ICT for smart mobility

Laboratory 2

Alessandro Redi, s310471
Marzio Reitano, s319569
Federico Volponi, s309709

Professors: Marco Mellia, Luca Vassio

Politecnico di Torino
Academic Year 2023-2024

Contents

1 Preliminary analysis

1.1 Time-series extraction and filtering

why such period (?)

The time series of the three cities is based on the number of rentals grouped for each hour of the day. The period under analysis starts from the first to the thirty of October 2017, 30 days in total. The data are filtered with the same criteria applied in the first laboratory: are considered as bookings only for the cars that moved and whose duration is greater than 2 minutes and less than 180 minutes. ✓

1.2 Preprocessing

The first thing that needs to be done is to add some minimum random values to the entries. This is done to avoid integer entries and optimize the next operation during the fitting of the model. The next step is to check if some data is missing within the time series. This is done because ARIMA models expect as input a constant-spaced time series. The number of entries must be in total 720[1], but in our case are 712 with the missing data at the same time for all three cities. One easy solution is to replace a missing value with the value of the previous sample. This works since we can assume that in one hour the number of rentals does not change much.

↳ what if there are consecutive missing values (?)

1.3 Stationary of the time series

After the preprocessing, the first thing to do is to check if the time series is stationary.
This can be done by plotting the rentals, as shown in figure 1.3.1 to check visually if there are any clear trends or seasonality. The plots reported are of Milano, but the similar conclusions can be drawn from the other two cities as well.

over which ΔT?

A more reliable approach to check the stationarity of a time series is done using the rolling statistics test, because at first sight, due to the continuous fluctuations, the plot of the time series is not clear enough to determine if it is stationary or not. The rolling statistics technique is done both on the average of the values and on its standard deviation, as shown in figure 1.3.1. The method evaluates the average and standard deviation for every time window assigned, which in our case is done both on a week and two-week window. If the results of the rolling statistics are constant over time then the time series is stationary. ✓
Thanks to these conclusions we can set to $d = 0$ the parameter of the Integrated part of the ARIMA model, meaning that there is no need to perform differencing. ✓
The other images are in section 3.3.

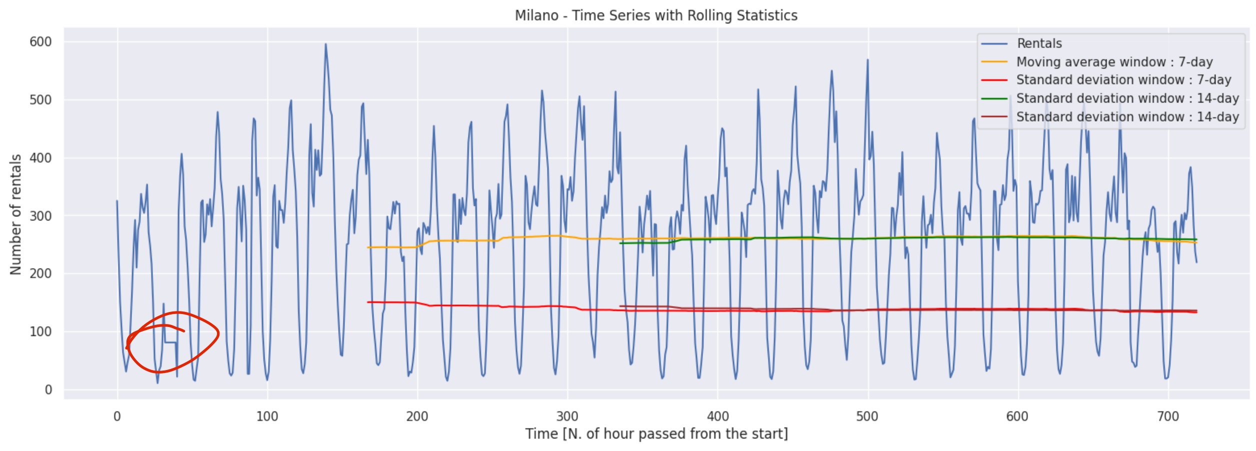


Figure 1.3.1: Rolling statistics of moving average and standard deviation of Montreal city

1.4 ACF and PCF

After determining the stationarity of the time series, the ACF[2] and PACF[3] are evaluated, from which the hyperparameters p and q can be estimated. Indeed, for the Autoregressive model p is evaluated by looking after how many lags the PACF becomes negligible, and for the Moving Average models q is set by checking how many lags after the ACF stops. In graph 1.4.1 it is possible to see a correlation over seven days (168 lags) and

under which conditions (?)

[1]One entry per hour multiplied by 24 hours and 30 days
[2]Autocorrelation Function
[3]Partial Autocorrelation Function

