Politecnico di Torino
ICT for Smart Mobility
Laboratory report # 2
Group 11

Berti Marco, s315819
Ravera Stefano, s318946
Scardi Alessia, s317628

A.Y. 2023/24

Contents

1 Prediction using ARIMA models

ARIMA (AutoRegressive Integrated Moving Average) is a time series forecasting method that combines autoregression, differencing, and moving average components. It is used for predicting future points in a time series dataset.

Laboratory 2 is centered on the implementation of the ARIMA models, especially focusing on the hyperparameters tuning, to fit the preprocessed data from Laboratory 1 and perform predictions.

1.1 Time-series extraction and filtering

In order to use ARIMA models, the dataset should be made of time series. The data returned from LAB 1 are filtered considering a 30 days window, going from the 6-*th* of September to the 6-*th* of October 2017, chosen because we expect stationarity in each of the examined cities (Torino, Seattle, Berlin) for the cited time window. The stationarity is highly probable since September/October are not festivity months: it is reasonable to expect a high regularity in car rentals.

1.2 Preprocessing

The available dataset is divided into hours, with the total of 720 points (30 days x 24 hours) in order to count the number of rentals recorded at each hour, and paying attention that the car was really moving. The resulting dataset is made of couples: datetime hour by hour and the count of rentals.

In order to work with float values, a random component is introduced in the number of rentals. how(?)

ARIMA models assume consistent time intervals between observations, not allowing missing samples: missing values are then defined as those whose count is not present for a specific hour. To address missing instances in the dataset, at first zeros are used, then an average methodology is employed. This involves the replacement of zero values with an average based on the available data from corresponding hours of the same day of the week within the time window of 30 days. The following step of the preprocessing part regards removing outliers from the dataset: a point is considered an outlier based on its distance from the just cited average, and replaced with the average if it is more than 100% or less than 30% of the average (the kept rentals should have a duration greater than 5 minutes and lower than 2 hours). do you also check if the car moved (?)

why first insert 0 (?)

(?)

why (?)

The graph of the filtered system utilization over time is shown in fig. 1, showing the number of rentals per hour for the three cities.

1.3 Time-series stationary properties

Forecasting might be challenging due to the non-deterministic nature of time series data, meaning we cannot predict future occurrences with certainty. However, the task becomes more manageable if the time series is stationary. In such cases, predicting future outcomes

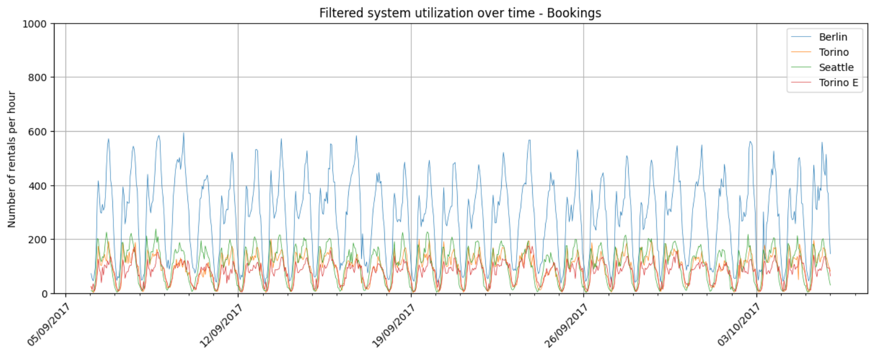


Figure 1: Filtered System Utilization over time for the Bookings.

involves assuming that the statistical properties of the time series will remain consistent over time.

Rolling statistics can be used to evaluate the stationary properties of the time series, which allows the setting of the parameter d - the number of differentiation steps to make the model stationary - of the ARIMA model: if the time series result to be stationary then d can be set to zero since no differentiation is needed.

By plotting the rolling average and the rolling standard deviation on a 7-days time window it can be shown that the time series of the three cities at ease are all stationary: fig. 2 shows the rolling statistics for the time series of Berlin (the other graphs are analogous and present in fig. 9, fig. 10, and fig. 11 in the Appendix section at 2.1). The resulting lines representing the rolling average and standard deviation are horizontal with respect to the time series, meaning that they do not change over time.

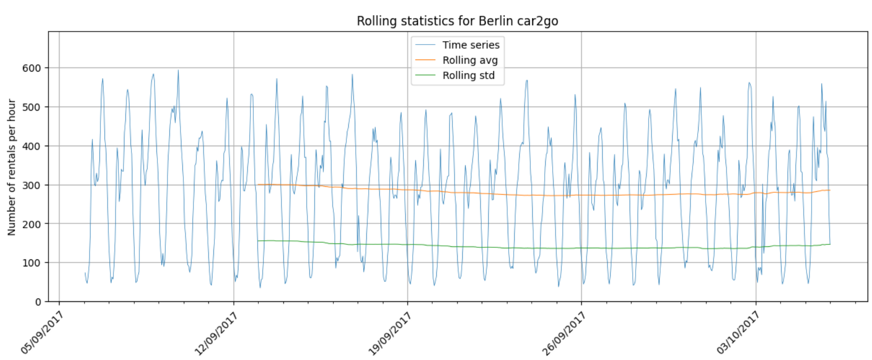


Figure 2: Rolling statistics with a 7-days window for Berlin's data.

1.4 ACF & PACF

Autocorrelation plots are also very useful to determine if a time series is stationary or not beacuse it is simply possible to look at the decay of the time series, that, in the case of non-stationarity, is very slow. However the Autocorrelation Function (ACF) and the Partial Autocorrelation Function (PACF) of the time series can be computed to evaluate the remaining parametrs of the ARIMA model: p - the lag in the AR model - and q - the lag in the MA model. The following observations are made:

- PACF stops after lag p for Autoregressive (AR) models   for pure AR
- ACF stops after lag q for Moving average (MA) models   for pure MA
- ARMA models are the result of the 2 components of the AR and MA models, whose order is denoted by the parameters p and q. The AR component of the model is designed to capture the interdependence between the current value of the time series and its preceding values. Consequently, the hyper-parameter p is intricately linked to the count of hours that exhibit the cited correlation. The MA component of the model is designed to encapsulate the dependence of the current time series value on the noise.

?

The plots for the ACF and the PACF are shown in fig. 3 for Berlin and in the Appendix in fig. 12, fig. 13 and fig. 14 for the other cities. The initial guess for both the parameters p and q is 2 for all of them since the plots are very similar.

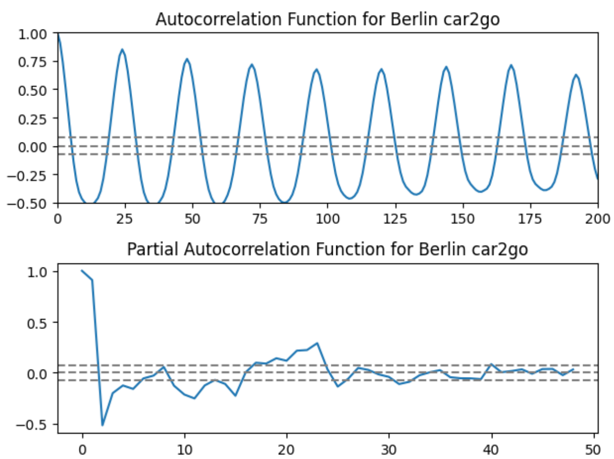


Figure 3: ACF and PACF for Berlin.

1.5 Training the model

The ARIMA model is trained and then used to predict new data. The number of past samples N used for the training set is equal to 360 (15 days), same as the number of samples used to test the model. The initial model employed a sliding window learning approach, involving the segmentation of a large dataset into smaller, overlapping subsets. Each of these subsets is used for training, allowing the model to iteratively update its knowledge as it traversed the dataset. In the testing phase, a distinct model was created for each test sample, enhancing adaptability to evolving data dynamics.

?

1.6 Metrics

Given N, (p,d,q) and a trained model, the error can be computed and evaluated through different metrics such as MPE (Mean Percentage Error) and MAPE (Mean Absolute Percentage Error) for each city.

MAPE is a widely used metric in forecasting and predictive modeling (including ARIMA) to evaluate the accuracy of a model's predictions. It provides a measure of how well a forecasting model is able to predict values in comparison to the actual observed values. MAPE measures the relative percentage difference between predicted and actual values. It is calculated as the average of the absolute percentage differences, as shown in the first expression of eq. 1

MPE is another metric used for evaluating the accuracy of forecasting models and it is a simpler variant of the MAPE, but it does not involve taking the absolute values of the percentage errors. It is evaluated as the second expression shown in eq. 1.

$$\text{MAPE} = \frac{100}{N}\sum_{t=1}^{N}\left|\frac{X_t - \hat{X}_t}{X_t}\right| \qquad \text{MPE} = \frac{100}{N}\sum_{t=1}^{N}\left(\frac{X_t - \hat{X}_t}{X_t}\right) \tag{1}$$

where N is the number of observations, X_t represent the actual values and $\hat{X}_t$ represent the predicted values.

?

?

MPE is affected by the scale of the data. This is because MPE is based on the raw percentage errors without taking their absolute values. The sign and magnitude of the percentage errors contribute directly to the MPE calculation. As a result, datasets with different scales can lead to different MPE values, and comparing MPE across datasets may not be meaningful.

In contrast, MAPE addresses this issue by taking the absolute values of the percentage errors before averaging them. This ensures that the metric is scale-independent, making it more suitable for comparing forecasting performance across datasets with different scales.

In the case of ARIMA models, MPE provides insights into the average direction - if MPE is positive, it indicates that, on average, the ARIMA model tends to overestimate the actual values. This means that the model predictions are, on average, higher than the observed values -and magnitude - a larger absolute MPE value reflects a higher average percentage error since the magnitude of MPE indicates the average percentage by which the model's predictions deviate from the actual values - of errors, while the lower the MAPE, the better the ARIMA models' performance.

The results for the three cities for MPE and MAPE are shown in table 1. It is clearly shown that Berlin has the lowest MAPE parameter among the three cities, meaning that ARIMA model performs better on Berlin's time series. Also MPEs are negative for all the cities, meaning that the ARIMA model tends to underestimate the actual values, and the predictions are, on average, lower than the observed values. For the city of Berlin the MPE is also lower in magnitude with respect to the other cities, meaning that the it is the city for which the the predictions deviate the least from the actual values.

1.7 Impact of the parameters

1.7.1 Fixed N, tuning p and q

The first strategy used to evaluate the impact of the parameters on the model is to fix the parameter N and perform a grid search on the parameters p and q. In grid search, a predefined grid of hyperparameter values is specified, and the model is trained and evaluated for each combination of these values. The hyperparameter tuning is performed in order to systematically find the best configuration of parameters that optimize the ARIMA model. The training approach used is the one described in section 1.5.

