

Academic Year: 2023-2024

Contents

Chapter 1

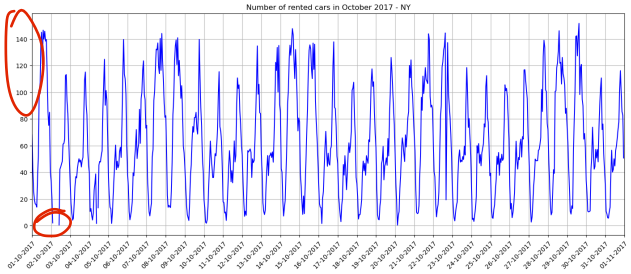
Prediction using ARIMA models

1.1 Task 1

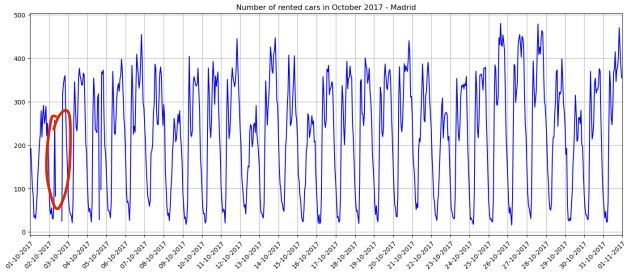
By using the time series obtained in the previous laboratory for Madrid, NYC and Torino, the objective of this laboratory is to predict car rental occurrences through the application of an ARIMA model.

1) For each city, consider the selected period of 30 days. Extract the number of rentals recorded at each hour- after properly filtering the outliers and those bookings that are not rentals.
The time series for each of the three cities comprises counts of rentals for each hour throughout the specified period. The data is derived from queries to the PermanentBookings tables in the database, specifically covering the month of October 2017. This timeframe selection aims to avoid the inclusion of anomalous periods, such as the Christmas holidays. The applied filter eliminates outliers by excluding rentals lasting less than 5 minutes or exceeding 3 hours. Additionally, only bookings with distinct starting and arrival points are considered for analysis.
The python code can be found at 2.2.1 in the appendix part.

2) Fitting Missing Data.
To employ an ARIMA model for predictions, it is necessary to conduct some preprocessing on the data. Initially, a degree of randomness is introduced into the data to facilitate working with floating values. Subsequently, an assessment is made to determine if the time series are continuous or if there are any missing data points. To achieve this, individual plots for each city are generated, as depicted in Figure 1.1. Notably, for NYC and Torino, observable gaps in values are evident for various hours between the 2nd and the 3rd of October. For the python code, refer to 2.2.2 in the appendix part.



(a) NY time series- missing points

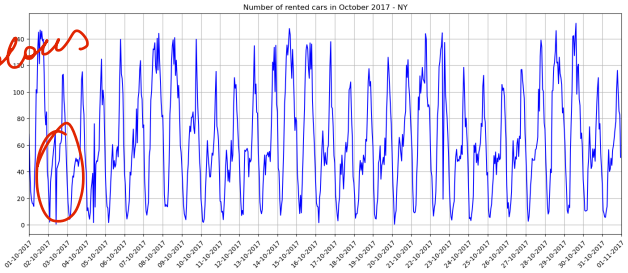


(b) Madrid time series- missing points

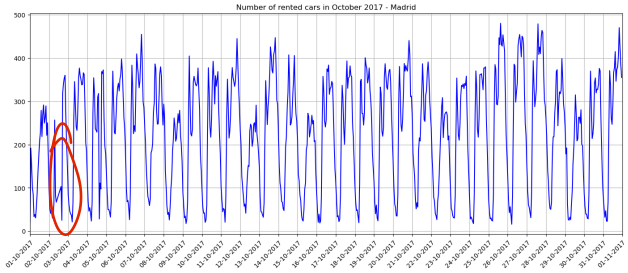
Figure 1.1: Missing Points in the time series

To address missing values in the time series data, a linear regression approach is applied. This involved augmenting the original time series with features such as day of the week (day) and hour of the day (hour). A linear regression model is trained on the original data to predict missing values in the time series. The outputted values are then inserted into the original time series. The results are shown in figures 1.2, while the Python code can be found in the appendix 2.2.3.

3) Check Stationarity.
In order to determine the stationarity of the time series, a reliable method is to examine the rolling statistics, specifically the mean and standard deviation. In Figure 1.3, focusing on NYC rentals, the green and red lines represent the rolling mean and standard deviation, respectively. As observed, both metrics remain relatively constant throughout the specified time range. Comparable outcomes are



(a) NY time series



(b) Madrid time series

Figure 1.2: Time series after LLS

also observed for Madrid and Torino, with reference graphs available in Section 2.3.
Given the absence of any discernible trends in the time series for all three cases, there is no necessity for detrending or differencing transformations. Thus, the number of times the raw observations need to be differenced is set to 0.

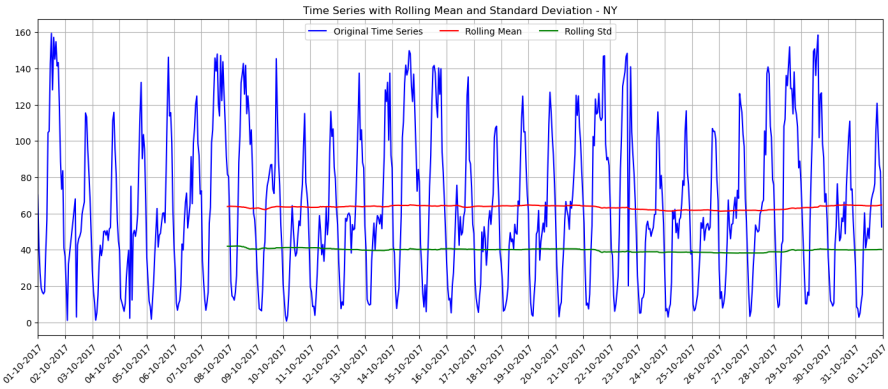
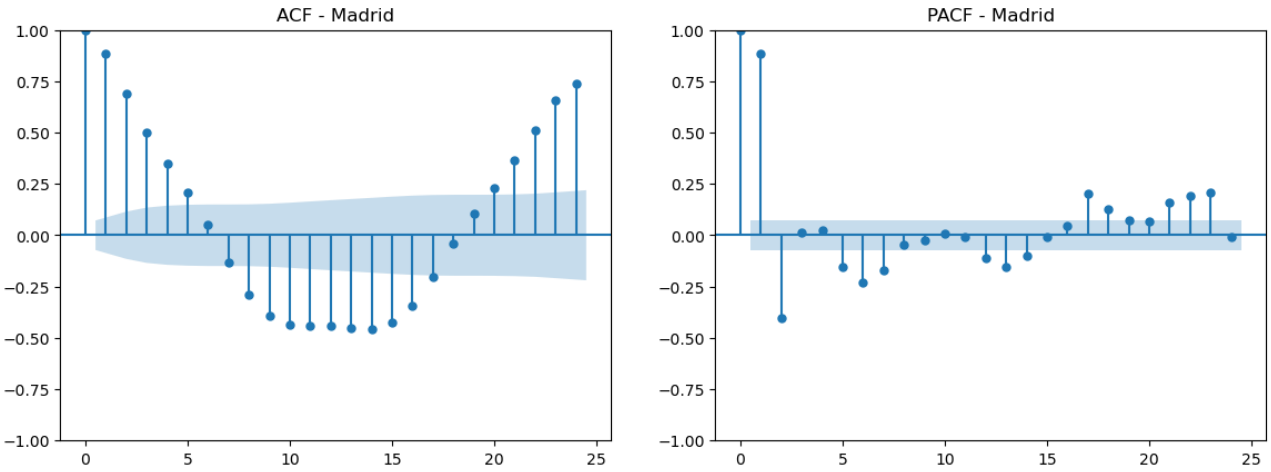


Figure 1.3: NY rolling mean & std

By setting d=0, an ARMA model is employed, necessitating the selection of the two hyperparameters 'p' and 'q' based on the characteristics of each case. The assessment of rolling metrics is conducted using a time window of one week in October 2017. For the python code refers to the section in the appendix 2.2.4.

4) Compute the ACF and PACF.
Both Autocorrelation Function (ACF) and Partial Autocorrelation Function (PACF) analyses play a crucial role in the exploratory data analysis of time series samples. These analyses aid in identifying patterns and assessing the presence of randomness in the data. The ACF indicates the direct correlation between past values and current observations, while the PACF identifies the correlation of residuals with the next lag value.
In case the model were purely AR, the ACF plot would exhibits a slow decay after 'q' lags. At the same time, the Partial Autocorrelation Function (PACF) would present significant spikes for the initial 'p' lags, followed by a rapid decay at 0. Similar reasoning can be applied to check whether the model is purely Moving Average (MA). Examining the plots below 1.4, if the model was purely MA, one would expect the PACF to slowly decay, while in the ACF, there should be significant peaks only initially, rapidly decreasing to 0.
Observing Figure 1.4a, the AutoCorrelation Function (ACF) plot for Madrid rentals within a 24-hour time window is presented. A sinusoidal trend fluctuating between approximately 0.75 and -0.5 indicates an opposite correlation among the values during nighttime and daytime, aligning with expectations. The samples with 24 hours apart have instead a high correlation among them.
In the event that the ACF plot slowly decreases and the PACF becomes negligible after 'p' lags, the model could be considered a pure AutoRegressive (AR) model. However, as evident also in Figure 1.4b, there are no significant spikes 'p' in PACF followed by rapid decay at 0. Since this is not the



(a) Madrid ACF

(b) Madrid PACF

Figure 1.4: Madrid ACF and PACF

case for any of the three cities, the model is not purely AR.
Concerning Moving Average (MA) model, one would expect to see in ACF figure 1.4a a trend rapidly decaying to 0 with very few values not negligible, and a certain number of 'p' lags in figure 1.4b before becoming negligible. However, as observed in all three case studies (see section 2.1.3, this pattern is not evident. Consequently, the models can be considered ARMA models. For the python code, refer to 2.2.5.