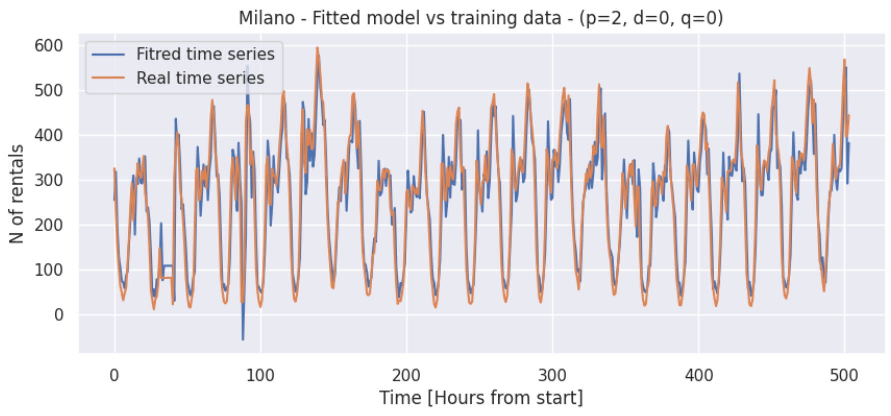


Figure 2.2.1: AR(2) model for Milano time series

between day and night (negative correlation). Analyzing more precisely the PACF in figure 1.4.3 it is possible to see that a good value of p is equal to 2. This choice can be made because the PACF will become negligible after more or less 2 lags. Indeed this will work well as an Autoregressive model and not as a Moving Average because looking at the figure 1.4.2 a value of q cannot be easily chosen. If instead, an ARMA process is the objective, then it will become impossible to have a first guess of the parameters p and q, because following the previous rules does not guarantee to obtain the best performance. Indeed a grid search has to be done to obtain the best set of parameters that will obtain the lower errors with at the same time the lower complexity. ✓

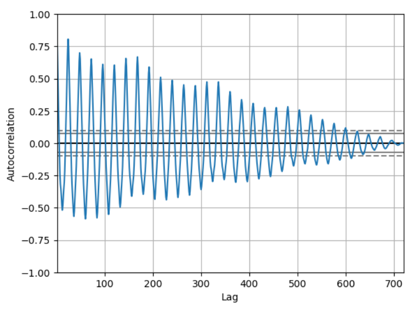


Figure 1.4.1: ACF

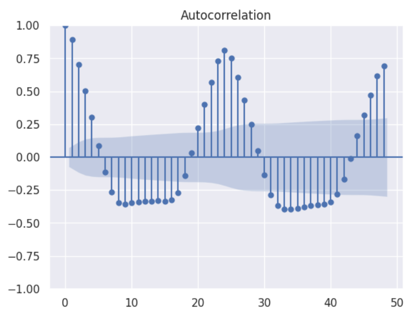


Figure 1.4.2: ACF short lag

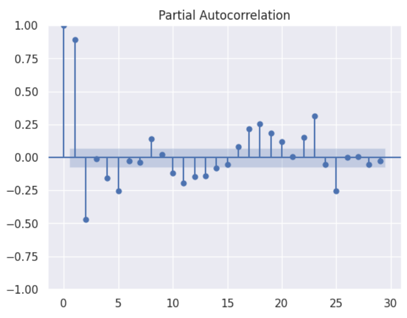


Figure 1.4.3: PACF

2 ARIMA model

After the preliminary analysis of the time series done in the previous section, in the following part, the ARIMA model is used to predict the future values in the series after an appropriate choice of the p,d, and q hyperparameters.

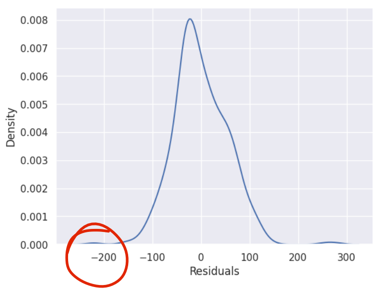
2.1 Training and test sets

At first, it is necessary to divide the 30-day data into the training and test sets. A reasonable is split to use 3 weeks for the training and the remaining one for the test. It is important to notice that the sets are divided into weeks to preserve the weekly correlation that came to light when analyzing figure 1.4.1. ✓

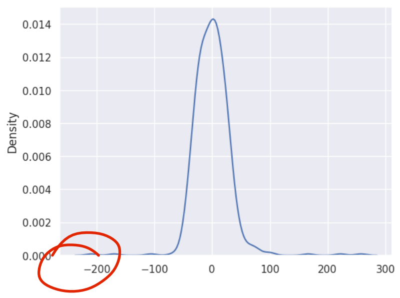
2.2 Choice of the hyperparameters

Thanks to the information drawn from the analysis of the ACF and PACF, it is reasonable to try to fit the training set using an AR(2) model. This choice makes sense for the three cities under analysis. The comparison between the fitted data and the training data of Milano Car2Go is shown in 2.2.1. In 2.2.2 are reported the Kernel Density Estimates of Milano, Torino, and Montreal. If the model fits the data well we expect the residual to be Normally distributed: in this sense, Torino is the best one in terms of distribution and mean, while the other two, especially Milano, have some bumps, and the mean is not exactly zero. Evaluating the Mean Absolute Percentage error of the fitted model with respect to the training data shows that the model works similarly for the three cities. The results are reported in table 1.

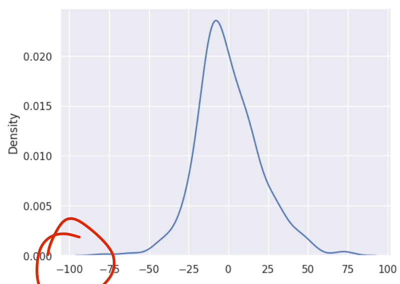
not really ... different scales do not allow a fair comparison