- N is fixed to 360 (15 days);
- Split training and test 50%/50% (360 hours of train data/360 hours of test data);
- Grid search with ranges $p = q = [0, 1, 2, 3, 4, 5]$ and $d = 0$;
- MPE is taken into account but MAPE is used as metric for evaluating the optimal parameters, since it is a scale-independent metric.

The heatmaps representing the MAPE for all the combination of parameters for Berlin and Torino are shown in fig. 4 while the heatmaps for Seattle and Torino_Enjoy are shown in the Appendix in fig. 15. The best combination of parameters for Berlin is $d = 0$ and $p = q = 2$ while for Torino the best choice is $d = 0$, $p = 2$ and $q = 5$ (same for Seattle and Torino_Enjoy).

1.7.2 Best parameter configuration, tuning N and learning strategy

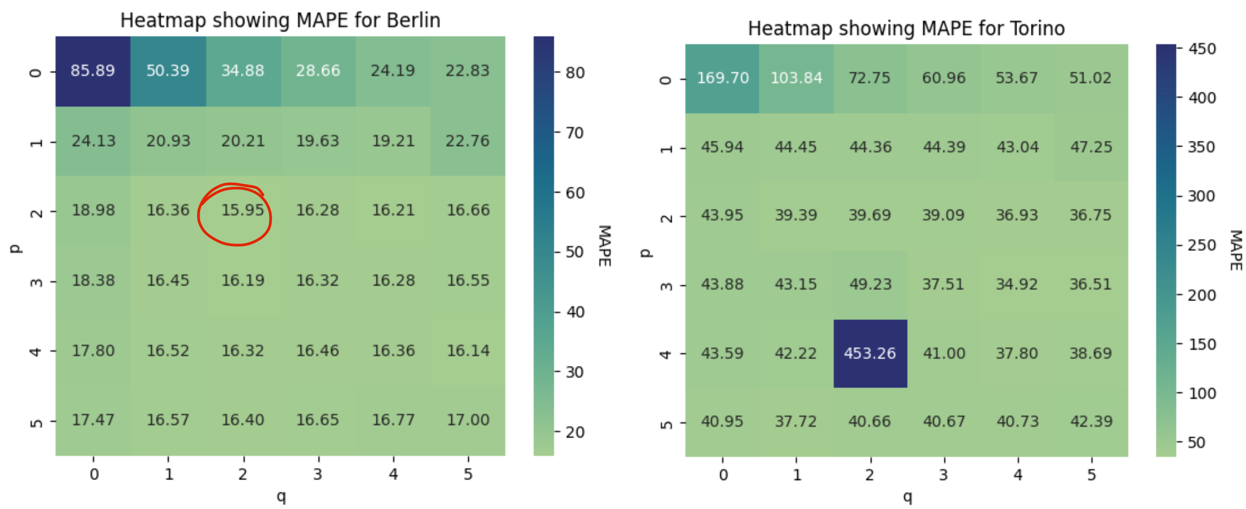
The best values for the parameters p and q are listed in the previous section but they have been slightly changed for the sake of simplicity: since the parameters p and q determine the complexity of the ARIMA model, lower values of the parameters can be translated into a simpler model.

The best value of q is changed to $q = 3$ for Torino and $q = 2$ for Torino_Enjoy, since the difference in the MAPE when using the best parameters is about a few units, the choice of a simpler, more robust model is favoured to the choice of a more flexible but complex model.

Always keeping the last 10 days (240 test samples) of observations for predictions and the parameters p, q and d constant (Berlin: ARIMA(2,0,2), Torino: ARIMA(2,0,3), Seattle: ARIMA(2,0,5) and Torino_Enjoy: ARIMA(2,0,2)) it is possible to observe the outcome of the model when using different values of N and different learning strategies (sliding and expanding window).

	Berlin	Torino	Seattle	Torino_Enjoy
MAPE	15.947	39.688	34.180	32.272
MPE	-4.823	-24.093	-20.870	-15.254

Table 1: Evaluation metrics for the three cities with parameters d=0, p=2, q=2.



(a) Berlin. (b) Torino.

Figure 4: Heatmap showing MAPE.

The sliding window learning strategy has already been mentioned in section 1.5 while the expanding window learning strategy entails training a model on progressively larger data subsets.

Table 2 is referred to the city of Berlin and it shows the metrics MAPE and MPE for different values of N and for both the sliding and the expanding window learning strategy. The tables (ta. 19a, 19b and ta. 3) for the other cities are listed in the Appendix section.

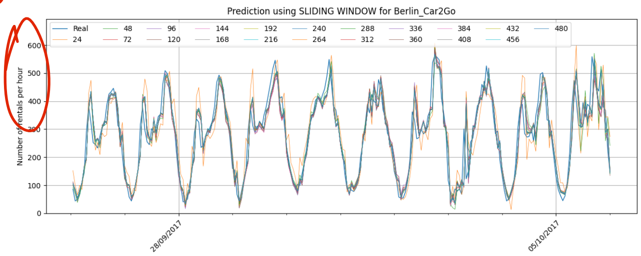
Figures 5a and 5b show respectively the graphical behaviour of the predicted time series when changing N through the sliding and the expanding window approach for Berlin (for other cities: fig. 16, fig. 17 and fig. 18 in Appendix).

	Sliding window		Expanding window	
N	MAPE	MPE	MAPE	MPE
24	27.524	-11.210	18.760	-3.593
48	18.583	-4.962	17.540	-1.135
72	17.820	-3.711	17.376	-3.253
96	18.226	-3.685	17.786	-4.345
120	18.416	-4.226	17.903	-4.342
144	18.319	-4.892	17.573	-4.431
168	18.107	-5.049	17.520	-4.079
192	17.785	-4.919	17.428	-3.647
216	17.452	-4.458	17.123	-3.100
240	17.359	-4.045	16.846	-3.757
264	17.542	-4.019	17.024	-4.570
288	17.634	-4.207	17.118	-4.936
312	17.514	-4.458	17.286	-5.282
336	17.550	-4.636	17.376	-5.446
360	17.456	-4.841	17.398	-5.398
384	17.284	-4.910	17.184	-5.038
408	17.069	-4.924	17.325	-5.436
432	17.028	-4.969	17.602	-5.884
456	17.177	-5.137	17.715	-6.142
480	17.348	-5.413	17.838	-6.345

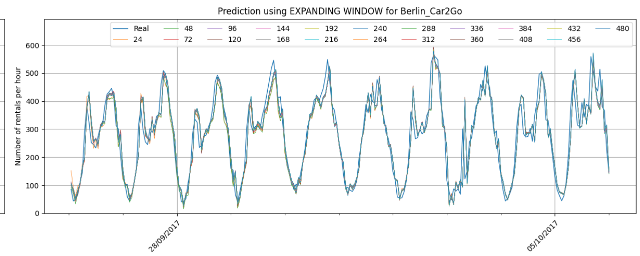
better plot these results

Table 2: Parameter N tuning for Berlin with $d = 0$ and $p = q = 2$.

too small



(a) Predictions of the number of rentals per hour while changing N, using the sliding window approach, for Berlin.



(b) Predictions of the number of rentals per hour while changing N, using the expanding window approach, for Berlin.

Figure 5: Berlin.

1.7.3 Results comparison

The influence of N on prediction accuracy is consistent across all cities. Larger values of N generally lead to improved forecasting performance, as demonstrated by the decrease in MAPE across different cities. This common trend suggests that capturing a longer historical context enhances model accuracy universally.

When comparing the different learning approaches for the three cities it occurs that the sliding window consistently outperforms the expanding window in Berlin, but this pattern is not universally observed across all cities. Seattle, for instance, exhibits closer performance between the two strategies, emphasizing the importance of considering city-specific characteristics in model selection.

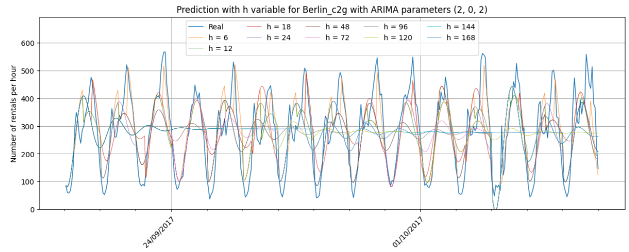
The trade-off between model complexity and accuracy is evident in all cities. Choosing simpler models ($p = q = 2$) generally results in a marginal increase in MAPE, emphasizing the practical importance of simpler and more robust models.

Considering the relative error (MAPE) in the context of absolute numbers of rentals is crucial for a comprehensive evaluation: in cities with smaller absolute rental numbers, higher relative errors may be observed.

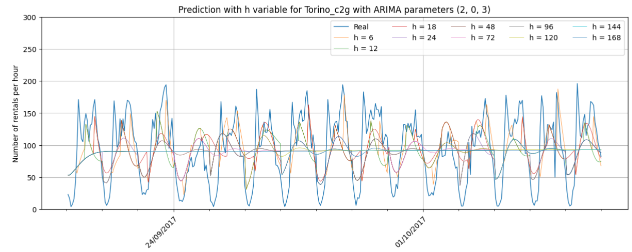
1.8 (Optional) Impact of the time horizon h of the prediction on the performance

The time horizon h of an ARIMA model refers to the number of future time points to be forecast. In ARMA models, when the differencing component is equal to zero, the time horizon is the number of future time points to be predicted, it determines how far into the future the model will project.

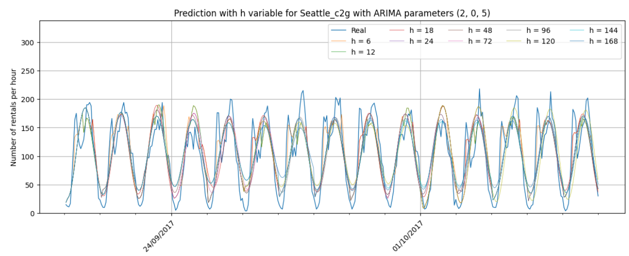
When the number of points to be predicted at each iteration is increased ($h = 6$(hours), 12, 18, 24, 48, 72, 96, 120, 144, 168 (7 days) points). While increasing h it can be noticed that the accuracy of the prediction decreases until becoming a horizontal time for Berlin and Torino. Fig. 6a shows the trend of the number of rentals per hour in Berlin through time, when changing the parameter h and keeping the ARIMA parameter described in sec. 1.7.2, and fig. 6b and 7b show a similar behaviour for the city of Torino. The just described trend is not present in the time series regarding the city of Seattle, as shown in fig. 7a.