5) Setting window length N.
To train the model, a period of 31 days is segmented, with the second and third weeks allocated for training and the fourth week designated for testing. This results in a total of N=336 samples (hours) for the training set and 168 samples (hours) for the test set. For the initial learning strategy, the 'sliding window' method is employed. Essentially, following the prediction of each sample (corresponding to the number of rentals in an hour of the day), the training set is shifted forward by 1 lag. Simultaneously, past data is discarded, maintaining a fixed window length. A similar approach is applied to the test set, where the fixed window length is shifted forward by 1 sample (hour).
This methodology ensures that the model remains updated with the latest values while discarding older ones that may no longer accurately reflect the current behavior.

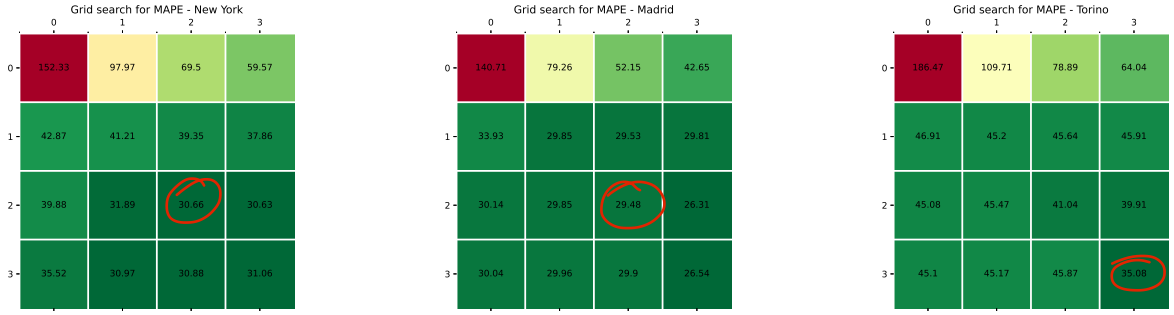
6-7.a) Model Training and Error Computation
To determine the optimal triplet for the hyperparameters, it is essential to assess various metrics for comparison. With the number of past samples (N) fixed, the next step involves selecting values for 'p' and 'q'. To achieve this, a grid search is conducted, comparing different combinations of 'p' and 'q' by visualizing prediction errors such as MAPE and MPE.
The Mean Absolute Percentage Error (MAPE), evaluated using the formula 1.1, indicates the effectiveness of the estimation achieved with a specific combination of hyperparameters. A smaller MAPE value implies better performance of the ARIMA model. This metric proves particularly valuable when working with data of variable scale or when assessing the percentage accuracy of predictions in relation to actual values.

$$MAPE = \frac{1}{n}\sum_{i=1}^{n}\left|\frac{A_i - F_i}{A_i}\right| \times 100\% \tag{1.1}$$

Another prediction error used is the MPE, or Mean Percentage Error. It is commonly used in time series analysis to evaluate the accuracy of forecasting models. Like the MAPE, the MPE provides a measure of the average percentage deviation between predicted and actual values, as it shows formula 1.2.

$$MPE = \frac{1}{n}\sum_{i=1}^{n}\left(\frac{A_i - F_i}{A_i}\right) \times 100\% \tag{1.2}$$

Once the grid search is done, the results are presented in Figure 1.5 for all three cities. Refer to Section 2.2.6 for the reference code.



(a) NY Grid Search (b) Madrid Grid Search (c) Torino Grid Search

Figure 1.5: Grid Search

Upon examination of the matrices, the best parameters within the chosen range of 0-4 are the following ones:
- Torino Enjoy: (3, 0, 3) - Madrid: (2, 0, 3) - New York City: (2, 0, 3)
The results of all predictions in comparison to the True values (depicted by the black line) are illustrated in Figure 1.6 for Madrid. To view the graphs for NYC and Torino, refer to section 2.1.4.

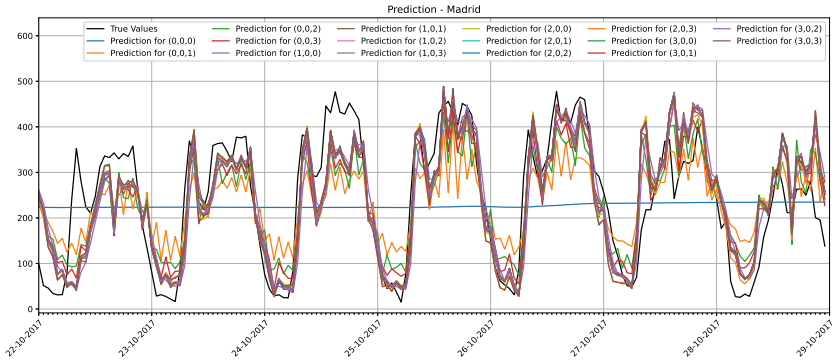


Figure 1.6: Prediction for Madrid - 3 weeks

7.b- 7.c) Train with different window size N and compare the results

Following the selection of the optimal triplets of hyperparameters for the ARIMA models, the window size for the training set is adjusted to assess the impact of N on predictions. The time interval for N spans from 1 day to 3 weeks of training data, while the test set has been kept fixed to 1 week (the last one, as before).

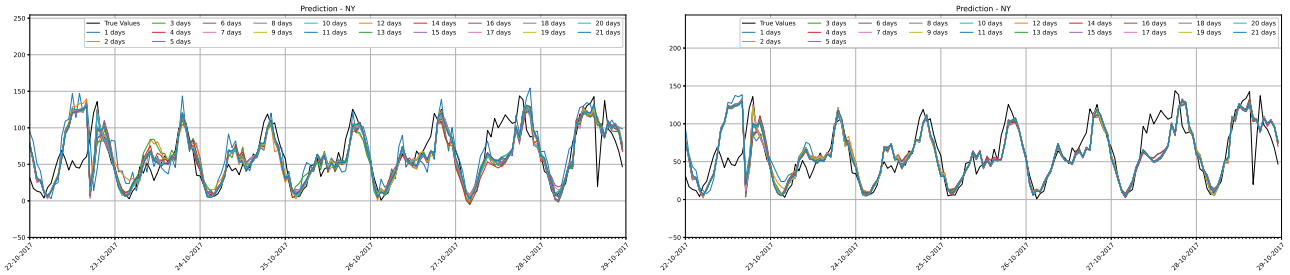
To compare results, two different learning strategies are employed. The first is the sliding window, and the second is the expanding window strategy. In the expanding window strategy, the model is trained by progressively increasing the subset of data at each step.

At the end, the training set in the expanding window strategy will encompass more data compared to the sliding window case. This can be both advantageous and disadvantageous. On one hand, the model has more samples for training, but on the other hand, many of these samples reflect past behaviors that may not be the most accurate descriptors of the current time.

To assess the performance of both strategies, the Mean Absolute Percentage Error (MAPE) prediction error is computed, keeping the parameters 'p' and 'q' fixed with the best results obtained as mentioned earlier.

In Figure 1.7, the predictions for New York, obtained with the two learning strategies, demonstrate similar performance. Similar results were obtained for Madrid and Torino, where the plots can be

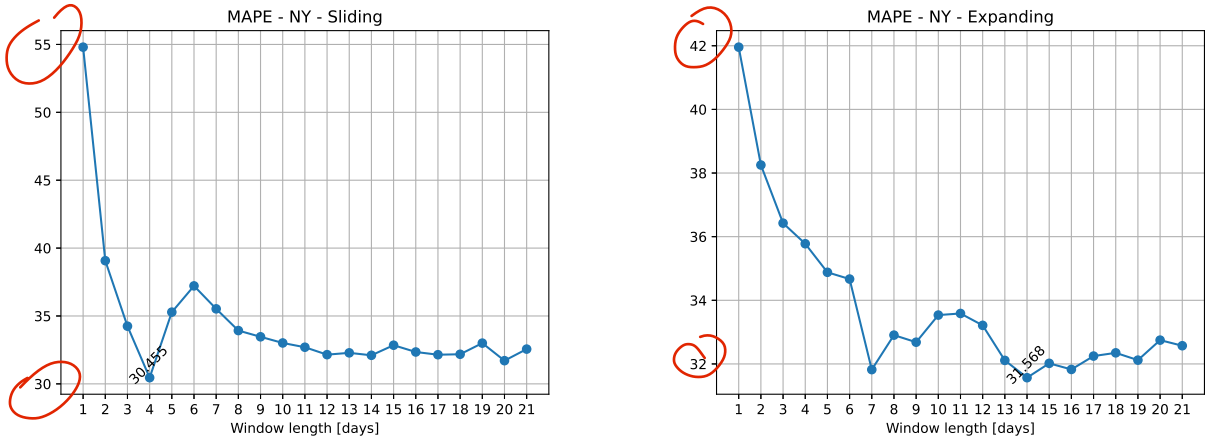
found in the appendix in Section 2.1.5. The only difference can be seen for big values of N; using too many points will worsen the results when using the expanding window. This ties in with what was said earlier concerning past samples that may not be able to reflect the present behaviour.