(a) *KDE of the residuals in Milano*



(b) *KDE of the residuals in Torino*



(c) *KDE of the residuals in Montreal*

Figure 2.2.2: Kernel density estimate of the residuals

	Milano	Torino	Montreal
MAPE	17.05	25.12	23.90

Table 1: Mean Absolute Percentage Error of the fitted model

2.2.1 Grid search

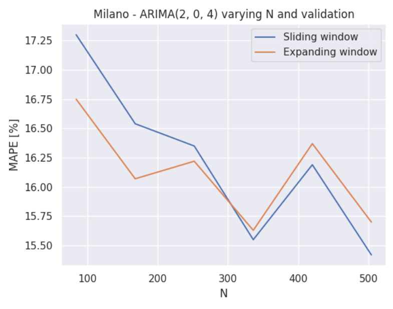
To definitively choose the hyperparameters, making a grid search of p and q (d remains set to zero) helps in this task. Now the errors are computed on the test set using the sliding window approach.
Table 2 reports the MSE, MAPE, and R2 score of the predicted data with respect to the test set for Milano. As expected the more complex the model the smaller the error; nevertheless, having a complex model has a high cost in terms of the time required. This table confirms that the AR(2) model was a good guess about the choice of hyperparameters since it is the model that achieves the best R2 score among the low complex models. By increasing the complexity to an ARMA(2, 4) model it is possible to achieve an R2 score of 0.86; although it is not a great improvement with respect to the AR(2) model we choose to proceed with the analysis using this model since is the one that achieves one of the best solutions with lowest complexity. figura 1.4.2 Instead, for Torino and Montreal the best trade-off between performance and complexity is achieved using respectively an ARMA(2, 3), which achieves an R2 score of 0.66, and an ARMA(2,4), which achieves an R2 score of 0.78. A note must be done on the ARIMA(0,0,0) model since it is not useful to fit the time series because it represents the mean of the number of rentals in the selected period. ✓ ? ✓

2.3 Choice of N and learning strategy

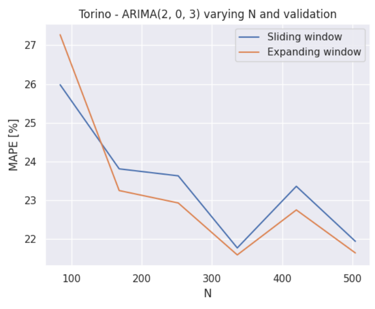
did you fix the test set ?

In the next step, the errors are evaluated by varying the number of training samples and changing the validation strategy (sliding window and expanding window): the outcome is shown in figure 2.3.1 which depicts how the MAPE changes varying N. Regarding the number of samples used for the training, the error tends to decrease for higher N for both the validation strategies. Even if some peaks are breaking the trend, we can notice that there is a big impact on performance when the training set is smaller or equal to a week of data. Concerning the validation strategy, on the other hand, there is a high dependence on the city under analysis: indeed, for Milano expanding window seems to work better with high N, for Torino the behavior is the opposite, while for Montreal the expanding window is able to obtain better results expect only for $N = 504$. ✓ ?
How the relative error changes w.r.t. the absolute number of rentals? LU decomposition error? ?

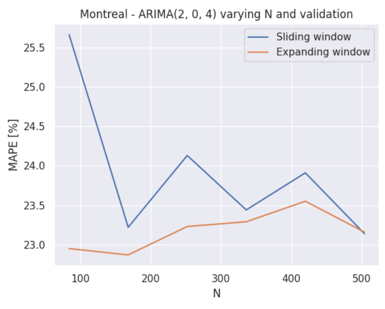
too coarse



(a) *MAPE variation for Milano*



(b) *MAPE variation for Torino*



(c) *MAPE variation for Montreal*

Figure 2.3.1: MAPE versus the number of samples in training set on different validation strategies

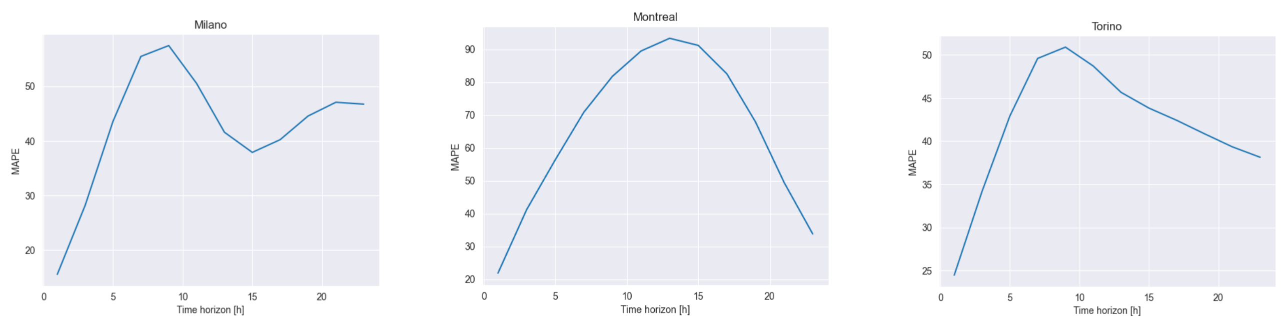
Model	MSE	MAPE	R2 score	time [s]
”(0, 0, 0)”	17655.96	43.16	0.01	12.7
”(0, 0, 1)”	6534.48	26.02	0.63	12.66
”(0, 0, 2)”	3969.37	20.07	0.78	23.14
”(0, 0, 3)”	3524.89	19.06	0.8	32.53
”(0, 0, 4)”	3207.54	17.89	0.82	37.38
”(1, 0, 0)”	3787.88	18.93	0.79	10.78
”(1, 0, 1)”	3008.28	16.77	0.83	16.57
”(1, 0, 2)”	2991.55	16.9	0.83	24.87
”(1, 0, 3)”	3002.86	16.91	0.83	34.92
”(1, 0, 4)”	2876.67	16.38	0.84	56.83
”(2, 0, 0)”	2990.35	16.81	0.83	12.33
”(2, 0, 1)”	3073.26	17.02	0.83	17.57
”(2, 0, 2)”	2865.03	16.12	0.84	41.51
”(2, 0, 3)”	2651.33	15.51	0.85	47.26
”(2, 0, 4)”	2517.28	15.39	0.86	57.45
”(3, 0, 0)”	2997.02	16.85	0.83	14.53
”(3, 0, 1)”	2985.66	16.84	0.83	36.55
”(3, 0, 2)”	3155.71	17.09	0.82	63.28
”(3, 0, 3)”	2591.77	15.3	0.85	90.45
”(3, 0, 4)”	2553.88	15.51	0.86	106.96
”(4, 0, 0)”	2763.1	15.9	0.84	23.94
”(4, 0, 1)”	2802.41	16.07	0.84	61.04
”(4, 0, 2)”	2950.29	16.69	0.83	103.95
”(4, 0, 3)”	2577.51	15.47	0.86	102.54
”(4, 0, 4)”	2457.43	15.42	0.86	191.94