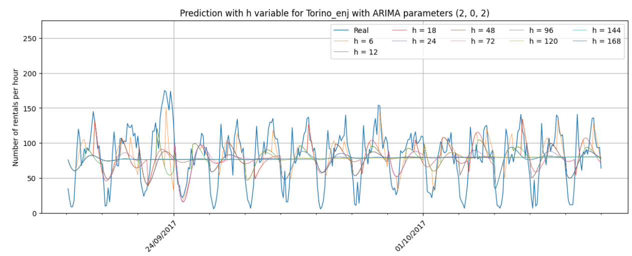
(a) Number of rentals per hour through time for Berlin with h variable.



(b) Number of rentals per hour through time for Torino with h variable.



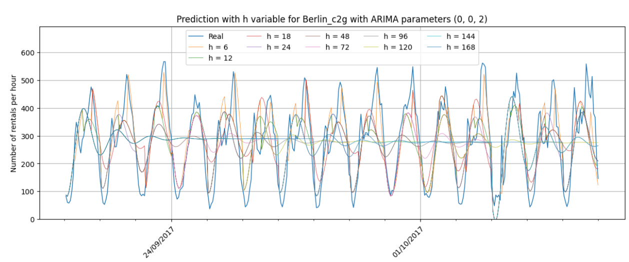
(a) Number of rentals per hour through time for Seattle with h variable.



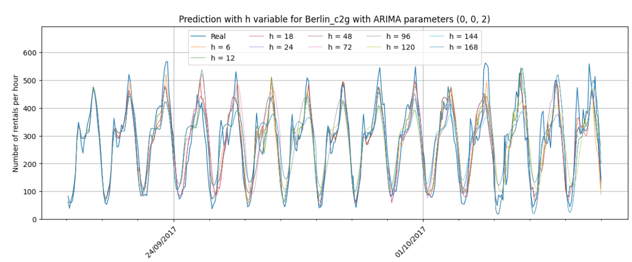
(b) Number of rentals per hour through time for Torino (Enjoy) with h variable.

The parameter h is then changed again as described above, but also the parameter p, the number of past data used for predictions, in changed for the analysis: p=4, 8, 12, 24, 48. The trade-off between the two quantities is what stands out from the graphs: when using a low value for the parameter p the short term accuracy is high, while the long term one is poor, also because of the presence of high values of h, resulting in a worsening of the performance when increasing.

When the chosen value for p is increased, the behaviour of the time series at the changing of the parameter h is similar to their real behaviour. The graphs for Berlin with the parameter p=4 and p=24 are respectively shown in fig. 8a and fig. 8b.



(a) Predictions with variable h and p=4 for Berlin.



(b) Predictions with variable h and p=48 for Berlin

- well done and complete
- missing baseline comparison

5⁻/5

2 Appendix

2.1 Figures

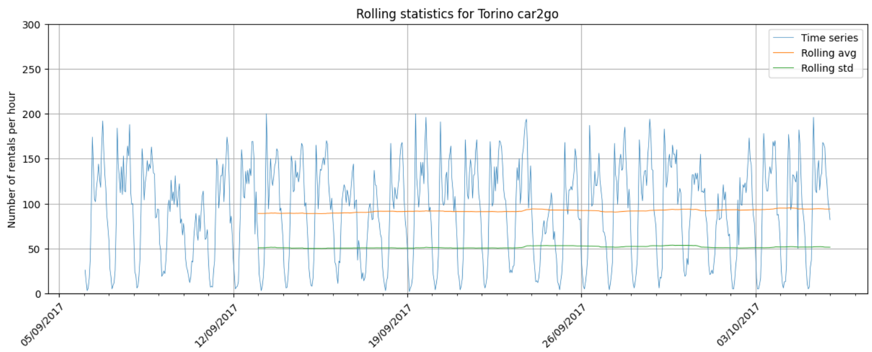


Figure 9: Rolling statistics with a 7-days window for Torino's data.

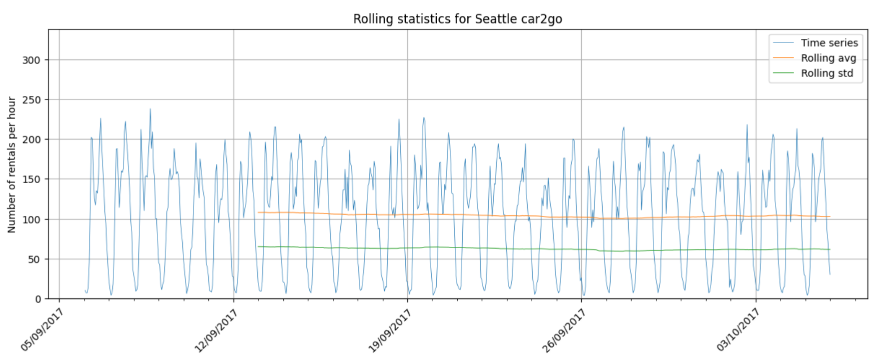


Figure 10: Rolling statistics with a 7-days window for Seattle's data.

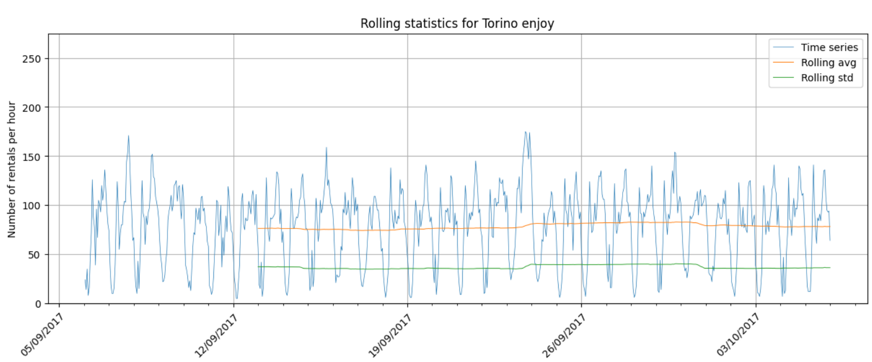


Figure 11: Rolling statistics with a 7-days window for Torino (Enjoy)'s data.

2.2 Tables

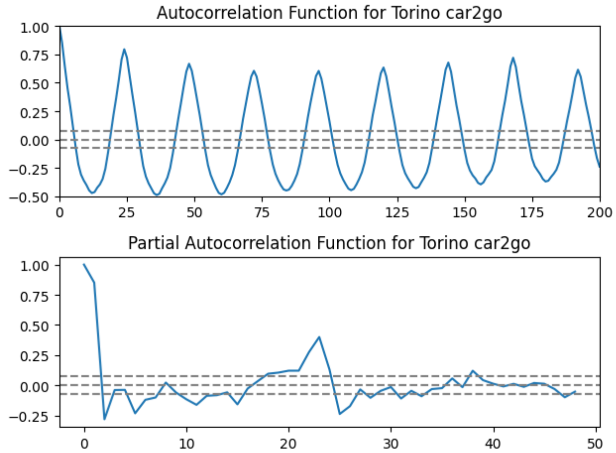


Figure 12: ACF and PACF for Berlin.

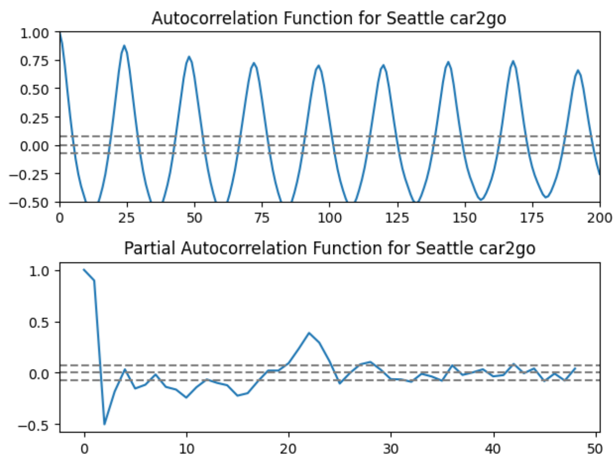


Figure 13: ACF and PACF for Berlin.

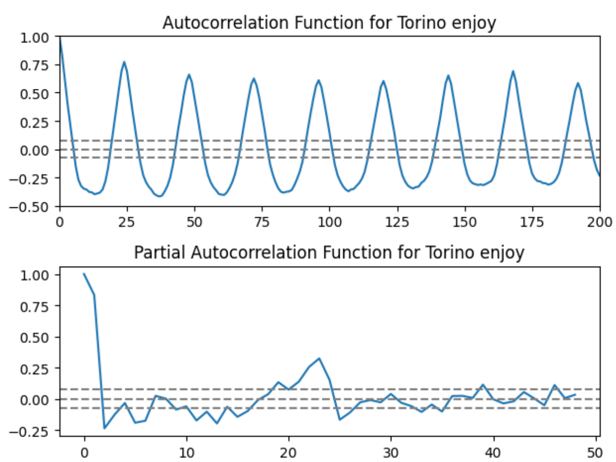
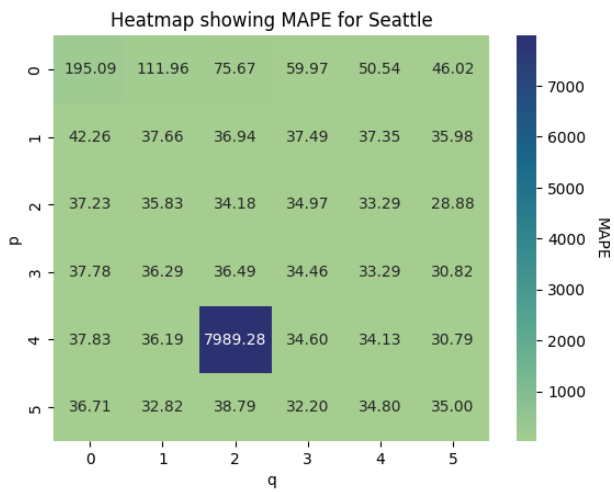
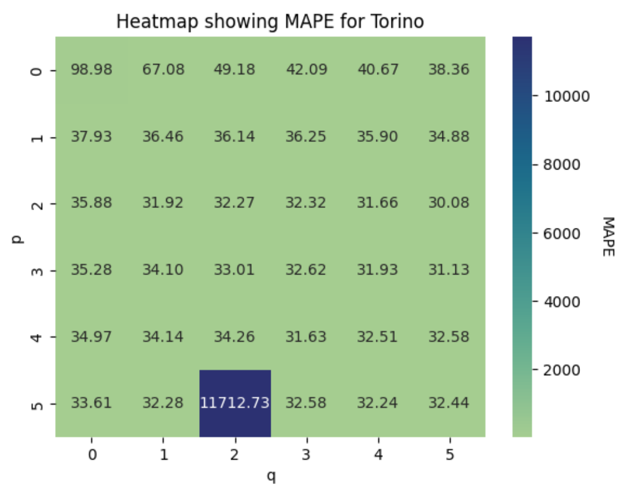


Figure 14: ACF and PACF for Berlin.