(a) NYC Prediction-Sliding Window (b) NYC Prediction - Expanding Window

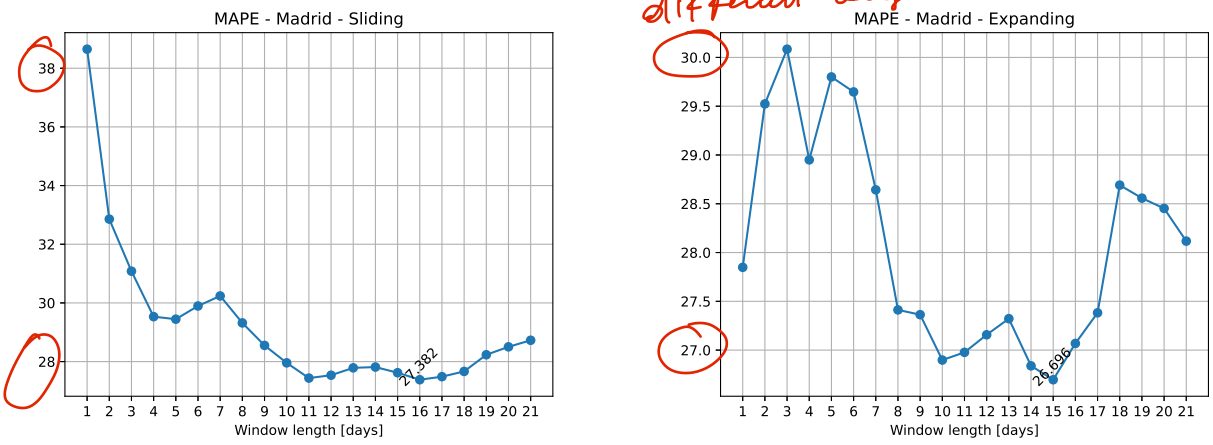
Figure 1.7: NYC Predictions

The differing factor in the obtained results for the cities lies in the MAPE values. Figure 1.8 presents the MAPE errors for NYC and Madrid (the results for Torino can be found in the appendix at Section 2.1.6). Particularly in the expanding window case, the cities exhibit similar trends only for values of N ranging from 10 to 17 days of samples. Consequently, it can be inferred that the best results can be achieved by using a time window of approximately 2 weeks. Outside this range, only NYC shows a relatively good development.



(a) NYC MAPE - Sliding Window (b) NYC MAPE - Expanding Window

Figure 1.8: NYC MAPE for Sliding and Expanding windows

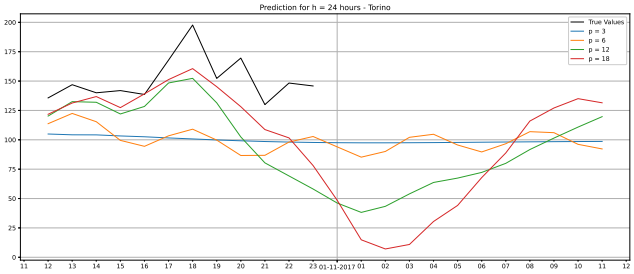


(a) Madrid MAPE - Sliding Window (b) Madrid MAPE - Expanding Window

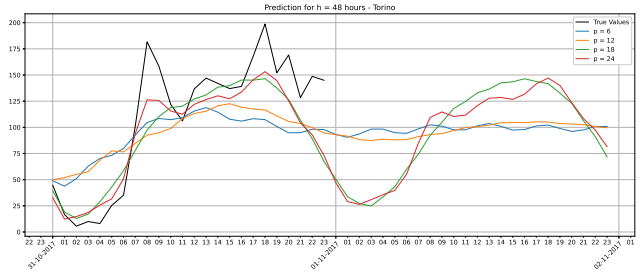
Figure 1.9: Madrid MAPE for Sliding and Expanding Window

The results of these analyses revealed that for Madrid the best results are with an ARIMA model of order (2, 0, 3) and an expanding window of length starting from 15 days, for New York City an ARIMA model of order (2, 0, 3) and sliding window of length of 4 days. Finally, for Torino Enjoy an ARIMA model of order (3, 0, 3) and expanding window of length starting from 17 days. The plots reveal that New York City exhibits a distinct trend in the expanding window behavior. This difference can be attributed to varying car usage patterns, likely influenced by cultural differences between American and European cities, influencing the habits of the residents.
For the reference code, see section 2.2.8 and section 2.2.9.

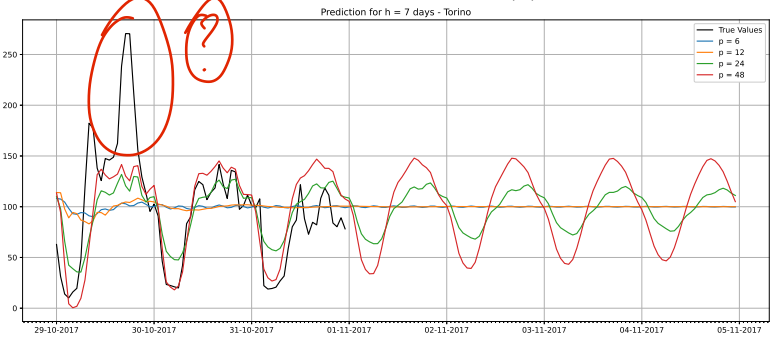
8 - (optional) Change the time horizon 'h'.
The time horizon 'h' indicates how much in the future the model should predict in terms of number of samples. Up until now, the time horizon was set to 1. Therefore, every time a new model was created (with a new training set), a single future sample was predicted.
By changing 'h', the future predictions will gradually converge towards the average value of the time series. If the model is more complex, for example, by increasing the hyperparameter p, this convergence will occur further into the future. As depicted in Figure 1.10, the higher p is, the better the predictions are at the beginning. Higher p-values will start converging to the dataset's mean value after some time, while smaller p-values will converge to the average faster. This expected outcome arises because having a higher number of past samples for forecasting leads to more accurate predictions, even in the more distant future.
Figures 1.10b, 1.10c, and 1.10a show predictions with time horizons of 48 hours, 24 hours, and 7 days, respectively. The plots for Madrid and NYC with the same values for h can be found in the appendix in Section 2.1.7.



(a) Torino 24 hours of time horizon



(b) Torino 48 hours of time horizon



(c) Torino 7 days of time horizon

Figure 1.10: Torino time horizon

In any case, it is expected in the long-term predictions, even with high values of 'p' and 'q', may result in poor accuracy. Therefore, ARIMA models are preferable when making short-term predictions, where only the upcoming hours can be considered as reliable estimations.
For the reference code, see the appendix in section 2.3.1.

Chapter 2

Appendix

2.1 Plots

2.1.1 Point 2 - Fitting Missing Data

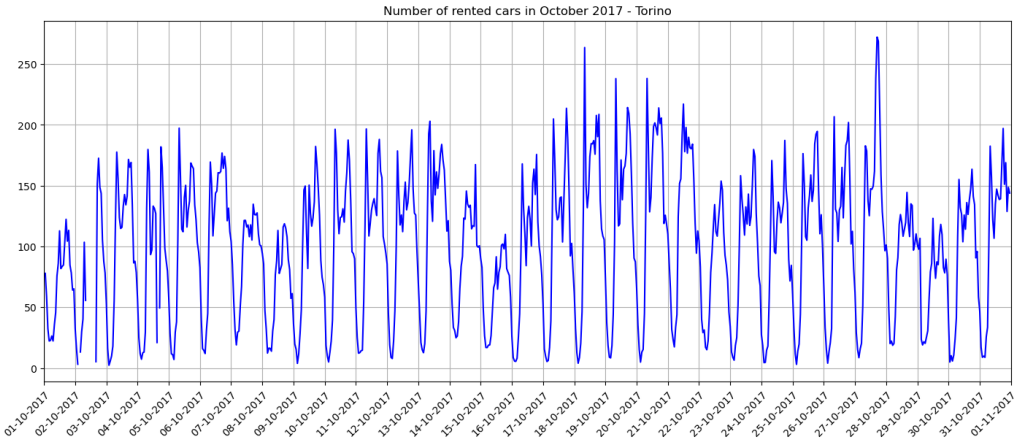


Figure 2.1: Torino time series- missing points

2.1.2 Point 3 - checking stationarity

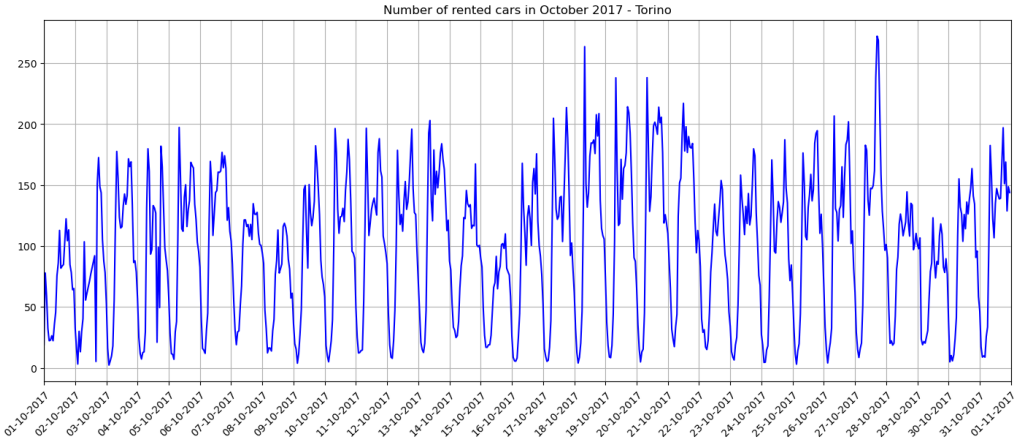
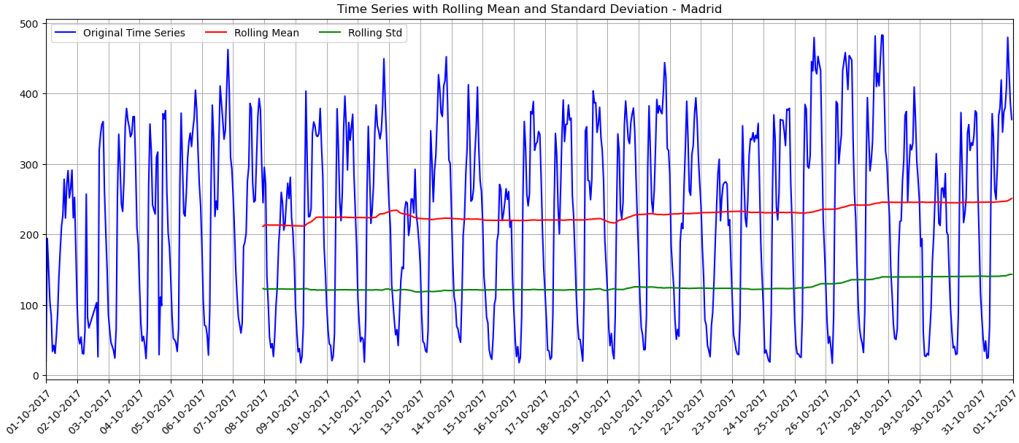
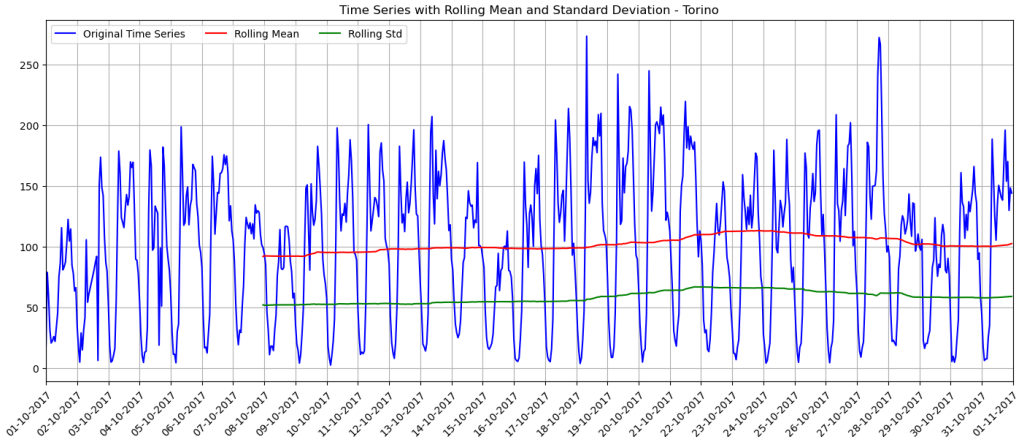