Table 2: Errors of different models for Milano

2.4 Optional: Horizon Impact

In the previous section, we discussed forecasts for the upcoming hour t+1. Now, we focus on the concept of the forecast horizon, denoted as h, which represents the timeframe into the future for generating predictions. This section aims to evaluate the accuracy of the forecast horizon by examining various metrics of errors. We present graphical representations depicting the MAPE performance as h varies for a model with optimized hyperparameters p, d, and q, focusing on the three cities. The estimation time target is set at t+h, and we increment the horizon values within the range of [1 hour, 24 hours] with a 2-hour interval.
Referencing Figure 2.4.1, we observe an upward trend in MAPE across all scenarios. It becomes apparent that projecting beyond one hour into the future results in the loss of information concerning the time intervals between the last observed data and the forecast horizon.
Furthermore, we can observe the Montreal case, Figure 2.4.1b, characterized by a stronger sense of regularity. After reaching its peak at the 12-hour horizon, the MAPE for Montreal displays a return to the typical daily trend, indicating reduced susceptibility to estimation errors as h approaches 24 hours. why this happens?

In Table 3, it is possible to observe the variation of MAPE value for each horizon h, at each value of p. The

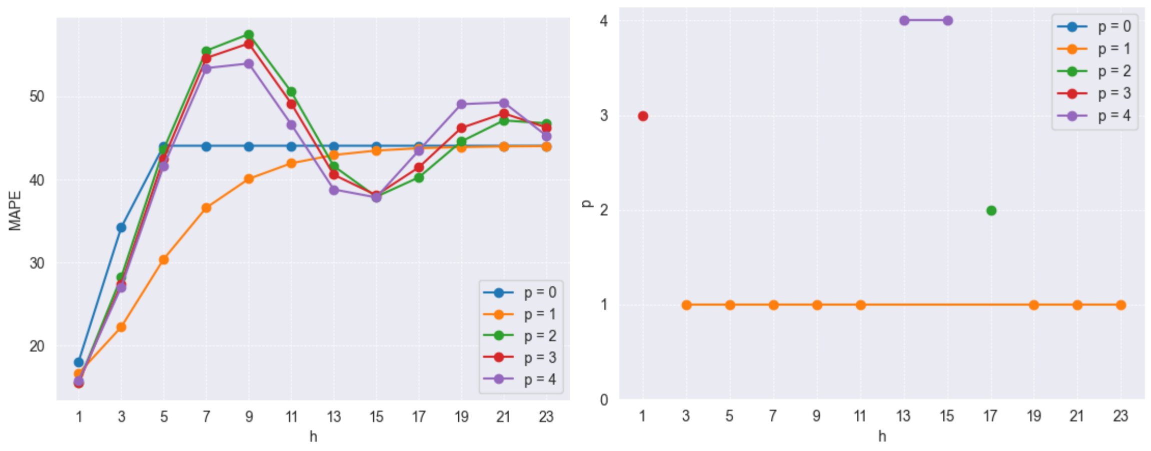


(a) *Milano* (b) *Montreal* (c) *Torino*

Figure 2.4.1: MAPE trend with respect to changes in the horizon values.

MAPE	p = 0	p = 1	p = 2	p = 3	p = 4
h = 1	18.02	16.62	15.52	15.50	15.71
h = 3	34.16	22.18	28.17	27.34	26.99
h = 5	44.02	30.37	43.58	42.46	41.61
h = 7	44.02	36.55	55.47	54.59	53.37
h = 9	44.02	40.03	57.46	56.34	53.93
h = 11	44.02	41.91	50.52	49.11	46.60
h = 13	44.02	42.91	41.57	40.53	38.76
h = 15	44.02	43.44	37.88	38.07	37.80
h = 17	44.02	43.72	40.21	41.42	43.45
h = 19	44.02	43.86	44.54	46.17	49.03
h = 21	44.02	43.94	47.07	47.92	49.24
h = 23	44.02	43.98	46.72	46.22	45.23

Table 3: MAPE at each h-value according to p-value for Milano



(a) Comparison of different models *(b) p-value with lowest MAPE for each row of the Table 3*

?

Figure 2.4.2: MAPE trend using different models for Milano.

comparison of the performance of the five different models, again in terms of MAPE, is depicted in Figure 2.4.2a, while in Figure 2.4.2b a graph has been added to more easily interpret the values given in the table, and it shows, for each value of h, which model has the lowest MAPE. It also allows us to observe how many times the different models turn out to be the best. The model with p=1 emerges as the optimal choice across 8 out of 12 different values of h. It is surpassed by more complex models (p=2 and p=4) only for h values nearing 12, that is during the period when all models exhibit their worst performance. Based on this empirical examination, we can infer that simplest models excel for h in proximity to 1 and, consequently, h close to 24, owing to the daily periodic trend in rentals. On the other hand, when predictions have to be made for h close to the value of the middle of a period, then the best predictions are made by the most complex models.