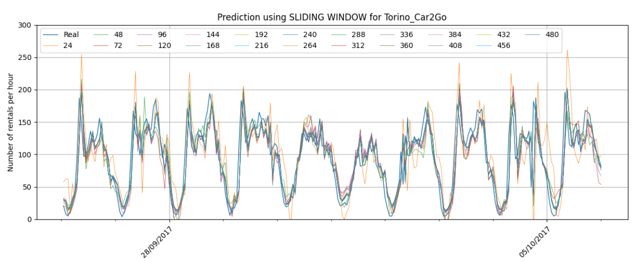


(a) Seattle.

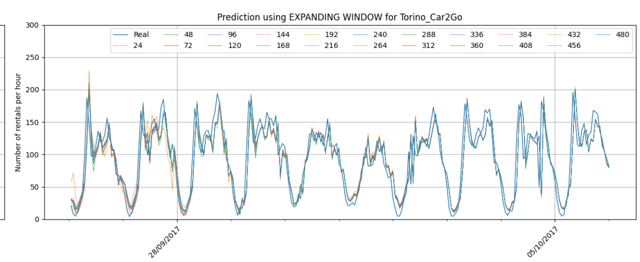


(b) Torino Enjoy.

Figure 15: Heatmap showing MAPE.

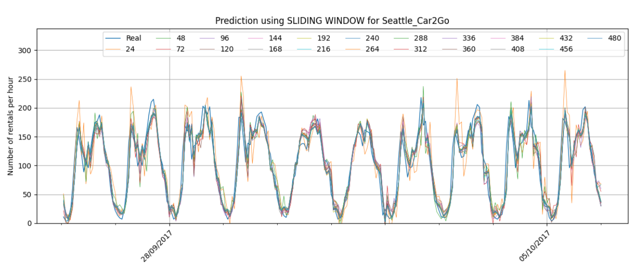


(a) Predictions of the number of rentals per hour while changing N, using the sliding window approach, for Torino.

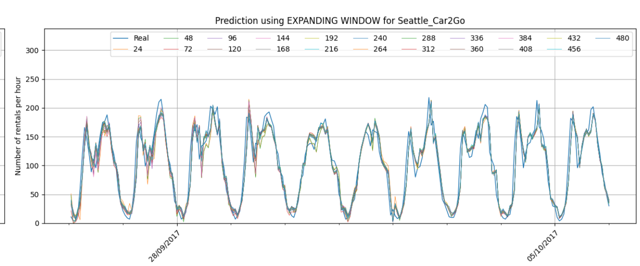


(b) Predictions of the number of rentals per hour while changing N, using the expanding window approach, for Torino.

Figure 16: Torino.

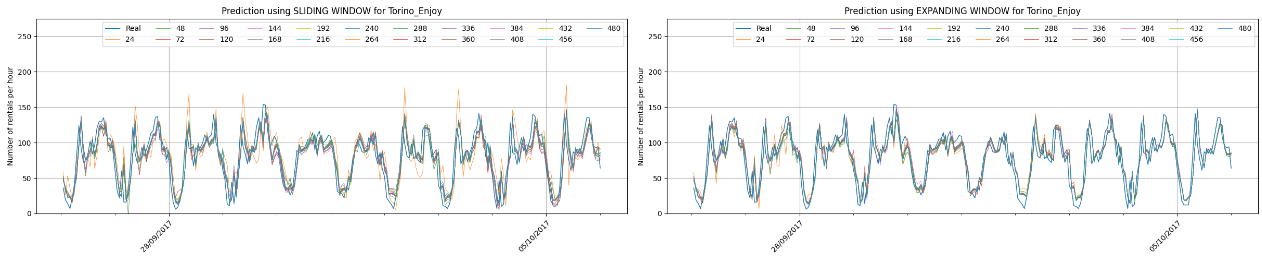


(a) Predictions of the number of rentals per hour while changing N, using the sliding window approach, for Seattle.



(b) Predictions of the number of rentals per hour while changing N, using the expanding window approach, for Seattle.

Figure 17: Seattle.



(a) Predictions of the number of rentals per hour while changing N, using the sliding window approach, for Torino_Enjoy.

(b) Predictions of the number of rentals per hour while changing N, using the expanding window approach, for Torino_Enjoy.

Figure 18: Torino.

2.3 Code

```
# %%
import pymongo as pm # import MongoClient only
import pprint # prettyprinting for json objects
import datetime
import pytz
import matplotlib as mpl
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.ticker as ticker
import matplotlib.dates as md
import pandas as pd
from statsmodels.tsa.stattools import acf, pacf
from statsmodels.tsa.arima.model import ARIMA
import numpy as np
import warnings

# %%
client = pm.MongoClient('bigdatadb.polito.it', ssl=True, authSource =
↪ 'carsharing', username = 'ictts', password ='Ict4SM22!',
↪ tlsAllowInvalidCertificates=True)
db = client["carsharing"] # Choose the DB to use

c2g_PermanentBookings = db["PermanentBookings"]
enj_PermanentBookings = db["enjoy_PermanentBookings"]

# %%
our_cities = ["Berlin", "Torino", "Seattle"]

c2g_dates_PB_Berlin = []
c2g_count_PB_Berlin = []
```

	Sliding window		Expanding window	
N	MAPE	MPE	MAPE	MPE
24	72.190	-43.668	44.068	-25.895
48	42.281	-19.140	36.114	-17.953
72	41.745	-14.853	33.702	-14.023
96	36.414	-16.570	35.843	-18.381
120	37.661	-19.521	37.217	-20.559
144	39.600	-22.504	38.809	-22.445
168	39.721	-23.445	39.153	-23.073
192	39.026	-22.879	39.610	-23.581
216	38.292	-22.149	38.626	-22.225
240	37.562	-21.263	36.668	-19.919
264	37.219	-20.788	36.794	-20.085
288	37.420	-21.002	36.924	-20.416
312	38.253	-21.815	37.619	-21.237
336	38.698	-22.447	38.239	-22.062
360	38.708	-22.464	38.265	-22.108
384	38.165	-21.975	37.579	-21.092
408	37.486	-21.135	36.219	-19.433
432	37.153	-20.728	36.220	-19.408
456	37.498	-20.990	37.008	-20.412
480	38.845	-22.478	38.947	-22.358

(a) Parameter N tuning for Torino with $d = 0$, $p = 2$ and $q = 3$.

	Sliding window		Expanding window	
N	MAPE	MPE	MAPE	MPE
24	47.217	-22.375	34.199	-17.035
48	41.870	-20.662	29.166	-9.378
72	36.628	-16.558	25.154	-5.328
96	33.132	-16.218	28.003	-12.346
120	31.751	-14.786	27.802	-12.940
144	31.982	-17.526	29.638	-15.007
168	32.722	-17.821	30.768	-17.411
192	30.801	-15.264	30.489	-16.899
216	29.182	-14.988	28.881	-14.954
240	28.840	-14.846	28.019	-14.124
264	28.228	-14.213	29.800	-16.616
288	28.442	-14.900	29.918	-17.206
312	28.266	-14.950	30.054	-17.141
336	28.555	-15.125	30.692	-17.638
360	29.992	-16.871	30.715	-17.633
384	30.167	-16.932	29.610	-16.267
408	29.097	-16.067	29.601	-16.393
432	29.407	-16.540	29.704	-16.939
456	28.965	-16.059	30.514	-17.886
480	29.449	-16.482	30.817	-17.967