Figure 2.2: Torino time series



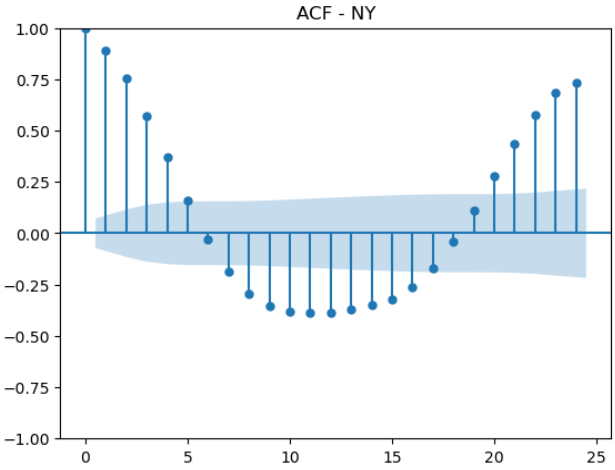
(a) Madrid rolling mean & std



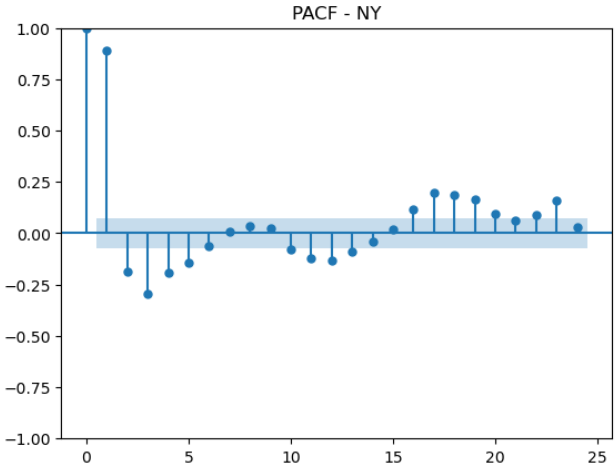
(b) Torino rolling mean & std

Figure 2.3: Rolling mean & standard deviations

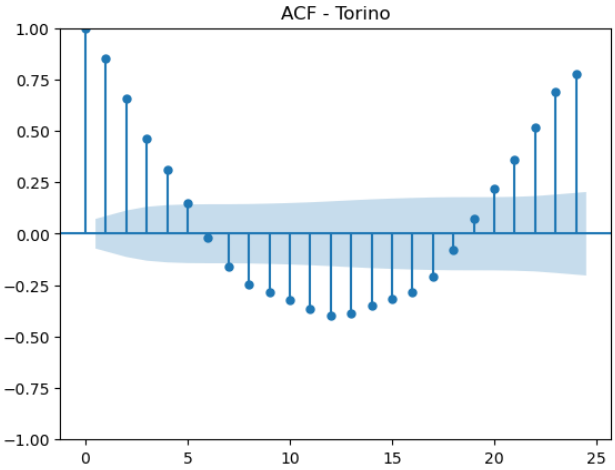
2.1.3 Point 4 - ACF & PACF



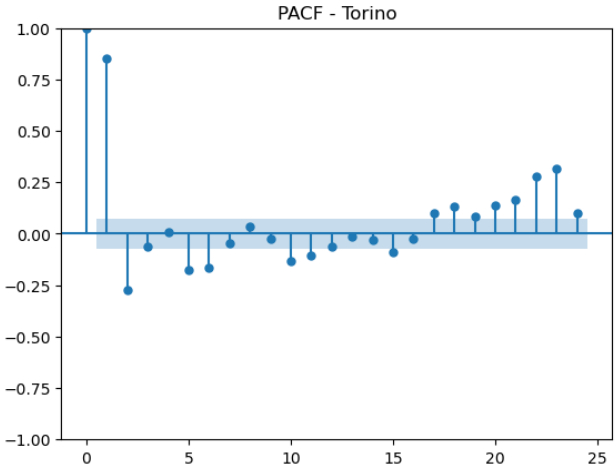
(a) NYC ACF



(b) NYC PACF



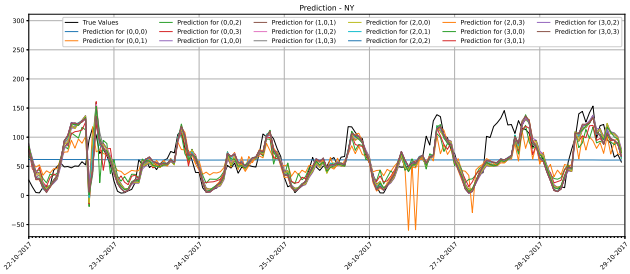
(c) Torino ACF



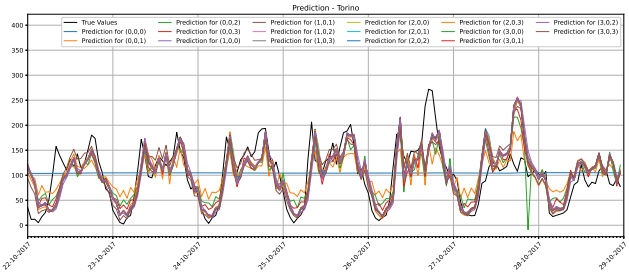
(d) Torino PACF

Figure 2.4: ACF & PACF Torino and NYC

2.1.4 Point 6/7.a

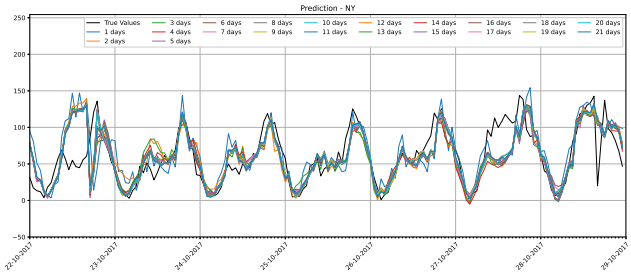


(a) NYC Predictions

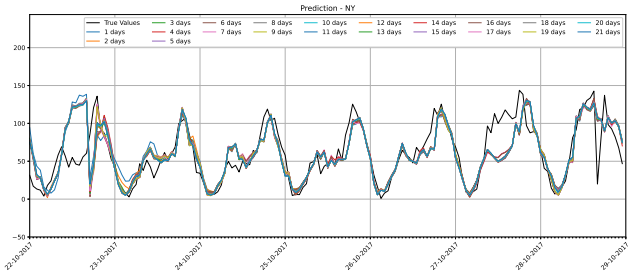


(b) Torino Prediction

2.1.5 Point 7.b

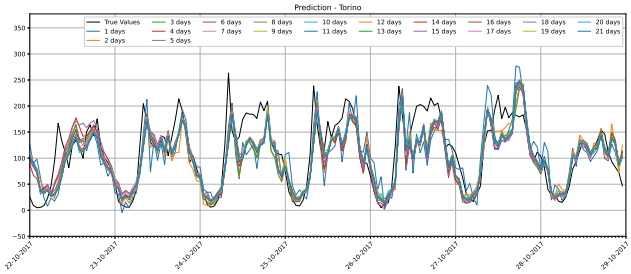


(a) NYC sliding window

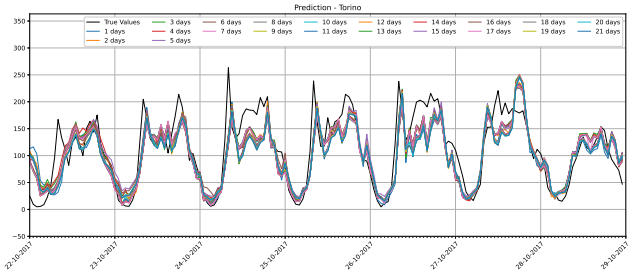