- complete and well done
- missing comparison with a baseline

5⁻/5

3 Appendix

3.1 Point 1

```
startDate = datetime(2017, 10, 1, 0, 0, 0)
endDate = datetime(2017, 11, 1, 0, 0, 0)

startTime = datetime.timestamp(startDate)
endTime = datetime.timestamp(endDate)

permBooking_enjoy = db["enjoy_PermanentBookings"]
permParkings_enjoy = db["enjoy_PermanentParkings"]
permBooking = db["PermanentBookings"]
permParkings = db["PermanentParkings"]

def pipeline_count_hour_filt_bookings(city):
    pipeline = [
        {
            "$match": {
            "$and":[
                {"city": city},
                {"init_time": {"$gte": startTime}},
                {"final_time": {"$lte": endTime}}
            ]
        }
        },
        {
            "$project": {
                "year": { "$year": "$init_date" },
                "month": { "$month": "$init_date" },
                "day": { "$dayOfMonth": "$init_date" },
                "hour": { "$hour": "$init_date" },
                "duration": {"$divide": [{"$subtract": ["$final_time", "$init_time"]}, 60]
                "moved" : { "$ne": [
                        {"$arrayElemAt": [ "$origin_destination.coordinates", 0]},
                        {"$arrayElemAt": [ "$origin_destination.coordinates", 1]}
                        ]
            }

            }
        },
        {
            "$match": {
                "duration": {"$lte": 180, "$gte": 2},
                "moved": True
            }
        },
        {
            "$group": {
                "_id": {"year": "$year", "month": "$month", "day": "$day", "hour": "$hour"
                "count": {"$sum": 1},
            }
        },
        {
        "$project":{
            "year" : "$_id.year",
            "month" : "$_id.month",
            "day" : "$_id.day",
            "hour" : "$_id.hour",
            "duration": 1,
            "count" : 1,
                "_id": 0
            }
        }

        ]


        return pipeline

        def utilizationPerHour(cursor):
        df = pd.DataFrame(list(cursor))
        df_date = pd.to_datetime(df[["year", "month", "day", "hour"]])
        df = df.set_index(df_date)
        df= df.sort_index()

        return df


cursorMon = permBooking.aggregate(pipeline_count_hour_filt_bookings("Montreal"))
cursorMi = permBooking.aggregate(pipeline_count_hour_filt_bookings("Milano"))
cursorTo = permBooking_enjoy.aggregate(pipeline_count_hour_filt_bookings("Torino"))

dfMon = utilizationPerHour(cursorMon)
dfMi = utilizationPerHour(cursorMi)
dfTo = utilizationPerHour(cursorTo)

dfMi.to_csv("Milano.csv")
dfMon.to_csv("Montreal.csv")
dfTo.to_csv("Torino.csv")


dfMon = pd.read_csv('Montreal.csv')
dfMi = pd.read_csv('Milano.csv')
dfTo = pd.read_csv('Torino.csv')

dfMon = dfMon.drop(["date"],axis=1)
dfMi = dfMi.drop(["date"],axis=1)
dfTo = dfTo.drop(["date"],axis=1)

print(f"Milano: {len(dfMi)} \n Montreal: {len(dfMon)} \n Torino: {len(dfTo)} \n")

# Done to have precisely 30 days of entryes
dfMon.drop(dfMon[dfMon["month"] == 9].index,inplace=True)
dfMon.drop(dfMon[dfMon["day"] == 31].index,inplace=True)
dfMi.drop(dfMi[dfMi["day"] == 31].index,inplace=True)
dfTo.drop(dfTo[dfTo["day"] == 31].index,inplace=True)
dfMon = dfMon.reset_index(drop=True)
dfMi = dfMi.reset_index(drop=True)
dfTo = dfTo.reset_index(drop=True)
print(f"Milano: {len(dfMi)} \n Montreal: {len(dfMon)} \n Torino: {len(dfTo)} \n")
```

3.2 Point 2

```
#add some randomnesso for better fit in the ARIMA model
dfMi['count'] += np.random.random_sample(len(dfMi))/10
dfMon['count'] += np.random.random_sample(len(dfMon))/10
dfTo['count'] += np.random.random_sample(len(dfTo))/10

def missing_policy(df):
    for i in range(len(df)-1):
        if df.at[i,"hour"] != (df.at[i+1,"hour"]-1):
            if df.at[i,"hour"] == 23:
                if df.at[i+1,"hour"] != 0:
                    new_row = pd.DataFrame({"count":df.at[i,"count"], "year":df.at[i,"year
                    df = pd.concat([df.loc[:i],new_row,df.loc[i+1:]]).reset_index(drop=Tru
            else:
                new_row = pd.DataFrame({"count":df.at[i,"count"], "year":df.at[i,"year"],"
                df = pd.concat([df.loc[:i],new_row,df.loc[i+1:]]).reset_index(drop=True)
    return df

dfMon = missing_policy(dfMon)
dfMi = missing_policy(dfMi)
dfTo = missing_policy(dfTo)
print(f"Milano: {len(dfMi)} \n Montreal: {len(dfMon)} \n Torino: {len(dfTo)} \n")
```