(b) Parameter N tuning for Seattle with $d = 0$, $p = 2$ and $q = 5$.

```
c2g_dates_PB_Torino = []
c2g_count_PB_Torino = []
c2g_dates_PB_Seattle = []
c2g_count_PB_Seattle = []

for city in our_cities:

    # PermanentBookings
    if city == "Berlin" or city == "Torino":
        start_date = datetime.datetime(2017, 9, 6, 0, 0, 0, tzinfo =
         ↪ datetime.timezone.utc).timestamp() - 3600
        end_date = datetime.datetime(2017, 10, 6, 0, 0, 0, tzinfo =
         ↪ datetime.timezone.utc).timestamp() - 3600
    else: #city == "Seattle"
        start_date = datetime.datetime(2017, 9, 6, 0, 0, 0, tzinfo =
         ↪ datetime.timezone.utc).timestamp() + 28800
        end_date = datetime.datetime(2017, 10, 6, 0, 0, 0, tzinfo =
         ↪ datetime.timezone.utc).timestamp() + 28800

    c2g_new_cursor_PB = c2g_PermanentBookings.aggregate([
            { "$match": { "city": city,
                        "init_time": {"$gte": start_date,
```

```
                                        "$lt": end_date}}},
            { "$project": { "_id": 0,
                            "init_address": 1,
                            "final_address": 1,
                            "duration": { "$divide": [ {"$subtract": ["$final_time", "$init_time"]}, 60]},
                            "unique_timestamp": { "$dateFromString": { "dateString": { "$dateToString": { "date": "$init_date", "format": "%d-%m-%Y-%H"}}}},
                            "equal": {"$eq": ["$init_address", "$final_address"]}}},
            { "$match": { "equal": False,
                        "duration": {"$gte": 5, "$lt": 120}}},
            {"$group": {"_id": "$unique_timestamp",
                        "count": {"$sum": 1}}},
            {"$project": {"_id": 1,
                        "count": 1}},
            {"$sort": {"_id": 1}}])

    for elem in c2g_new_cursor_PB:
        if city == "Berlin":
            c2g_dates_PB_Berlin.append(elem["_id"])
            c2g_count_PB_Berlin.append(elem["count"])
        elif city == "Torino":
            c2g_dates_PB_Torino.append(elem["_id"])
            c2g_count_PB_Torino.append(elem["count"])
        else:
            c2g_dates_PB_Seattle.append(elem["_id"])
            c2g_count_PB_Seattle.append(elem["count"])

# Introducing a random component to make the counts float
for i in c2g_count_PB_Berlin:
    i = i + np.random.normal(0,1)
for i in c2g_count_PB_Torino:
    i = i + np.random.normal(0,1)
for i in c2g_count_PB_Seattle:
    i = i + np.random.normal(0,1)

# Creating the new lists of dates containing the missing values/dates
c2g_dates_PB_Berlin_UPDATED = []
c2g_dates_PB_Torino_UPDATED = []
c2g_dates_PB_Seattle_UPDATED = []

start = datetime.datetime(2017, 9, 6, 1, 0, 0)
end = datetime.datetime(2017, 10, 6, 0, 0, 0)
```

```python
delta = datetime.timedelta(hours=1)

while start <= end:
    c2g_dates_PB_Berlin_UPDATED.append(start)
    c2g_dates_PB_Torino_UPDATED.append(start)
    c2g_dates_PB_Seattle_UPDATED.append(start)
    start += delta

# Creating the new lists of counts inserting 0 in the missing values
for i in range(len(c2g_dates_PB_Berlin_UPDATED)-1): #It could be also c2g_dates_PB_Torino_UPDATED or c2g_dates_PB_Seattle_UPDATED

    if c2g_dates_PB_Berlin_UPDATED[i] not in c2g_dates_PB_Berlin:
        c2g_count_PB_Berlin.insert(i, 0)
    if c2g_dates_PB_Torino_UPDATED[i] not in c2g_dates_PB_Torino:
        c2g_count_PB_Torino.insert(i, 0)
    if c2g_dates_PB_Seattle_UPDATED[i] not in c2g_dates_PB_Seattle:
        c2g_count_PB_Seattle.insert(i, 0)

# %%
# DOING THE SAME THING FOR TORINO ENJOY

# PermanentBookings
enj_dates_PB_Torino = []
enj_count_PB_Torino = []

enj_new_cursor_PB = enj_PermanentBookings.aggregate([
        { "$match": { "city": "Torino",
                      "init_time": {"$gte": datetime.datetime(2017, 9, 6, 0, 0, 0).replace(tzinfo = datetime.timezone.utc).timestamp() - 3600,
                                    "$lt": datetime.datetime(2017, 10, 6, 0, 0, 0).replace(tzinfo = datetime.timezone.utc).timestamp() - 3600}}},
        { "$project": { "_id": 0,
                        "init_address": 1,
                        "final_address": 1,
                        "duration": { "$divide": [ {"$subtract": ["$final_time", "$init_time"]}, 60]},
                        "unique_timestamp": { "$dateFromString": { "dateString": { "$dateToString": { "date": "$init_date", "format": "%d-%m-%Y-%H"}}}},
                        "equal": {"$eq": ["$init_address", "$final_address"]}}},
```

```
        { "$match": { "equal": False,
                      "duration": {"$gte": 5, "$lt": 120}}},
        {"$group": {"_id": "$unique_timestamp",
                    "count": {"$sum": 1}}},
        {"$project": {"_id": 1,
                      "count": 1}},
        {"$sort": {"_id": 1}}])

for elem in enj_new_cursor_PB:
    enj_dates_PB_Torino.append(elem["_id"])
    enj_count_PB_Torino.append(elem["count"])

# Introducing a random component to make the counts float
for i in enj_count_PB_Torino:
    i = i + np.random.normal(0,1)

# Creating the new lists of dates containing the missing values/dates
enj_dates_PB_Torino_UPDATED = []

start_enj = datetime.datetime(2017, 9, 6, 1, 0, 0)
end_enj = datetime.datetime(2017, 10, 6, 0, 0, 0)
delta_enj = datetime.timedelta(hours=1)

while start_enj <= end_enj:
    enj_dates_PB_Torino_UPDATED.append(start_enj)
    start_enj += delta_enj

# Creating the new lists of counts inserting 0 in the missing values
for i in range(len(enj_dates_PB_Torino_UPDATED)-1): #It could be also c2g_dates_PB_Torino_UPDATED or c2g_dates_PB_Seattle_UPDATED
    if enj_dates_PB_Torino_UPDATED[i] not in enj_dates_PB_Torino:
        enj_count_PB_Torino.insert(i, 0)

# %%
def substituteByItsMean(list_dates, list_counts):
    for i in range(len(list_dates)-1): #Loop over dates length
        if list_counts[i] == 0: # If this was a missing value
            day = list_dates[i].weekday()
            hour = list_dates[i].hour
            sum = 0
            count = 0
            #print("\n" + str(i), " ", str(day), " ", str(hour))
            for j in range(len(list_dates)-1):
                if list_dates[j].weekday() == day and list_dates[j].hour == hour and list_dates[j] != list_dates[i]:
```

```
                    sum += list_counts[j]
                    count += 1
                    #print(str(list_dates[j].weekday()) + " " + str(list_dates[j].hour) + " " + str(list_counts[j]))
            avg = sum/count
            rounded_avg = round(avg)
            #print(str(avg) + " " + str(rounded_avg))
            list_counts[i] = rounded_avg
            #print(list_counts[i])

def deleteOutliers(list_dates, list_counts):
    for i in range(len(list_dates)-1): #Loop over dates length
        day = list_dates[i].weekday()
        hour = list_dates[i].hour
        sum = 0
        count = 0
        #print("\n" + str(i), " ", str(day), " ", str(hour))
        for j in range(len(list_dates)-1):
            if list_dates[j].weekday() == day and list_dates[j].hour == hour and list_dates[j] != list_dates[i]:
                sum += list_counts[j]
                count += 1
                #print(str(list_dates[j].weekday()) + " " + str(list_dates[j].hour) + " " + str(list_counts[j]))
        avg = sum/count
        rounded_avg = round(avg)
        #print(str(avg) + " " + str(rounded_avg))
        if list_counts[i] > 2*rounded_avg or list_counts[i] < 0.3*rounded_avg:
            list_counts[i] = rounded_avg
        #print(list_counts[i])

# %%
# METHOD TO REPLACE ZERO VALUES WITH THE MEAN OF THE SAME DAYS OF THE WEEK AT THE SAME HOURS
substituteByItsMean(c2g_dates_PB_Berlin_UPDATED, c2g_count_PB_Berlin)
substituteByItsMean(c2g_dates_PB_Torino_UPDATED, c2g_count_PB_Torino)
substituteByItsMean(c2g_dates_PB_Seattle_UPDATED, c2g_count_PB_Seattle)

deleteOutliers(c2g_dates_PB_Berlin_UPDATED, c2g_count_PB_Berlin)
deleteOutliers(c2g_dates_PB_Torino_UPDATED, c2g_count_PB_Torino)
deleteOutliers(c2g_dates_PB_Seattle_UPDATED, c2g_count_PB_Seattle)

substituteByItsMean(enj_dates_PB_Torino_UPDATED, enj_count_PB_Torino)
```

```python

deleteOutliers(enj_dates_PB_Torino_UPDATED, enj_count_PB_Torino)

# %%
# Valutare se inserire il plot nel report!!!!!!!
# fig1, ax1 = plt.subplots(layout="constrained")
# fig1.set_figwidth(12)
# ax1.plot(c2g_dates_PB_Berlin_UPDATED, c2g_count_PB_Berlin, label="Berlin", linewidth=0.5)
# ax1.plot(c2g_dates_PB_Torino_UPDATED, c2g_count_PB_Torino, label="Torino", linewidth=0.5)
# ax1.plot(c2g_dates_PB_Seattle_UPDATED, c2g_count_PB_Seattle, label="Seattle", linewidth=0.5)
# ax1.plot(enj_dates_PB_Torino_UPDATED, enj_count_PB_Torino, label="Torino E", linewidth=0.5)
# ax1.xaxis.set_major_locator(md.DayLocator(interval=7))
# ax1.xaxis.set_major_formatter(md.DateFormatter("%d/%m/%Y"))
# ax1.xaxis.set_minor_locator(md.DayLocator())
# plt.grid()
# plt.ylabel("Number of rentals per hour", rotation=90)
# plt.xticks(rotation=45, ha="right", rotation_mode="anchor")
# plt.ylim([0,2000])
# plt.title("Filtered system utilization over time - Bookings")
# plt.legend()
# #plt.savefig("/Users/marcoberti/Desktop/POLITO/ICT4SS/Secondo_anno/ICT_for_Smart_Mobility/Mellia/Utilization_PB_cities_FILTERED.png")
# plt.show()

# %%
# Creating a dataframe for each city

c2g_Berlin = pd.DataFrame({"Date": c2g_dates_PB_Berlin_UPDATED,
                            "Rentals": c2g_count_PB_Berlin})
c2g_Torino = pd.DataFrame({"Date": c2g_dates_PB_Torino_UPDATED,
                            "Rentals": c2g_count_PB_Torino})
c2g_Seattle = pd.DataFrame({"Date": c2g_dates_PB_Seattle_UPDATED,
                            "Rentals": c2g_count_PB_Seattle})
enj_Torino = pd.DataFrame({"Date": enj_dates_PB_Torino_UPDATED,
                            "Rentals": enj_count_PB_Torino})

dataframes = [c2g_Berlin, c2g_Torino, c2g_Seattle, enj_Torino]
cities = ["Berlin car2go", "Torino car2go", "Seattle car2go", "Torino enjoy"]

# %%
```

```python
def plotRollingStats(df, city):
    fig, ax = plt.subplots(layout="constrained")
    fig.set_figwidth(12)
    ax.plot(df["Date"], df["Rentals"], label="Time series", linewidth=0.5)
    ax.plot(df["Date"], df["Rolling_avg"], label="Rolling avg", linewidth=0.7)
    ax.plot(df["Date"], df["Rolling_std"], label="Rolling std", linewidth=0.7)
    ax.xaxis.set_major_locator(md.DayLocator(interval=7))
    ax.xaxis.set_major_formatter(md.DateFormatter("%d/%m/%Y"))
    ax.xaxis.set_minor_locator(md.DayLocator())
    plt.grid()
    plt.ylabel("Number of rentals per hour", rotation=90)
    plt.xticks(rotation=45, ha="right", rotation_mode="anchor")
    plt.ylim([0,max(df["Rentals"])+100])
    plt.title("Rolling statistics for " + city)
    plt.legend()
    #plt.savefig("/Users/marcoberti/Desktop/POLITO/ICT4SS/Secondo_anno/ICT_for_Smart_Mobility/Mellia/Utilization_PB_cities_FILTERED.png")
    plt.show()

def plot_ACF_PACF(df, city):
    lag_acf = acf(df["Rentals"], nlags=200)
    lag_pacf = pacf(df["Rentals"], nlags=48)

    #Plot ACF:
    plt.subplot(211)
    plt.plot(lag_acf)
    plt.axis([0, 200, -.5, 1])
    plt.axhline(y=0,linestyle='--',color='gray')
    plt.axhline(y=-1.96/np.sqrt(len(df["Rentals"])),linestyle='--',color='gray')
    plt.axhline(y=1.96/np.sqrt(len(df["Rentals"])),linestyle='--',color='gray')
    plt.title('Autocorrelation Function for ' + city)

    #Plot PACF:
    plt.subplot(212)
    plt.plot(lag_pacf)
    plt.axhline(y=0,linestyle='--',color='gray')
    plt.axhline(y=-1.96/np.sqrt(len(df["Rentals"])),linestyle='--',color='gray')
    plt.axhline(y=1.96/np.sqrt(len(df["Rentals"])),linestyle='--',color='gray')
    plt.title('Partial Autocorrelation Function for ' + city)
```

```python
    plt.tight_layout()

    plt.show()

# %%
# Adding "Rolling avg" and "Rolling var" for each time series..
for i in dataframes:
    i['Rolling_avg'] = i.Rentals.rolling(168).mean() # 168 means 168 hours in a week (= 24*7)
    i['Rolling_std'] = i.Rentals.rolling(168).std()

# %%
# Plot rolling statistics for each city
for i in range(len(dataframes)):
    plotRollingStats(dataframes[i], cities[i])

# %%
#Plot ACF and PACF for each city
for i in range(len(dataframes)):
    plot_ACF_PACF(dataframes[i], cities[i])

# %%
def meanPercentageError(real, predicted):
    sum = 0
    for i in range(len(real)):
        diff = real[i]-predicted[i]
        div = diff/real[i]
        sum += div
    return sum/len(real)*100

def meanAbsolutePercentageError(real, predicted):
    sum = 0
    for i in range(len(real)):
        diff = real[i]-predicted[i]
        div = diff/real[i]
        sum += abs(div)
    return sum/len(real)*100

# %%
def slidingWindow(df, p, d, q, N, searchGrid, city):

    train_len = N # 24 * 15 = 360
    test_len = 720 - N # 720 - 360 = 360

    prediction = np.zeros((test_len))
```

```
    for i in range(0, test_len):
        history = df["Rentals"][0 + i : train_len + i]
        arima_model = ARIMA(history, order=(p,d,q), enforce_stationarity=False)
        fitted_model = arima_model.fit(method="statespace")
        predicted_value = fitted_model.forecast()
        prediction[i] = predicted_value.values[0]

    mape = meanAbsolutePercentageError(df["Rentals"][test_len:].to_numpy(), prediction)
    mpe = meanPercentageError(df["Rentals"][test_len:].to_numpy(), prediction)

    print("\nMetrics for " + city + " with hyperparameters p=" + str(p) + " and q=" + str(q))
    print('MAPE: %.3f -- MPE: %.3f ' % (mape, mpe))

    if searchGrid == False:
        fig, ax = plt.subplots(layout="constrained")
        fig.set_figwidth(12)
        ax.plot(df["Date"][test_len:].to_numpy(), df["Rentals"][test_len:].to_numpy(), label="Real values", linewidth=0.5)
        ax.plot(df["Date"][test_len:].to_numpy(), prediction, label="Predicted values", linewidth=0.5)
        ax.xaxis.set_major_locator(md.DayLocator(interval=7))
        ax.xaxis.set_major_formatter(md.DateFormatter("%d/%m/%Y"))
        ax.xaxis.set_minor_locator(md.DayLocator())
        plt.grid()
        plt.ylabel("Number of rentals per hour", rotation=90)
        plt.xticks(rotation=45, ha="right", rotation_mode="anchor")
        plt.ylim([0,max(df["Rentals"][test_len:].to_numpy())+100])
        plt.title("Real and Predicted system utilization over time for " + city)
        plt.legend()
        #plt.savefig("/Users/marcoberti/Desktop/POLITO/ICT4SS/Secondo_anno/ICT_for_Smart_Mobility/Mellia/Utilization_PB_cities_FILTERED.png")
        plt.show()

        return (p, d, q), mape, mpe

# %%
```

```
# First attempt with p=2, d=0 (we've seen that the time series is stationary), and q=2
# Split training and test 50%/50% (360 hours of data/360 hours of data)
warnings.filterwarnings("ignore", category=UserWarning)
for i in range(len(dataframes)):
    (p, d, q), mape, mpe = slidingWindow(dataframes[i], 2, 0, 2, 360, False, cities[i])

# %%
def gridSearch(df, city):

    # Grid search with ranges p = q = [0,1,2,3,4,5] and d = 0
    triplet_list = [] #saving (p,d,q)
    mape_list = [] #saving MAPE
    mpe_list = [] # saving MPE
    for j in range(0, 6): #p ... the 6 is not included
        for k in range (0, 6): #q ... the 6 is not included
            pdq, mape, mpe = slidingWindow(df, j, 0, k, 360, True, city)
            triplet_list.append(pdq)
            mape_list.append(mape)
            mpe_list.append(mpe)
    result = pd.DataFrame({"p, d, q": triplet_list,
                           "MAPE": mape_list,
                           "MPE": mpe_list})
    return result

# %%
warnings.filterwarnings("ignore", category=UserWarning)
for i in range(len(dataframes)):
    output = gridSearch(dataframes[i], cities[i])
    print(output)
    if cities[i] == "Berlin car2go":
        output.to_csv("Berlin_c2g_GridSearch.csv", index=False) # Best result: (2,0,2)

    elif cities[i] == "Torino car2go":
        output.to_csv("Torino_c2g_GridSearch.csv", index=False) # Best result: (2,0,5) but (2,0,3) can be used

    elif cities[i] == "Seattle car2go":
        output.to_csv("Seattle_c2g_GridSearch.csv", index=False) # Best result (2,0,5)

    else: # cities[i] == "Torino enjoy":
```

```