(b) NYC Expanding Window

Figure 2.6: NYC learning strategies Predictions



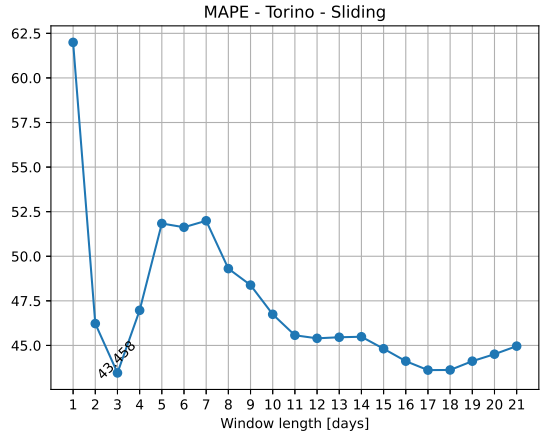
(a) Torino sliding window



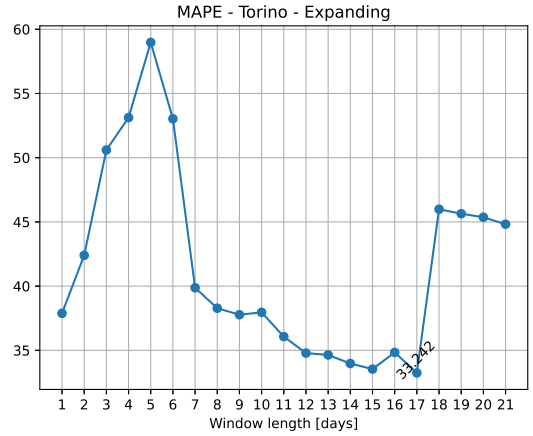
(b) Torino Expanding Window

Figure 2.7: Torino learning strategies

2.1.6 Point 7.b- MAPE



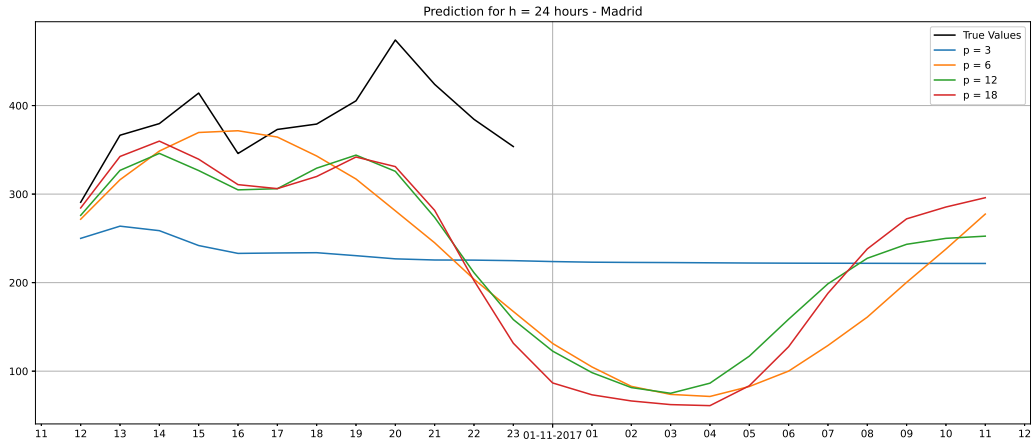
(a) Torino- Sliding Window



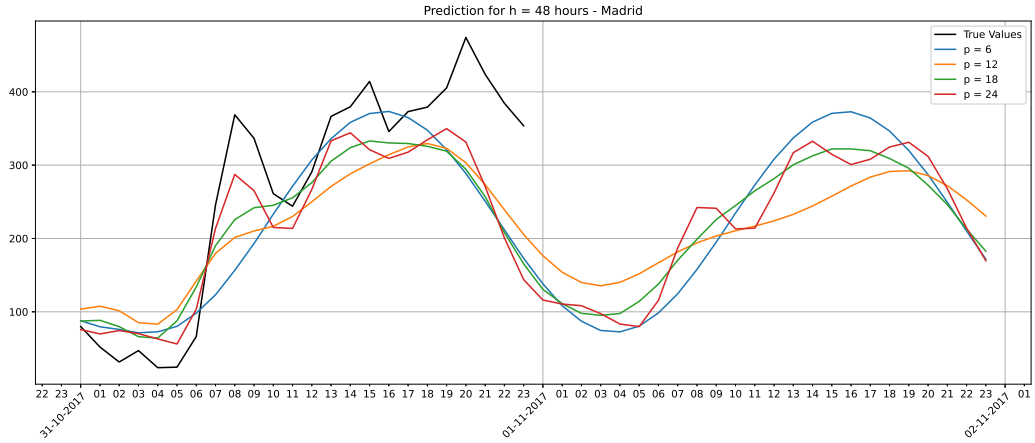
(b) Torino Expanding Window

Figure 2.8: Torino MAPE evaluation

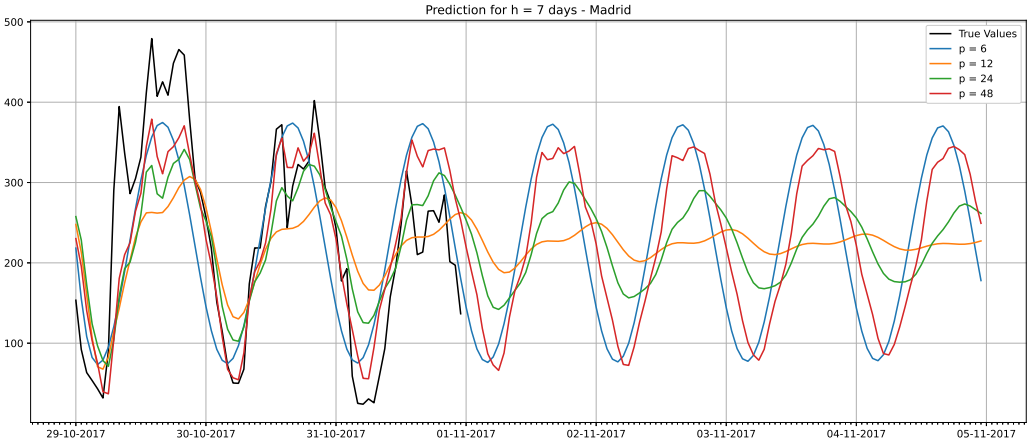
2.1.7 Point 8 optional



(a) Madrid- time horizon 24 hours

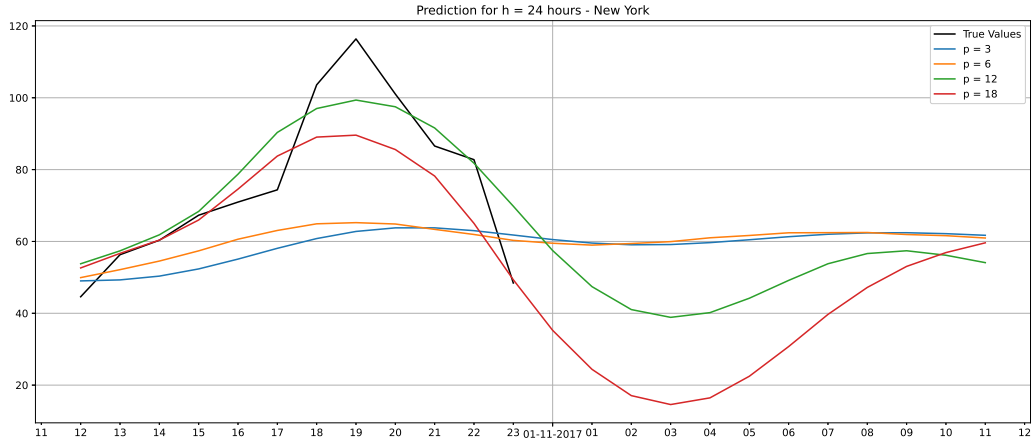


(b) Madrid - time horizon 48 hours

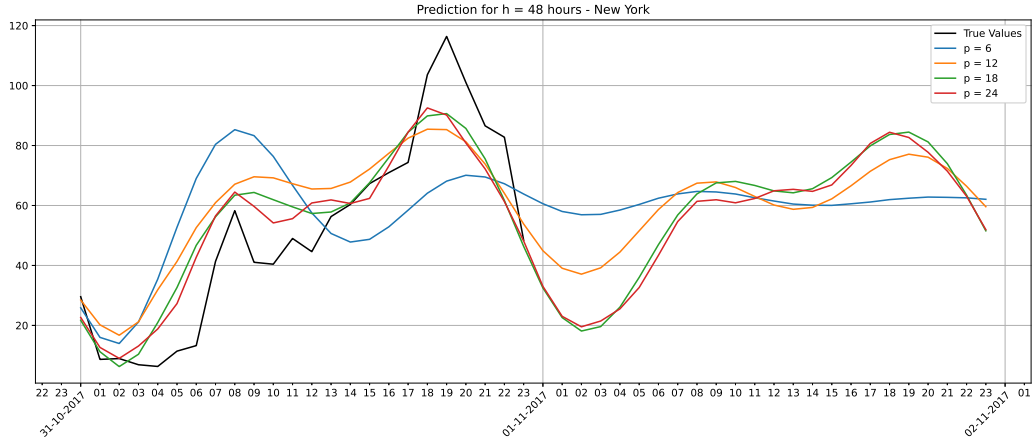


(c) Madrid - time horizon 7 days

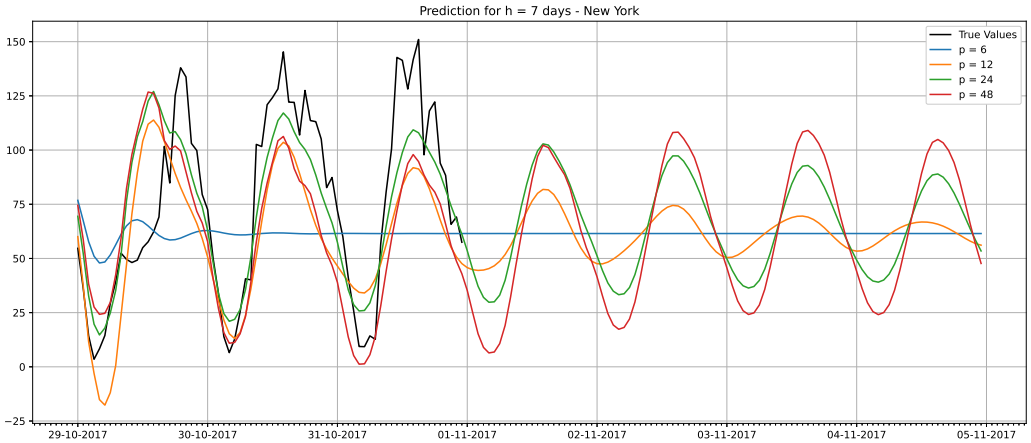
Figure 2.9: Madrid - different time horizon