3.3 Point 3

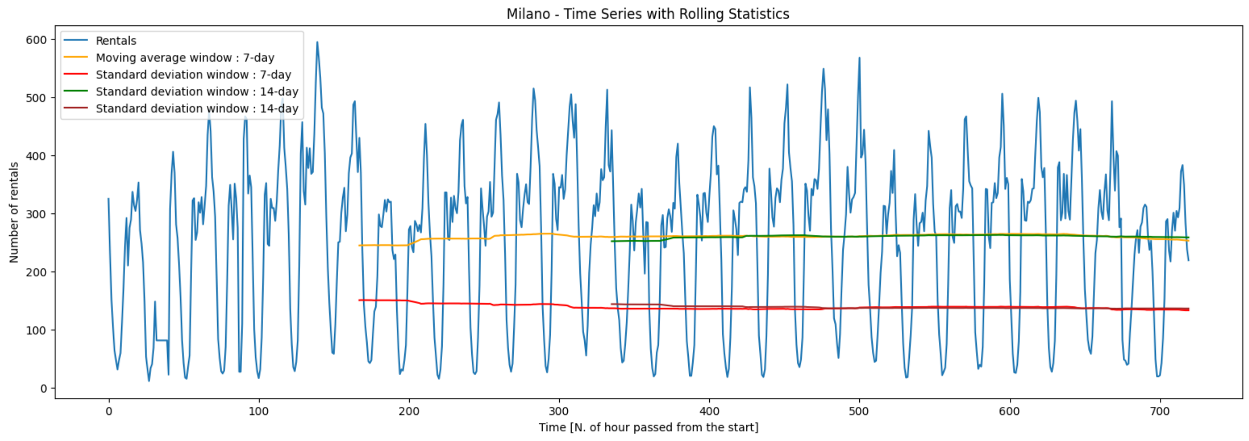


Figure 3.3.1: Rolling statistics of moving average and standard deviation of Milano city

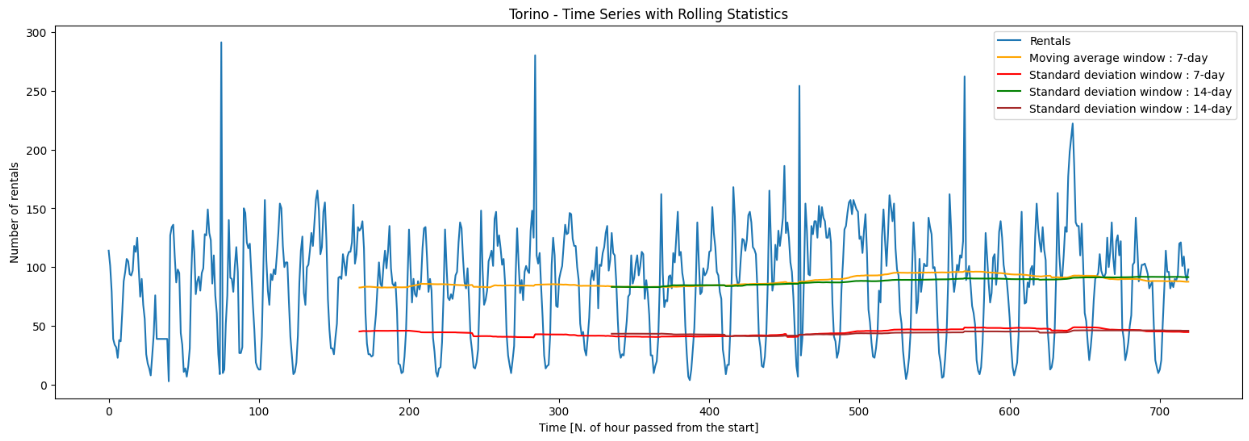


Figure 3.3.2: Rolling statistics of moving average and standard deviation of Torino city

```
city ="Montreal"
if city == "Milano":
    df = dfMi.copy()
elif city == "Torino":
    df = dfTo.copy()
elif city == "Montreal":
    df = dfMon.copy()
```

```
def rolling_mean(df, window):
    return df.rolling(window=window).mean()

def rolling_std(df, window):
    return df.rolling(window=window).std()

week = 24*7
average_7 = rolling_mean(df['count'], week)
std_7 = rolling_std(df['count'], week)
average_14 = rolling_mean(df['count'], 2*week)
std_14 = rolling_std(df['count'], 2*week)

plt.figure(figsize=(19, 6))
plt.plot(df.index, df['count'], label='Rentals')
plt.plot(df.index, average_7, label=f'Moving average window : 7-day', color='orange')
plt.plot(df.index, std_7, label=f'Standard deviation window : 7-day', color='red')
plt.plot(df.index, average_14, label=f'Standard deviation window : 14-day', color='green')
plt.plot(df.index, std_14, label=f'Standard deviation window : 14-day', color='brown')
plt.title(f'{city} - Time Series with Rolling Statistics')
plt.xlabel('Time [N. of hour passed from the start]')
plt.ylabel('Number of rentals')
#plt.grid()
plt.legend()
plt.show()
```

3.4 Point 4

```
# Total ACF
pd.plotting.autocorrelation_plot(df["count"])

# Short ACF
plot_acf(df["count"], lags=48)

# Total PACF
plot_pacf(df["count"])
```

3.5 Point 5

```
# N is the number of past samples used for training
N = week*3
# For the test like prof said is better to assign to the test the last samples so to not c
len_test = week

X = df['count'].values.astype(float)

X_train = X[:N]
X_test = X[N:(N+len_test)]
```