        output.to_csv("Torino_enj_GridSearch.csv", index=False) # Best result (2,0,5) but (2,0,2) can be used


# %%
def expandingWindow_N_variable(df, p, d, q, N_list):
    # N_list is the list containing the various size of the training set
    # test_len is the fixed time window over which we predict

    train_len = 24 * 20
    test_len = 24 * 10 # Keep always the last 10 days of data for predictions

    prediction = np.zeros((len(N_list), test_len))

    for i in range(len(N_list)):
        print("\n\n" + str(N_list[i]) + "\n\n")
        for j in range(0, test_len):
            history = df["Rentals"][train_len - N_list[i] : train_len + j]
            arima_model = ARIMA(history, order=(p,d,q), enforce_stationarity=False)
            fitted_model = arima_model.fit(method="statespace")
            predicted_value = fitted_model.forecast()
            prediction[i , j] = predicted_value.values[0]

    return prediction

def slidingWindow_N_variable(df, p, d, q, N_list):
    # N_list is the list containing the various size of the training set
    # test_len is the fixed time window over which we predict

    train_len = 24 * 20
    test_len = 24 * 10 # Keep always the last 10 days of data for predictions

    prediction = np.zeros((len(N_list), test_len))

    for i in range(len(N_list)):
        print("\n\n" + str(N_list[i]) + "\n\n")
        for j in range(0, test_len):
            history = df["Rentals"][train_len - N_list[i] + j : train_len + j]
            arima_model = ARIMA(history, order=(p,d,q), enforce_stationarity=False)
            fitted_model = arima_model.fit(method="statespace")
```

```
            predicted_value = fitted_model.forecast()
            prediction[i , j] = predicted_value.values[0]

    return prediction


# %%
N_list = np.arange(24, 504, 24) # 504 and not 480 because the last element is not included

# %%
warnings.filterwarnings("ignore", category=UserWarning)

p = 2; d = 0; q = 2     # Best results from grid search for Berlin c2g
print("Sliding Window for Berlin Car2Go with ARIMA parameters (%i,%i,%i)" % (p, d, q))
pred_SLIDING_Berlin_c2g = slidingWindow_N_variable(c2g_Berlin, p, d, q, N_list)

p = 2; d = 0; q = 3     # Best results from grid search for Berlin c2g
print("Sliding Window for Torino Car2Go with ARIMA parameters (%i,%i,%i)" % (p, d, q))
pred_SLIDING_Torino_c2g = slidingWindow_N_variable(c2g_Torino, p, d, q, N_list)

p = 2; d = 0; q = 5     # Best results from grid search for Berlin c2g
print("Sliding Window for Seattle Car2Go with ARIMA parameters (%i,%i,%i)" % (p, d, q))
pred_SLIDING_Seattle_c2g = slidingWindow_N_variable(c2g_Seattle, p, d, q, N_list)

p = 2; d = 0; q = 2     # Best results from grid search for Berlin c2g
print("Sliding Window for Torino Enjoy with ARIMA parameters (%i,%i,%i)"  % (p, d, q))
pred_SLIDING_Torino_enj = slidingWindow_N_variable(enj_Torino, p, d, q, N_list)

# %%
def plot_sli_or_exp_windows(df, pred, city, sli_or_exp):
    fig, ax = plt.subplots(layout="constrained")
    fig.set_figwidth(12)
    ax.plot(df["Date"][480:], df["Rentals"][480:], label="Real", linewidth=1)

    for j in range(len(N_list)):
```

```
        ax.plot(df["Date"][480:], pred[j], label = N_list[j], linewidth=0.5)

    ax.xaxis.set_major_locator(md.DayLocator(interval=7))
    ax.xaxis.set_major_formatter(md.DateFormatter("%d/%m/%Y"))
    ax.xaxis.set_minor_locator(md.DayLocator())
    plt.grid()
    plt.ylabel("Number of rentals per hour", rotation=90)
    plt.xticks(rotation=45, ha="right", rotation_mode="anchor")
    plt.ylim([0,max(df["Rentals"])+100])
    if sli_or_exp == 0:
        plt.title("Prediction using SLIDING WINDOW for " + city)
    else: #sli_or_exp == 1
        plt.title("Prediction using EXPANDING WINDOW for " + city)
    plt.legend(ncols=11)
    if sli_or_exp == 0:
        plt.savefig("/Users/marcoberti/Desktop/POLITO/ICT4SS/Secondo_anno/ICT_for_Smart_Mobility/Mellia/Lab_2/SLIDING_W_N_Variable_" + city + ".png")
    else: #sli_or_exp == 1
        plt.savefig("/Users/marcoberti/Desktop/POLITO/ICT4SS/Secondo_anno/ICT_for_Smart_Mobility/Mellia/Lab_2/EXPANDING_W_N_Variable_" + city + ".png")
    plt.show()

# %%
plot_sli_or_exp_windows(c2g_Berlin, pred_SLIDING_Berlin_c2g, "Berlin_Car2Go", 0)
plot_sli_or_exp_windows(c2g_Torino, pred_SLIDING_Torino_c2g, "Torino_Car2Go", 0)
plot_sli_or_exp_windows(c2g_Seattle, pred_SLIDING_Seattle_c2g, "Seattle_Car2Go", 0)
plot_sli_or_exp_windows(enj_Torino, pred_SLIDING_Torino_enj, "Torino_Enjoy", 0)

# %%
warnings.filterwarnings("ignore", category=UserWarning)

p = 2; d = 0; q = 2     # Best results from grid search for Berlin c2g
print("Expanding Window for Berlin Car2Go with ARIMA parameters (%i,%i,%i)" % (p, d, q))
pred_EXPANDING_Berlin_c2g = expandingWindow_N_variable(c2g_Berlin, p, d, q, N_list)

p = 2; d = 0; q = 3     # Best results from grid search for Berlin c2g
```

```
print("Expanding Window for Torino Car2Go with ARIMA parameters (%i,%i,%i)" % (p, d, q))
pred_EXPANDING_Torino_c2g = expandingWindow_N_variable(c2g_Torino, p, d, q, N_list)

p = 2; d = 0; q = 5     # Best results from grid search for Berlin c2g
print("Expanding Window for Seattle Car2Go with ARIMA parameters (%i,%i,%i)"  % (p, d, q))
pred_EXPANDING_Seattle_c2g = expandingWindow_N_variable(c2g_Seattle, p, d, q, N_list)

p = 2; d = 0; q = 2     # Best results from grid search for Berlin c2g
print("Expanding Window for Torino Enjoy with ARIMA parameters (%i,%i,%i)" % (p, d, q))
pred_EXPANDING_Torino_enj = expandingWindow_N_variable(enj_Torino, p, d, q, N_list)

# %%
plot_sli_or_exp_windows(c2g_Berlin, pred_EXPANDING_Berlin_c2g, "Berlin_Car2Go", 1)
plot_sli_or_exp_windows(c2g_Torino, pred_EXPANDING_Torino_c2g, "Torino_Car2Go", 1)
plot_sli_or_exp_windows(c2g_Seattle, pred_EXPANDING_Seattle_c2g, "Seattle_Car2Go", 1)
plot_sli_or_exp_windows(enj_Torino, pred_EXPANDING_Torino_enj, "Torino_Enjoy", 1)

# %%
def compute_MAPE_and_MPE_per_Prediction(df, predicted, N_list):
    real = df["Rentals"][480:].to_numpy()
    MAPE_list = []
    MPE_list = []
    for row in predicted: #predicted is the bidimensional array
        mape = meanAbsolutePercentageError(real, row)
        mpe = meanPercentageError(real, row)
        MAPE_list.append(mape)
        MPE_list.append(mpe)
    result = pd.DataFrame({"Train": N_list,
                           "MAPE": MAPE_list,
                           "MPE": MPE_list})
    return result

# %%
stats_SLIDING_Berlin_c2g = compute_MAPE_and_MPE_per_Prediction(c2g_Berlin, pred_SLIDING_Berlin_c2g, N_list)
```

```python
stats_SLIDING_Berlin_c2g.to_csv("Stats_SLIDING_Berlin_c2g.csv", index=False)
stats_SLIDING_Torino_c2g = compute_MAPE_and_MPE_per_Prediction(c2g_Torino, pred_SLIDING_Torino_c2g, N_list)
stats_SLIDING_Torino_c2g.to_csv("Stats_SLIDING_Torino_c2g.csv", index=False)
stats_SLIDING_Seattle_c2g = compute_MAPE_and_MPE_per_Prediction(c2g_Seattle, pred_SLIDING_Seattle_c2g, N_list)
stats_SLIDING_Seattle_c2g.to_csv("Stats_SLIDING_Seattle_c2g.csv", index=False)
stats_SLIDING_Torino_enj = compute_MAPE_and_MPE_per_Prediction(enj_Torino, pred_SLIDING_Torino_enj, N_list)
stats_SLIDING_Torino_enj.to_csv("Stats_SLIDING_Torino_enj.csv", index=False)

stats_EXPANDING_Berlin_c2g = compute_MAPE_and_MPE_per_Prediction(c2g_Berlin, pred_EXPANDING_Berlin_c2g, N_list)
stats_EXPANDING_Berlin_c2g.to_csv("Stats_EXPANDING_Berlin_c2g.csv", index=False)
stats_EXPANDING_Torino_c2g = compute_MAPE_and_MPE_per_Prediction(c2g_Torino, pred_EXPANDING_Torino_c2g, N_list)
stats_EXPANDING_Torino_c2g.to_csv("Stats_EXPANDING_Torino_c2g.csv", index=False)
stats_EXPANDING_Seattle_c2g = compute_MAPE_and_MPE_per_Prediction(c2g_Seattle, pred_EXPANDING_Seattle_c2g, N_list)
stats_EXPANDING_Seattle_c2g.to_csv("Stats_EXPANDING_Seattle_c2g.csv", index=False)
stats_EXPANDING_Torino_enj = compute_MAPE_and_MPE_per_Prediction(enj_Torino, pred_EXPANDING_Torino_enj, N_list)
stats_EXPANDING_Torino_enj.to_csv("Stats_EXPANDING_Torino_enj.csv", index=False)

# %%
warnings.filterwarnings("ignore", category=UserWarning)

#This method can be used at first for the optimal arima parameters for each city, and then varying p!
def predict_more_values(df, p, d, q, N, h):

    train_len = N # 24 * 15 = 360
    test_len = 720 - N # 720 - 360 = 360

    prediction = np.zeros((test_len))
```

```python

    for i in range(0, test_len, h):
        if train_len+h < len(df["Rentals"]):
            history = df["Rentals"][i : train_len + i]
        else:
            history = df["Rentals"][i :]
        arima_model = ARIMA(history, order=(p,d,q), enforce_stationarity=False)
        fitted_model = arima_model.fit(method="statespace")
        predicted_value = fitted_model.forecast(steps=h)
        if i+h < test_len:
            prediction[i:i+h] = predicted_value.values
        else:
            prediction[i:] = predicted_value.values[:test_len%i]

    return prediction, h


# %%
for i in range(len(dataframes)):
    if cities[i] == "Berlin car2go":
        p = 2; d = 0; q = 2
        h6, n6 = predict_more_values(dataframes[i], p, d, q, 360, 6)
        h12, n12 = predict_more_values(dataframes[i], p, d, q, 360, 12)
        h18, n18 = predict_more_values(dataframes[i], p, d, q, 360, 18)
        h24, n24 = predict_more_values(dataframes[i], p, d, q, 360, 24)
        h48, n48 = predict_more_values(dataframes[i], p, d, q, 360, 48)
        h72, n72 = predict_more_values(dataframes[i], p, d, q, 360, 72)
        h96, n96 = predict_more_values(dataframes[i], p, d, q, 360, 96)
        h120, n120 = predict_more_values(dataframes[i], p, d, q, 360, 120)
        h144, n144 = predict_more_values(dataframes[i], p, d, q, 360, 144)
        h168, n168 = predict_more_values(dataframes[i], p, d, q, 360, 168)
        print("Berlin car2go DONE")

        h_list = [h6, h12, h18, h24, h48, h72, h96, h120, h144, h168]
        n_list = [n6, n12, n18, n24, n48, n72, n96, n120, n144, n168]

        listone_Berlin_c2g = [h_list, n_list]

    elif cities[i] == "Torino car2go":
        p = 2; d = 0; q = 3
        h6, n6 = predict_more_values(dataframes[i], p, d, q, 360, 6)
        h12, n12 = predict_more_values(dataframes[i], p, d, q, 360, 12)
        h18, n18 = predict_more_values(dataframes[i], p, d, q, 360, 18)
        h24, n24 = predict_more_values(dataframes[i], p, d, q, 360, 24)
```

```
        h48, n48 = predict_more_values(dataframes[i], p, d, q, 360, 48)
        h72, n72 = predict_more_values(dataframes[i], p, d, q, 360, 72)
        h96, n96 = predict_more_values(dataframes[i], p, d, q, 360, 96)
        h120, n120 = predict_more_values(dataframes[i], p, d, q, 360, 120)
        h144, n144 = predict_more_values(dataframes[i], p, d, q, 360, 144)
        h168, n168 = predict_more_values(dataframes[i], p, d, q, 360, 168)
        print("Torino car2go DONE")

        h_list = [h6, h12, h18, h24, h48, h72, h96, h120, h144, h168]
        n_list = [n6, n12, n18, n24, n48, n72, n96, n120, n144, n168]

        listone_Torino_c2g = [h_list, n_list]

    elif cities[i] == "Seattle car2go":
        p = 2; d = 0; q = 5
        h6, n6 = predict_more_values(dataframes[i], p, d, q, 360, 6)
        h12, n12 = predict_more_values(dataframes[i], p, d, q, 360, 12)
        h18, n18 = predict_more_values(dataframes[i], p, d, q, 360, 18)
        h24, n24 = predict_more_values(dataframes[i], p, d, q, 360, 24)
        h48, n48 = predict_more_values(dataframes[i], p, d, q, 360, 48)
        h72, n72 = predict_more_values(dataframes[i], p, d, q, 360, 72)
        h96, n96 = predict_more_values(dataframes[i], p, d, q, 360, 96)
        h120, n120 = predict_more_values(dataframes[i], p, d, q, 360, 120)
        h144, n144 = predict_more_values(dataframes[i], p, d, q, 360, 144)
        h168, n168 = predict_more_values(dataframes[i], p, d, q, 360, 168)
        print("Seattle car2go DONE")

        h_list = [h6, h12, h18, h24, h48, h72, h96, h120, h144, h168]
        n_list = [n6, n12, n18, n24, n48, n72, n96, n120, n144, n168]

        listone_Seattle_c2g = [h_list, n_list]

    else: # cities[i] == "Torino enjoy":
        p = 2; d = 0; q = 2
        h6, n6 = predict_more_values(dataframes[i], p, d, q, 360, 6)
        h12, n12 = predict_more_values(dataframes[i], p, d, q, 360, 12)
        h18, n18 = predict_more_values(dataframes[i], p, d, q, 360, 18)
        h24, n24 = predict_more_values(dataframes[i], p, d, q, 360, 24)
        h48, n48 = predict_more_values(dataframes[i], p, d, q, 360, 48)
        h72, n72 = predict_more_values(dataframes[i], p, d, q, 360, 72)
        h96, n96 = predict_more_values(dataframes[i], p, d, q, 360, 96)
        h120, n120 = predict_more_values(dataframes[i], p, d, q, 360, 120)
        h144, n144 = predict_more_values(dataframes[i], p, d, q, 360, 144)
        h168, n168 = predict_more_values(dataframes[i], p, d, q, 360, 168)
        print("Torino enjoy DONE")
```

```
        h_list = [h6, h12, h18, h24, h48, h72, h96, h120, h144, h168]
        n_list = [n6, n12, n18, n24, n48, n72, n96, n120, n144, n168]

        listone_Torino_enj = [h_list, n_list]


# %%
def plot_different_h(df, listone, city, p_variable, p_val):

    if p_variable == False:
        if city == "Berlin_c2g":
            p = 2; d = 0; q = 2
        elif city == "Torino_c2g":
            p = 2; d = 0; q = 3
        elif city == "Seattle_c2g":
            p = 2; d = 0; q = 5
        else: #city == "Torino_enj"
            p = 2; d = 0; q = 2

        fig, ax = plt.subplots(layout="constrained")
        fig.set_figwidth(12)
        ax.plot(df["Date"][360:], df["Rentals"][360:], label="Real", linewidth=1)

        for j in range(len(listone[0])):
            ax.plot(df["Date"][360:], listone[0][j], label = f"h = {listone[1][j]}", linewidth=0.5)

        ax.xaxis.set_major_locator(md.DayLocator(interval=7))
        ax.xaxis.set_major_formatter(md.DateFormatter("%d/%m/%Y"))
        ax.xaxis.set_minor_locator(md.DayLocator())
        plt.grid()
        plt.ylabel("Number of rentals per hour", rotation=90)
        plt.xticks(rotation=45, ha="right", rotation_mode="anchor")
        plt.ylim([0,max(df["Rentals"])+100])
        plt.title("Prediction with h variable for " + city + " with ARIMA parameters (%i, %i, %i)" % (p, d, q))
        plt.legend(ncols=5)
        plt.savefig("/Users/marcoberti/Desktop/POLITO/ICT4SS/Secondo_anno/ICT_for_Smart_Mobility/Mellia/Lab_2/Plot_h_variable_p=" + str(p) + "_" + city + ".png")
        plt.show()

    else: # p_variable == True
```

```
        if city == "Berlin_c2g":
            d = 0; q = 2
        elif city == "Torino_c2g":
            d = 0; q = 3
        elif city == "Seattle_c2g":
            d = 0; q = 5
        else: #city == "Torino_enj"
            d = 0; q = 2

        fig, ax = plt.subplots(layout="constrained")
        fig.set_figwidth(12)
        ax.plot(df["Date"][360:], df["Rentals"][360:], label="Real", linewidth=1)

        for j in range(len(listone[0])):
            ax.plot(df["Date"][360:], listone[0][j], label = f"h = {listone[1][j]}", linewidth=0.5)

        ax.xaxis.set_major_locator(md.DayLocator(interval=7))
        ax.xaxis.set_major_formatter(md.DateFormatter("%d/%m/%Y"))
        ax.xaxis.set_minor_locator(md.DayLocator())
        plt.grid()
        plt.ylabel("Number of rentals per hour", rotation=90)
        plt.xticks(rotation=45, ha="right", rotation_mode="anchor")
        plt.ylim([0,max(df["Rentals"])+100])
        plt.title("Prediction with h variable for " + city + " with ARIMA parameters (%i, %i, %i)" % (p_val, d, q))
        plt.legend(ncols=5)
        plt.savefig("/Users/marcoberti/Desktop/POLITO/ICT4SS/Secondo_anno/ICT_for_Smart_Mobility/Mellia/Lab_2/Plot_h_variable_p=" + str(p_val) + "_" + city + ".png")
        plt.show()


# %%
plot_different_h(c2g_Berlin, listone_Berlin_c2g, "Berlin_c2g", False, 0)
plot_different_h(c2g_Torino, listone_Torino_c2g, "Torino_c2g", False, 0)
plot_different_h(c2g_Seattle, listone_Seattle_c2g, "Seattle_c2g", False, 0)
plot_different_h(enj_Torino, listone_Torino_enj, "Torino_enj", False, 0)

# %%
warnings.filterwarnings("ignore", category=UserWarning)

p_list = [4, 8, 12, 24, 48]