(a) NYC - time horizon 24 hours



(b) NYC - time horizon 48 hours



(c) NYC - time horizon 7 days

Figure 2.10: NYC - different time horizon

2.2 Code Step 1

2.2.1 Filter for time series

```
collections = [
"ActiveBookings",
"ActiveParkings",
"PermanentBookings",
"PermanentParkings",
"enjoy_ActiveBookings",
"enjoy_ActiveParkings",
"enjoy_PermanentBookings",
"enjoy_PermanentParkings",
]
car2go_city_list = ["Madrid", "New York City"]
enjoy_city_list = ["Torino"]
car2go_collections = ["PermanentBookings", "PermanentParkings"]
enjoy_collections = ["enjoy_PermanentBookings", "enjoy_PermanentParkings"]
    )

filtered_booking_Madrid_query = db.PermanentBookings.aggregate(
[
    {
        "$match": {
            "city": "Madrid",
            "init_time": {"$gte": init_time - 3600, "$lt": final_time - 3600},
        }
    },
    {
        "$project": {
            "_id": 0,
            "init_address": 1,
            "final_address": 1,
            "date": {
                "$dateFromString": {
                    "dateString": {
                        "$dateToString": {
                            "date": "$init_date",
                            "format": "%d-%m-%Y-%H",
                        }
                    }
                }
            },
            "duration": {
                "$divide": [
                    {"$subtract": ["$final_time", "$init_time"]},
                    60,
                ]
            },
            "equal": {"$eq": ["$init_address", "$final_address"]},
        }
    },
    {
        "$match": {
            "equal": False,
            "duration": {
                "$gte": 5,
                "$lte": 180,
            },
        }
```

```
    },
    {"$group": {"_id": "$date", "count": {"$sum": 1}}},
    {"$project": {"_id": 0, "date": "$_id", "count": 1}},
    {"$sort": {"date": 1}},
]
)

filtered_booking_New_York_City_query = db.PermanentBookings.aggregate(
    [
        {
            "$match": {
                "city": "New York City",
                "init_time": {
                    "$gte": init_time + 4 * 3600,
                    "$lt": final_time + 4 * 3600,
                },
            }
        },
        {
            "$project": {
                "_id": 0,
                "init_address": 1,
                "final_address": 1,
                "date": {
                    "$dateFromString": {
                        "dateString": {
                            "$dateToString": {
                                "date": "$init_date",
                                "format": "%d-%m-%Y-%H",
                            }
                        }
                    }
                },
                "duration": {
                    "$divide": [
                        {"$subtract": ["$final_time", "$init_time"]},
                        60,
                    ]
                },
                "equal": {"$eq": ["$init_address", "$final_address"]},
            }
        },
        {
            "$match": {
                "equal": False,
                "duration": {
                    "$gte": 5,
                    "$lte": 180,
                },
            }
        },
        {"$group": {"_id": "$date", "count": {"$sum": 1}}},
        {"$project": {"_id": 0, "date": "$_id", "count": 1}},
        {"$sort": {"date": 1}},
    ]
)

filtered_booking_Torino_query = db.PermanentBookings.aggregate(
    [
        {
            "$match": {
```

```
                "city": "Torino",
                "init_time": {"$gte": init_time - 3600, "$lt": final_time - 3600},
            }
        },
        {
            "$project": {
                "_id": 0,
                "init_address": 1,
                "final_address": 1,
                "date": {
                    "$dateFromString": {
                        "dateString": {
                            "$dateToString": {
                                "date": "$init_date",
                                "format": "%d-%m-%Y-%H",
                            }
                        }
                    }
                },
                "duration": {
                    "$divide": [
                        {"$subtract": ["$final_time", "$init_time"]},
                        60,
                    ]
                },
                "equal": {"$eq": ["$init_address", "$final_address"]},
            }
        },
        {
            "$match": {
                "equal": False,
                "duration": {
                    "$gte": 5,
                    "$lte": 180,
                },
            }
        },
        {"$group": {"_id": "$date", "count": {"$sum": 1}}},
        {"$project": {"_id": 0, "date": "$_id", "count": 1}},
        {"$sort": {"date": 1}},
    ]
)

filtered_booking_Madrid = []
filtered_book_days_Madrid = []
for element in filtered_booking_Madrid_query:
    filtered_book_days_Madrid.append(element["date"])
    filtered_booking_Madrid.append(element["count"])

filtered_booking_New_York_City = []
filtered_book_days_New_York_City = []
for element in filtered_booking_New_York_City_query:
    filtered_book_days_New_York_City.append(element["date"])
    filtered_booking_New_York_City.append(element["count"])

filtered_booking_Torino = []
filtered_book_days_Torino = []
for element in filtered_booking_Torino_query:
    filtered_book_days_Torino.append(element["date"])
    filtered_booking_Torino.append(element["count"])
```

2.2.2 Plots for time series

```
def visualize_missing_values(df):
df = df.resample("1H").asfreq()
range_date = pd.date_range(
    start="2017-10-01 00:00:00", end="2017-10-31 23:00:00", freq="1H"
)
df = df.reindex(range_date, fill_value=np.nan)
return df


def plot_df(dataset, title):
    fig, ax = plt.subplots(layout="constrained", figsize=(14, 6))
    ax.plot(dataset, color="blue")
    ax.xaxis.set_major_locator(md.DayLocator(interval=1))
    ax.xaxis.set_major_formatter(md.DateFormatter("%d-%m-%Y"))
    # ax.xaxis.set_minor_formatter(md.DateFormatter("%d-%m-%Y"))
    plt.xticks(rotation=45, ha="right", rotation_mode="anchor", fontsize=10)
    plt.xlim(
        [
            dt.datetime.strptime("2017-10-01 00", format("%Y-%m-%d %H")),
            dt.datetime.strptime("2017-11-01 00", format("%Y-%m-%d %H")),
        ]
    )
    plt.title(f"Number of rented cars in October 2017 - {title}")
    plt.grid("minor")
    plt.show()


filtered_booking_Madrid = [x + np.random.normal(0, 1) for x in filtered_booking_Madrid]
filtered_booking_New_York_City = [
    x + np.random.normal(0, 1) for x in filtered_booking_New_York_City
]
filtered_booking_Torino = [x + np.random.normal(0, 1) for x in filtered_booking_Torino]

# BOOKINGS
df_booking_madrid = pd.DataFrame(
    {"date": filtered_book_days_Madrid, "count": filtered_booking_Madrid}
)

df_booking_ny = pd.DataFrame(
    {"date": filtered_book_days_New_York_City, "count": filtered_booking_New_York_City}
)

df_booking_torino = pd.DataFrame(
    {"date": filtered_book_days_Torino, "count": filtered_booking_Torino}
)

df_booking_madrid.set_index("date", inplace=True, drop=True)
df_booking_ny.set_index("date", inplace=True, drop=True)
df_booking_torino.set_index("date", inplace=True, drop=True)

df_booking_madrid_full = visualize_missing_values(df_booking_madrid)
df_booking_ny_full = visualize_missing_values(df_booking_ny)
df_booking_torino_full = visualize_missing_values(df_booking_torino)

plot_df(df_booking_ny_full, "NY")
plot_df(df_booking_madrid_full, "Madrid")
plot_df(df_booking_torino_full, "Torino")
```

2.2.3 Linear Regression

```
def solveLLS(X, Y):
return np.linalg.inv(X.T @ X) @ (X.T @ Y)


def linearRegression(df_original, df):
    df["day"] = [element.weekday() for element in df.index]
    df["hour"] = [element.hour for element in df.index]
    df_original["day"] = [element.day for element in df_original.index]
    df_original["hour"] = [element.hour for element in df_original.index]

    df_test = df[df["count"].isnull()]
    if df_test.empty:
        return df

    X = df_original[["day", "hour"]].values
    X_norm = (X - X.mean()) / X.std()
    Y = df_original["count"].values
    Y_norm = (Y - Y.mean()) / Y.std()

    w_hat = solveLLS(X_norm, Y_norm[:, np.newaxis])

    res = (df_test[["day", "hour"]].values @ w_hat) * X.std() + X.mean()
    for i in range(len(res)):
        df.at[df_test.index[i], "count"] = res[i]

    return df


df_booking_ny_full = linearRegression(df_booking_ny, df_booking_ny_full)
df_booking_madrid_full = linearRegression(df_booking_madrid, df_booking_madrid_full)
df_booking_torino_full = linearRegression(df_booking_torino, df_booking_torino_full)
```

2.2.4 Check stationarity

```
def see_rolling_mean(dataset, title):
    rolling_mean = dataset["count"].rolling(window=168).mean()
    rolling_std = dataset["count"].rolling(window=168).std()
    fig, ax = plt.subplots(layout="constrained", figsize=(14, 6))
    ax.plot(dataset["count"], color="blue", label="Original Time Series")
    ax.plot(rolling_mean, color="red", label="Rolling Mean")
    ax.plot(rolling_std, color="green", label="Rolling Std")
    ax.xaxis.set_major_locator(md.DayLocator(interval=1))
    ax.xaxis.set_major_formatter(md.DateFormatter("%d-%m-%Y"))
    plt.xticks(rotation=45, ha="right", rotation_mode="anchor", fontsize=10)
    plt.xlim(
        [
            dt.datetime.strptime("2017-10-01 00", format("%Y-%m-%d %H")),
            dt.datetime.strptime("2017-11-01 00", format("%Y-%m-%d %H")),
        ]
    )
    plt.title(f"Time Series with Rolling Mean and Standard Deviation - {title}")
    plt.grid("minor")
    plt.legend(ncols=3)
    plt.show()


print(df_booking_madrid.size)
```

```
    see_rolling_mean(df_booking_ny_full, "NY")
    see_rolling_mean(df_booking_madrid_full, "Madrid")
    see_rolling_mean(df_booking_torino_full, "Torino")

    df_booking_madrid_full.drop(columns=["day", "hour"], inplace=True)
    df_booking_ny_full.drop(columns=["day", "hour"], inplace=True)
    df_booking_torino_full.drop(columns=["day", "hour"], inplace=True)
```