3.6 Point 6

```
p = 2
d = 0
q = 0
history = [x for x in X_train]
model = ARIMA(history, order=(p, d, q))
model_fit = model.fit()
plt.figure(figsize=(10,4))
plt.title(f"{city} - Fitted model vs training data - (p={p}, d={d}, q={q})")
plt.plot(model_fit.fittedvalues , label ="Fitred time series")
plt.plot(history , label="Real time series")
plt.ylabel("N of rentals")
plt.xlabel("Time [Hours from start]")
plt.legend()
plt.show()
model_fit.summary()


# Plot resildual
residuals = pd.DataFrame(model_fit.resid)
sns.set_theme()
sns.kdeplot(residuals ,legend=False)
residuals.plot(grid=True)

print(f"{city} - MSE",mean_squared_error(X_train , model_fit.fittedvalues))
print(f"{city} - MAPE", mean_absolute_error(X_train , model_fit.fittedvalues)/X_train.mean(
print(f"{city} - R2",r2_score(X_train , model_fit.fittedvalues))
```

3.7 Point 7

	Model	MSE	MAPE	R2_score	time
0	"(0, 0, 0)"	2006.88	53.94	0.0	10.71
1	"(0, 0, 1)"	942.35	36.17	0.53	12.14
2	"(0, 0, 2)"	666.74	29.29	0.67	19.95
3	"(0, 0, 3)"	587.93	27.62	0.71	20.75
4	"(0, 0, 4)"	576.6	27.14	0.71	23.76
5	"(1, 0, 0)"	572.13	26.49	0.72	3.09
6	"(1, 0, 1)"	550.22	25.53	0.73	13.47
7	"(1, 0, 2)"	541.45	25.5	0.73	17.1
8	"(1, 0, 3)"	543.65	25.55	0.73	24.84
9	"(1, 0, 4)"	543.43	25.46	0.73	32.82
10	"(2, 0, 0)"	542.09	25.3	0.73	5.58
11	"(2, 0, 1)"	503.71	22.87	0.75	24.33
12	"(2, 0, 2)"	510.45	23.29	0.75	48.85
13	"(2, 0, 3)"	508.64	22.91	0.75	39.38
14	"(2, 0, 4)"	445.79	21.5	0.78	87.05
15	"(3, 0, 0)"	540.68	25.33	0.73	7.9
16	"(3, 0, 1)"	513.42	23.43	0.74	42.22
17	"(3, 0, 2)"	505.14	23.15	0.75	58.74
18	"(3, 0, 3)"	459.5	22.23	0.77	83.93
19	"(3, 0, 4)"	433.67	20.91	0.78	98.91
20	"(4, 0, 0)"	541.45	25.04	0.73	10.34
21	"(4, 0, 1)"	504.49	22.9	0.75	51.35
22	"(4, 0, 2)"	526.87	23.9	0.74	78.88
23	"(4, 0, 3)"	462.0	21.68	0.77	885.48
24	"(4, 0, 4)"	508.75	23.14	0.75	111.22

Table 4: Errors of different model for Montreal

```
p_list = [0, 1,2,3,4]
d = 0
q_list = [0, 1,2,3,4]
predictions = {}
times = {}
sliding_window = True
for p in p_list:
    for q in q_list:
```

	Model	MSE	MAPE	R2_score	time
0	”(0, 0, 0)”	2177.95	40.69	0.0	10.72
1	”(0, 0, 1)”	1148.47	29.22	0.47	11.3
2	”(0, 0, 2)”	946.97	26.03	0.57	14.9
3	”(0, 0, 3)”	865.53	24.85	0.6	21.91
4	”(0, 0, 4)”	838.63	24.05	0.62	32.31
5	”(1, 0, 0)”	825.69	23.59	0.62	9.76
6	”(1, 0, 1)”	821.65	23.45	0.62	14.03
7	”(1, 0, 2)”	821.76	23.51	0.62	22.79
8	”(1, 0, 3)”	817.21	23.38	0.63	32.19
9	”(1, 0, 4)”	816.66	23.42	0.63	40.51
10	”(2, 0, 0)”	820.86	23.44	0.62	11.03
11	”(2, 0, 1)”	810.84	23.13	0.63	25.14
12	”(2, 0, 2)”	808.91	23.17	0.63	57.61
13	”(2, 0, 3)”	739.96	22.42	0.66	57.64
14	”(2, 0, 4)”	788.53	22.98	0.64	103.26
15	”(3, 0, 0)”	820.65	23.51	0.62	16.76
16	”(3, 0, 1)”	779.61	22.59	0.64	56.96
17	”(3, 0, 2)”	806.9	23.2	0.63	87.33
18	”(3, 0, 3)”	726.85	21.58	0.67	110.54
19	”(3, 0, 4)”	789.23	23.39	0.64	172.75
20	”(4, 0, 0)”	810.81	23.35	0.63	25.05
21	”(4, 0, 1)”	768.64	22.18	0.65	70.14
22	”(4, 0, 2)”	765.98	22.17	0.65	117.95