```

```
for p in p_list:
    for i in range(len(dataframes)):
        if cities[i] == "Berlin car2go":
            d = 0; q = 2
            h6, n6 = predict_more_values(dataframes[i], p, d, q, 360, 6)
            h12, n12 = predict_more_values(dataframes[i], p, d, q, 360, 12)
            h18, n18 = predict_more_values(dataframes[i], p, d, q, 360, 18)
            h24, n24 = predict_more_values(dataframes[i], p, d, q, 360, 24)
            h48, n48 = predict_more_values(dataframes[i], p, d, q, 360, 48)
            h72, n72 = predict_more_values(dataframes[i], p, d, q, 360, 72)
            h96, n96 = predict_more_values(dataframes[i], p, d, q, 360, 96)
            h120, n120 = predict_more_values(dataframes[i], p, d, q, 360,
             ↪  120)
            h144, n144 = predict_more_values(dataframes[i], p, d, q, 360,
             ↪  144)
            h168, n168 = predict_more_values(dataframes[i], p, d, q, 360,
             ↪  168)
            print("Berlin car2go DONE")

            h_list = [h6, h12, h18, h24, h48, h72, h96, h120, h144, h168]
            n_list = [n6, n12, n18, n24, n48, n72, n96, n120, n144, n168]

            listone_Berlin_c2g = [h_list, n_list]

        elif cities[i] == "Torino car2go":
            d = 0; q = 3
            h6, n6 = predict_more_values(dataframes[i], p, d, q, 360, 6)
            h12, n12 = predict_more_values(dataframes[i], p, d, q, 360, 12)
            h18, n18 = predict_more_values(dataframes[i], p, d, q, 360, 18)
            h24, n24 = predict_more_values(dataframes[i], p, d, q, 360, 24)
            h48, n48 = predict_more_values(dataframes[i], p, d, q, 360, 48)
            h72, n72 = predict_more_values(dataframes[i], p, d, q, 360, 72)
            h96, n96 = predict_more_values(dataframes[i], p, d, q, 360, 96)
            h120, n120 = predict_more_values(dataframes[i], p, d, q, 360,
             ↪  120)
            h144, n144 = predict_more_values(dataframes[i], p, d, q, 360,
             ↪  144)
            h168, n168 = predict_more_values(dataframes[i], p, d, q, 360,
             ↪  168)
            print("Torino car2go DONE")

            h_list = [h6, h12, h18, h24, h48, h72, h96, h120, h144, h168]
            n_list = [n6, n12, n18, n24, n48, n72, n96, n120, n144, n168]

            listone_Torino_c2g = [h_list, n_list]
```

```python

        elif cities[i] == "Seattle car2go":
            d = 0; q = 5
            h6, n6 = predict_more_values(dataframes[i], p, d, q, 360, 6)
            h12, n12 = predict_more_values(dataframes[i], p, d, q, 360, 12)
            h18, n18 = predict_more_values(dataframes[i], p, d, q, 360, 18)
            h24, n24 = predict_more_values(dataframes[i], p, d, q, 360, 24)
            h48, n48 = predict_more_values(dataframes[i], p, d, q, 360, 48)
            h72, n72 = predict_more_values(dataframes[i], p, d, q, 360, 72)
            h96, n96 = predict_more_values(dataframes[i], p, d, q, 360, 96)
            h120, n120 = predict_more_values(dataframes[i], p, d, q, 360,
             ↪  120)
            h144, n144 = predict_more_values(dataframes[i], p, d, q, 360,
             ↪  144)
            h168, n168 = predict_more_values(dataframes[i], p, d, q, 360,
             ↪  168)
            print("Seattle car2go DONE")

            h_list = [h6, h12, h18, h24, h48, h72, h96, h120, h144, h168]
            n_list = [n6, n12, n18, n24, n48, n72, n96, n120, n144, n168]

            listone_Seattle_c2g = [h_list, n_list]

        else: # cities[i] == "Torino enjoy":
            d = 0; q = 2
            h6, n6 = predict_more_values(dataframes[i], p, d, q, 360, 6)
            h12, n12 = predict_more_values(dataframes[i], p, d, q, 360, 12)
            h18, n18 = predict_more_values(dataframes[i], p, d, q, 360, 18)
            h24, n24 = predict_more_values(dataframes[i], p, d, q, 360, 24)
            h48, n48 = predict_more_values(dataframes[i], p, d, q, 360, 48)
            h72, n72 = predict_more_values(dataframes[i], p, d, q, 360, 72)
            h96, n96 = predict_more_values(dataframes[i], p, d, q, 360, 96)
            h120, n120 = predict_more_values(dataframes[i], p, d, q, 360,
             ↪  120)
            h144, n144 = predict_more_values(dataframes[i], p, d, q, 360,
             ↪  144)
            h168, n168 = predict_more_values(dataframes[i], p, d, q, 360,
             ↪  168)
            print("Torino enjoy DONE")

            h_list = [h6, h12, h18, h24, h48, h72, h96, h120, h144, h168]
            n_list = [n6, n12, n18, n24, n48, n72, n96, n120, n144, n168]

            listone_Torino_enj = [h_list, n_list]