2.2.5 ACF and PACF

```
    from statsmodels.graphics.tsaplots import plot_acf
    from statsmodels.graphics.tsaplots import plot_pacf
    import statsmodels.api as sm


    plot_acf(df_booking_madrid_full["count"], title="ACF - Madrid", lags=24)
    plot_acf(df_booking_ny_full["count"], title="ACF - NY", lags=24)
    plot_acf(df_booking_torino_full["count"], title="ACF - Torino", lags=24)
    plt.show()

    pacf_madrid = plot_pacf(df_booking_madrid_full["count"], title="PACF - Madrid", lags=24)
    pacf_ny = plot_pacf(df_booking_ny_full["count"], title="PACF - NY", lags=24)
    pacf_torino = plot_pacf(df_booking_torino_full["count"], title="PACF - Torino", lags=24)
```

2.2.6 Metrics

```
def evaluate_arima(serie, p_range, q_range, train_len, test_len):
    future = np.zeros((len(p_range), len(q_range), test_len))
    for p in p_range:
        for q in q_range:
            start = 0
            print("Testing ARIMA order (%i,%i,%i)" % (p, d, q))
            while start < test_len:
                train = serie["count"][24 * 7 + start : 24 * 7 + train_len + start]
                arima_model = ARIMA(train, order=(p, d, q))
                m_model = arima_model.fit(
                    method="statespace",
                )
                forecast = m_model.forecast()
                future[p, q, start] = forecast.values[0]
                start += 1
    return future


train_len = 24 * 7 * 2 # 2 weeks
test_len = 24 * 7 # 1 week
d = 0
p_range = np.arange(0, 4)
q_range = np.arange(0, 4)

future_m = evaluate_arima(df_booking_madrid_full, p_range, q_range, train_len, test_len)
future_ny = evaluate_arima(df_booking_ny_full, p_range, q_range, train_len, test_len)
future_t = evaluate_arima(df_booking_torino_full, p_range, q_range, train_len, test_len)

def gridSearch(test, predicted, p_range, q_range):
    mape = np.zeros((len(p_range), len(q_range)))
    mpe = np.zeros((len(p_range), len(q_range)))
```

```
    for p in p_range:
        for q in q_range:
            mape[p, q] = np.mean(np.abs((test - predicted[p, q]) / test)) * 100
            mpe[p, q] = np.mean((test - predicted[p, q]) / test) * 100
    return mape, mpe


madrid_mape, madrid_mpe = gridSearch(
    df_booking_madrid_full["count"][24 * 7 + train_len : 24 * 7 + train_len + test_len],
    future_m,
    p_range,
    q_range,
)
ny_mape, ny_mpe = gridSearch(
    df_booking_ny_full["count"][24 * 7 + train_len : 24 * 7 + train_len + test_len],
    future_ny,
    p_range,
    q_range,
)
torino_mape, torino_mpe = gridSearch(
    df_booking_torino_full["count"][24 * 7 + train_len : 24 * 7 + train_len + test_len],
    future_t,
    p_range,
    q_range,
)
```

2.2.7 Matrix & Prediction Plots

```
dates = pd.date_range(start="2017-10-22 00:00:00", end="2017-10-28 23:00:00", freq="1H")


def plot_predictions(dates, true_values, predicted_values, title, p_range, q_range):
    fig, ax = plt.subplots(layout="constrained", figsize=(14, 6))
    ax.plot(
        dates,
        true_values,
        color="black",
        label="True Values",
    )
    for p in p_range:
        for q in q_range:
            ax.plot(
                dates,
                predicted_values[p, q],
                label=f"Prediction for ({p},{d},{q})",
            )
    ax.xaxis.set_major_locator(md.DayLocator(interval=1))
    ax.xaxis.set_major_formatter(md.DateFormatter("%d-%m-%Y"))
    ax.xaxis.set_minor_locator(md.HourLocator())
    plt.ylim(top=max(np.max(true_values), np.max(predicted_values)) + 150)
    plt.xticks(rotation=45, ha="right", rotation_mode="anchor", fontsize=10)
    plt.xlim(
        [
            dt.datetime.strptime("2017-10-22 00", "%Y-%m-%d %H"),
            dt.datetime.strptime("2017-10-29 00", "%Y-%m-%d %H"),
        ]
    )
    plt.title(f"Prediction with Sliding Window - {title}")
    plt.grid("minor")
```

```
    plt.legend(ncols=6)
    plt.savefig(f"Prediction - {title}.eps", format="eps")
    plt.show()


# Plot for Madrid
plot_predictions(
    dates,
    df_booking_madrid_full["count"][22 * 24 : 29 * 24],
    future_m,
    "Madrid",
    p_range,
    q_range,
)

# Plot for NY
plot_predictions(
    dates,
    df_booking_ny_full["count"][22 * 24 : 29 * 24],
    future_ny,
    "NY",
    p_range,
    q_range,
)

# Plot for Torino
plot_predictions(
    dates,
    df_booking_torino_full["count"][22 * 24 : 29 * 24],
    future_t,
    "Torino",
    p_range,
    q_range,
)

def showMetricMatrix(matrix, title, cmap):
    fig, ax = plt.subplots(layout="constrained")
    ax.matshow(matrix, cmap=cmap)
    ax.set_xticks(np.arange(len(matrix)), np.arange(0, len(matrix)))
    ax.set_yticks(np.arange(len(matrix[0])), np.arange(0, len(matrix[0])))
    ax.set_xticks(np.arange(len(matrix)) - 0.5, minor=True)
    ax.set_yticks(
        np.arange(len(matrix[0])) - 0.5,
        minor=True,
    )
    ax.spines[:].set_visible(False)
    ax.grid(which="minor", color="w", linestyle="-", linewidth=2)
    ax.tick_params(which="minor", bottom=False, left=False)
    for i in range(len(matrix)):
        for j in range(len(matrix[0])):
            text = ax.text(
                j,
                i,
                round(matrix[i, j], 2),
                ha="center",
                va="center",
                color="k",
            )
    plt.title(f"Grid search for {title}")
    plt.savefig(f"GridSearch - {title}.eps", format="eps")
    plt.show()
```

```
showMetricMatrix(madrid_mape, "MAPE - Madrid", "RdYlGn_r")
# showMetricMatrix(madrid_mpe, "MPE - Madrid", "RdYlGn")
showMetricMatrix(ny_mape, "MAPE - New York", "RdYlGn_r")
# showMetricMatrix(ny_mpe, "MPE - New York", "RdYlGn")
showMetricMatrix(torino_mape, "MAPE - Torino", "RdYlGn_r")
# showMetricMatrix(torino_mpe, "MPE - Torino", "RdYlGn")
```

2.2.8 Change window size N

```
    #Learning strategy: sliding window
# The training set start from the first day of the test set -1 hour
# going backward in the training set till 3 weeks ago (-N)
def diffTrainLengthSliding(p, q, serie, n_samples, test_len):
    d = 0
    print("Testing ARIMA order (%i,%i,%i)" % (p, d, q))
    future = np.zeros((len(n_samples), test_len))
    for n in n_samples:
        print(n)
        start = 0
        while start < test_len:
            train = serie["count"][24 * 7 * 3 - n + start : 24 * 7 * 3 + start]
            arima_model = ARIMA(train, order=(p, d, q), enforce_stationarity=False)
            m_model = arima_model.fit(
                method="statespace",
            )
            forecast = m_model.forecast()
            future[n // 24 - 1, start] = forecast.values[0]
            start += 1
    return future

#Learning strategy: expanding window
# The training set start from the first day of the test set -1 hour
# Going backward in the training set till 3 weeks ago (-N)
def diffTrainLengthExpanding(p, q, serie, n_samples, test_len):
    d = 0
    print("Testing ARIMA order (%i,%i,%i)" % (p, d, q))
    future = np.zeros((len(n_samples), test_len))
    for n in n_samples:
        print(n)
        start = 0
        while start < test_len:
            train = serie["count"][24 * 7 * 3 - n : 24 * 7 * 3 + start]
            arima_model = ARIMA(train, order=(p, d, q), enforce_stationarity=False)
            m_model = arima_model.fit(
                method="statespace",
            )
            forecast = m_model.forecast()
            future[n // 24 - 1, start] = forecast.values[0]
            start += 1
    return future


best_p_m, best_q_m = np.unravel_index(madrid_mape.argmin(), madrid_mape.shape)
best_p_ny, best_q_ny = np.unravel_index(ny_mape.argmin(), ny_mape.shape)
best_p_t, best_q_t = np.unravel_index(torino_mape.argmin(), torino_mape.shape)

#Range for the N past samples to test - start: 24 hours, end: 3 weeks, step: 24 hours
```

```
n_samples = np.arange(24, 24 * 7 * 3 + 1, 24)
best_future_m_sliding = diffTrainLengthSliding(
    best_p_m, best_q_m, df_booking_madrid_full, n_samples, test_len
)
best_future_ny_sliding = diffTrainLengthSliding(
    best_p_ny, best_q_ny, df_booking_ny_full, n_samples, test_len
)
best_future_t_sliding = diffTrainLengthSliding(
    best_p_ny, best_q_ny, df_booking_torino_full, n_samples, test_len
)
best_future_m_expanding = diffTrainLengthExpanding(
    best_p_m, best_q_m, df_booking_madrid_full, n_samples, test_len
)
best_future_ny_expanding = diffTrainLengthExpanding(
    best_p_ny, best_q_ny, df_booking_ny_full, n_samples, test_len
)
best_future_t_expanding = diffTrainLengthExpanding(
    best_p_t, best_q_t, df_booking_torino_full, n_samples, test_len
)
```