```

```
    plot_different_h(c2g_Berlin, listone_Berlin_c2g, "Berlin_c2g", True, 0)
    plot_different_h(c2g_Torino, listone_Torino_c2g, "Torino_c2g", True, 0)
    plot_different_h(c2g_Seattle, listone_Seattle_c2g, "Seattle_c2g", True,
     ↪  0)
    plot_different_h(enj_Torino, listone_Torino_enj, "Torino_enj", True, 0)
```

	Sliding window		Expanding window	
N	MAPE	MPE	MAPE	MPE
24	53.239	-32.106	40.273	-22.496
48	36.803	-18.796	35.416	-16.758
72	35.394	-17.269	34.490	-17.422
96	34.775	-15.660	34.949	-19.622
120	35.490	-17.607	34.528	-19.774
144	36.627	-19.173	36.680	-20.740
168	37.062	-20.548	35.405	-20.238
192	36.743	-21.317	34.634	-19.389
216	35.306	-20.092	33.963	-18.184
240	35.485	-20.084	33.656	-17.561
264	34.229	-18.550	33.962	-17.892
288	34.268	-18.505	33.898	-17.880
312	34.420	-18.763	34.058	-18.066
336	34.486	-18.771	33.933	-17.844
360	34.349	-18.429	34.025	-17.439
384	34.004	-18.020	33.478	-16.502
408	33.821	-17.554	33.112	-16.148
432	33.767	-17.298	33.152	-16.347
456	33.785	-17.324	33.380	-16.557
480	33.959	-17.380	33.647	-17.075

Table 3: Parameter N tuning for Torino_Enjoy with $d = 0$, $p = 2$ and $q = 2$.