2.2.9 Plots with different window size N

```
def plot_predictions_sliding(dates, true_values, predicted_values, title, n_samples):
    fig, ax = plt.subplots(layout="constrained", figsize=(14, 6))
    ax.plot(
        dates,
        true_values,
        color="black",
        label="True Values",
    )
    for n in n_samples:
        ax.plot(
            dates,
            predicted_values[n // 24 - 1],
            label=f"{n//24} days",
        )
    ax.xaxis.set_major_locator(md.DayLocator(interval=1))
    ax.xaxis.set_major_formatter(md.DateFormatter("%d-%m-%Y"))
    ax.xaxis.set_minor_locator(md.HourLocator())
    plt.ylim(bottom=-50, top=max(np.max(true_values), np.max(predicted_values)) + 100)
    plt.xticks(rotation=45, ha="right", rotation_mode="anchor", fontsize=10)
    plt.xlim(
        [
            dt.datetime.strptime("2017-10-22 00", "%Y-%m-%d %H"),
            dt.datetime.strptime("2017-10-29 00", "%Y-%m-%d %H"),
        ]
    )
    plt.title(f"Prediction - {title}")
    plt.grid("minor")
    plt.legend(ncols=10)
    plt.savefig(f"Prediction - Sliding - {title}.eps", format="eps")
    plt.show()


plot_predictions_sliding(
    dates,
    df_booking_madrid_full["count"][15 * 24 : 22 * 24],
    best_future_m_sliding,
    "Madrid",
```

```
    n_samples,
)
plot_predictions_sliding(
    dates,
    df_booking_ny_full["count"][15 * 24 : 22 * 24],
    best_future_ny_sliding,
    "NY",
    n_samples,
)
plot_predictions_sliding(
    dates,
    df_booking_torino_full["count"][15 * 24 : 22 * 24],
    best_future_t_sliding,
    "Torino",
    n_samples,
)

def plot_predictions_expanding(dates, true_values, predicted_values, title, n_samples):
    fig, ax = plt.subplots(layout="constrained", figsize=(14, 6))
    ax.plot(
        dates,
        true_values,
        color="black",
        label="True Values",
    )
    for n in n_samples:
        ax.plot(
            dates,
            predicted_values[n // 24 - 1],
            label=f"{n//24} days",
        )
    ax.xaxis.set_major_locator(md.DayLocator(interval=1))
    ax.xaxis.set_major_formatter(md.DateFormatter("%d-%m-%Y"))
    ax.xaxis.set_minor_locator(md.HourLocator())
    plt.ylim(
        bottom=-50,
        top=max(np.max(true_values), np.max(predicted_values)) + 100,
    )
    plt.xticks(rotation=45, ha="right", rotation_mode="anchor", fontsize=10)
    plt.xlim(
        [
            dt.datetime.strptime("2017-10-22 00", "%Y-%m-%d %H"),
            dt.datetime.strptime("2017-10-29 00", "%Y-%m-%d %H"),
        ]
    )
    plt.title(f"Prediction - {title}")
    plt.grid("minor")
    plt.legend(ncols=10)
    plt.savefig(f"Prediction - Expanding - {title}.eps", format="eps")
    plt.show()


plot_predictions_expanding(
    dates,
    df_booking_madrid_full["count"][15 * 24 : 22 * 24],
    best_future_m_expanding,
    "Madrid",
    n_samples,
)
plot_predictions_expanding(
    dates,
```

```
    df_booking_ny_full["count"][15 * 24 : 22 * 24],
    best_future_ny_expanding,
    "NY",
    n_samples,
)
plot_predictions_expanding(
    dates,
    df_booking_torino_full["count"][15 * 24 : 22 * 24],
    best_future_t_expanding,
    "Torino",
    n_samples,
)
```

2.3 Optional part

2.3.1 Part 8

```
def evaluate_arima(serie, p_range, q):
    d = 0
    future = np.zeros((len(p_range), 24 * 7))

    for p_idx, p in enumerate(p_range):
        train = serie["count"]
        arima_model = ARIMA(train[: -24 * 3], order=(p, d, q))
        model_fit = arima_model.fit(method="statespace")
        forecast = model_fit.forecast(steps=24 * 7)

        future[p_idx] = forecast

    return future


train_len = 24 * 7 * 2 # 2 weeks
test_len = 24 * 7 # 1 week
p_range = [6, 12, 24, 48]


future_m = evaluate_arima(
    df_booking_madrid_full,
    p_range,
    best_q_m,
)
future_ny = evaluate_arima(df_booking_ny_full, p_range, best_q_ny)
future_t = evaluate_arima(df_booking_torino_full, p_range, best_q_t)

def plotPredictions(dates, p_range, original, predicted, city):
    fig, ax = plt.subplots(layout="constrained", figsize=(14, 6))
    ax.plot(
        dates[0],
        original[24 * 7 * 3 + 24 * 5 : 24 * 7 * 3 + 24 * 8],
        color="black",
        label="True Values",
    )
    for p in range(len(p_range)):
        ax.plot(dates[1], predicted[p], label=f"p = {p_range[p]}")
    ax.xaxis.set_major_locator(md.DayLocator(interval=1))
    ax.xaxis.set_major_formatter(md.DateFormatter("%d-%m-%Y"))
    ax.xaxis.set_minor_locator(md.HourLocator())
    plt.title(f"Prediction for h = 7 days - {city}")
```

```
    plt.grid("minor")
    plt.legend()
    plt.savefig(f"Prediction_7_days_{city}.eps", format="eps")
    plt.show()


dates1 = pd.date_range(
    start="2017-10-29 00:00:00", end="2017-10-31 23:00:00", freq="1H"
)
dates2 = pd.date_range(
    start="2017-10-29 00:00:00", end="2017-11-04 23:00:00", freq="1H"
)
plotPredictions([dates1, dates2], p_range, df_booking_madrid_full, future_m, "Madrid")
plotPredictions([dates1, dates2], p_range, df_booking_ny_full, future_ny, "New York")
plotPredictions([dates1, dates2], p_range, df_booking_torino_full, future_t, "Torino")

def evaluate_arima(serie, p_range, q):
    d = 0
    future = np.zeros((len(p_range), 24))

    for p_idx, p in enumerate(p_range):
        train = serie["count"][:-12]
        arima_model = ARIMA(train, order=(p, d, q))
        model_fit = arima_model.fit(method="statespace")
        forecast = model_fit.forecast(steps=24)

        future[p_idx] = forecast

    return future


train_len = 24 * 7 * 2 # 2 weeks
test_len = 24 * 7 # 1 week
p_range = [3, 6, 12, 18]


future_m = evaluate_arima(
    df_booking_madrid_full,
    p_range,
    best_q_m,
)
future_ny = evaluate_arima(df_booking_ny_full, p_range, best_q_ny)
future_t = evaluate_arima(df_booking_torino_full, p_range, best_q_t)

def plotPredictions(dates, p_range, original, predicted, city):
    fig, ax = plt.subplots(layout="constrained", figsize=(14, 6))
    ax.plot(
        dates[0],
        original[-12:],
        color="black",
        label="True Values",
    )
    for p in range(len(p_range)):
        ax.plot(dates[1], predicted[p], label=f"p = {p_range[p]}")
    ax.xaxis.set_major_locator(md.DayLocator(interval=1))
    ax.xaxis.set_major_formatter(md.DateFormatter("%d-%m-%Y"))
    ax.xaxis.set_minor_locator(md.HourLocator())
    ax.xaxis.set_minor_formatter(md.DateFormatter("%H"))
    plt.title(f"Prediction for h = 24 hours - {city}")
    plt.grid("both")
    plt.legend()
```

```
    plt.savefig(f"Prediction_24_hours_{city}.eps", format="eps")
    plt.show()


dates1 = pd.date_range(
    start="2017-10-31 12:00:00", end="2017-10-31 23:00:00", freq="1H"
)
dates2 = pd.date_range(
    start="2017-10-31 12:00:00", end="2017-11-01 11:00:00", freq="1H"
)

plotPredictions([dates1, dates2], p_range, df_booking_madrid_full, future_m, "Madrid")
plotPredictions([dates1, dates2], p_range, df_booking_ny_full, future_ny, "New York")
plotPredictions([dates1, dates2], p_range, df_booking_torino_full, future_t, "Torino")

def evaluate_arima(serie, p_range, q):
    d = 0
    future = np.zeros((len(p_range), 48))

    for p_idx, p in enumerate(p_range):
        train = serie["count"][:-24]
        arima_model = ARIMA(train, order=(p, d, q))
        model_fit = arima_model.fit(method="statespace")
        forecast = model_fit.forecast(steps=48)

        future[p_idx] = forecast

    return future


train_len = 24 * 7 * 2 # 2 weeks
test_len = 24 * 7 # 1 week
p_range = [6, 12, 18, 24]


future_m = evaluate_arima(
    df_booking_madrid_full,
    p_range,
    best_q_m,
)
future_ny = evaluate_arima(df_booking_ny_full, p_range, best_q_ny)
future_t = evaluate_arima(df_booking_torino_full, p_range, best_q_t)

def plotPredictions(dates, p_range, original, predicted, city):
    fig, ax = plt.subplots(layout="constrained", figsize=(14, 6))
    ax.plot(
        dates[0],
        original[-24:],
        color="black",
        label="True Values",
    )
    for p in range(len(p_range)):
        ax.plot(dates[1], predicted[p], label=f"p = {p_range[p]}")
    ax.xaxis.set_major_locator(md.DayLocator(interval=1))
    ax.xaxis.set_major_formatter(md.DateFormatter("%d-%m-%Y"))
    ax.xaxis.set_minor_locator(md.HourLocator())
    ax.xaxis.set_minor_formatter(md.DateFormatter("%H"))
    plt.xticks(rotation=45, ha="right", rotation_mode="anchor", fontsize=10)
    plt.title(f"Prediction for h = 48 hours - {city}")
    plt.grid("both")
    plt.legend()
```

```
    plt.savefig(f"Prediction_48_hours_{city}.eps", format="eps")
    plt.show()


dates1 = pd.date_range(
    start="2017-10-31 00:00:00", end="2017-10-31 23:00:00", freq="1H"
)
dates2 = pd.date_range(
    start="2017-10-31 00:00:00", end="2017-11-01 23:00:00", freq="1H"
)

plotPredictions([dates1, dates2], p_range, df_booking_madrid_full, future_m, "Madrid")
plotPredictions([dates1, dates2], p_range, df_booking_ny_full, future_ny, "New York")
plotPredictions([dates1, dates2], p_range, df_booking_torino_full, future_t, "Torino")

client.close()
```