Table 5: Errors of different model for Torino

```
        print((p,d,q))
        pred_model = []
        t_start = time.time()
        history = [x for x in X_train]
        for x in X_test:
            model = ARIMA(history, order=(p, d, q))
            model.initialize_approximate_diffuse()
            model_fit = model.fit()

            pred_model.append(model_fit.forecast()[0])

            history.append(x)
            if sliding_window:
                history=history[1:]  # add for sliding window
        times[(p,d,q)] = time.time() - t_start
        predictions[(p,d,q)] = pred_model

df_err = pd.DataFrame(columns=["Model","MSE", "MAPE", "R2_score", "time"])

for i, model in enumerate(list(predictions.keys())):
    df_err.at[i, "Model"] = model
    df_err.at[i, "MSE"] = round(mean_squared_error(X_test, predictions[model]),2)
    df_err.at[i, "MAPE"] = round(mean_absolute_error(X_test, predictions[model])/X_test.me
    df_err.at[i, "R2_score"] = round(r2_score(X_test, predictions[model]),2)
    df_err.at[i, "time"] = round(times[model], 2)


# Choose learning strategy and N
best_p = 2
d = 0
best_q = 4
predictions = {}
```

```
times = {}
N = (week/2, week, 3/2*week, 2*week,5/2*week,3*week)
sliding_window = [True, False]
for n_train in N:
    train = X[0:int(n_train)]
    for win in sliding_window:
        print(f"{(best_p,d,best_q)} - N = {n_train} - sliding window = {win}")
        pred_model = []
        t_start = time.time()
        history = [x for x in train]
        for x in X_test:
            model = ARIMA(history, order=(p, d, q))
            model.initialize_approximate_diffuse()
            model_fit = model.fit()

            pred_model.append(model_fit.forecast()[0])

            history.append(x)
            if win:
                history=history[1:]  # add for sliding window
        times[(n_train, win)] = time.time() - t_start
        predictions[(n_train, win)] = pred_model

df_err = pd.DataFrame(columns=["Model","N", "Sliding window", "MSE", "MAPE", "R2_score","t

for i, model in enumerate(list(predictions.keys())):
    df_err.at[i, "Model"] = f"({best_p}, 0, {best_q})"
    df_err.at[i, "N"] = model[0]
    df_err.at[i, "Sliding window"] = model[1]
    df_err.at[i, "MSE"] = round(mean_squared_error(X_test, predictions[model]),2)
    df_err.at[i, "MAPE"] = round(mean_absolute_error(X_test, predictions[model])/X_test.me
    df_err.at[i, "R2_score"] = round(r2_score(X_test, predictions[model]),2)
    df_err.at[i, "time"] = round(times[model], 2)

plt.figure()
plt.plot(df_err[df_err["Sliding window"] == True]["N"], df_err[df_err["Sliding window"] ==
plt.plot(df_err[df_err["Sliding window"] == False]["N"], df_err[df_err["Sliding window"] =
plt.xlabel("N")
plt.ylabel("MAPE [%]")
plt.title(f"{city} - ARIMA({best_p}, 0, {best_q}) varying N and validation")
plt.legend()
plt.savefig(f"results/{city}varN.png")
```

3.8 Point 8

```
p = 2
d = 0
q = 4

X_train = X[:3*week]
X_test = X[3*week: 4*week]

predictions = {}
times = {}
h_values = list(range(1, 24, 2))
for h in h_values:
    print(f'Horizon: {h} h.')
    pred_model = []
    t_start = time.time()
    history = [x for x in X_train]
    for test in X_test:
        # Fit the model
        model = ARIMA(history, order=(p, d, q))
        model_fit = model.fit()
        output = model_fit.forecast(steps=h)[-1]
        pred_model.append(output)
        history.append(test)
    elapsed_time = time.time() - t_start
    print(f'Elapsed time: {elapsed_time} s.')
    times[h] = elapsed_time
    predictions[h] = pred_model

# Create the dataframe with the metrics
df_err_h = pd.DataFrame(columns=["Horizon", "MSE", "RMSE", "MAPE", "R2_score", "time"])
for idx, h in enumerate(h_values):
    df_err_h.at[idx, "Horizon"] = h
    df_err_h.at[idx, "MSE"] = round(mean_squared_error(X_test, predictions[h]), 2)
    df_err_h.at[idx, "RMSE"] = round(np.sqrt(mean_squared_error(X_test, predictions[h])),
    df_err_h.at[idx, "MAPE"] = round(mean_absolute_error(X_test, predictions[h])/X_test.me
    df_err_h.at[idx, "R2_score"] = round(r2_score(X_test, predictions[h]), 2)
    df_err_h.at[idx, "time"] = round(times[h], 2)

# Plot the results
plt.figure()
plt.plot(h_values, df_err_h.MAPE)
plt.xlabel('Time horizon [h]')
plt.ylabel('MAPE')
plt.title(city)

# Create a dataframe consisting of the MAPE values at each value of h for each value of p
data = {
    "Horizon": list(range(1, 24, 2)),
    "p = 0": [18.02, 34.16, 44.02, 44.02, 44.02, 44.02, 44.02, 44.02, 44.02, 44.02, 44.02,
    "p = 1": [16.62, 22.18, 30.37, 36.55, 40.03, 41.91, 42.91, 43.44, 43.72, 43.86, 43.94,
    "p = 2": [15.52, 28.17, 43.58, 55.47, 57.46, 50.52, 41.57, 37.88, 40.21, 44.54, 47.07,
    "p = 3": [15.5, 27.34, 42.46, 54.59, 56.34, 49.11, 40.53, 38.07, 41.42, 46.17, 47.92,
    "p = 4": [15.71, 26.99, 41.61, 53.37, 53.93, 46.6, 38.76, 37.8, 43.45, 49.03, 49.24, 4
    }

df = pd.DataFrame(data)

# Plot the comparison of the different models
colors = []
for column in df.columns[1:]:
    line, = plt.plot(df.Horizon, df[column], marker='o', label=column)
    colors.append(line.get_color())

plt.title("MAPE at each h-value according to p-value")
plt.xticks(df.Horizon.to_list())
plt.xlabel("h")
plt.ylabel("MAPE")
plt.legend(loc='lower right', ncol=1)

# Create a dictionary to keep track of how many times a model has the lowest MAPE
hor_win = {h: 0 for h in df.Horizon}
for index, row in df.iterrows():
    min_col = row[1:].idxmin()
    hor_win[df.Horizon.iloc[index]] = int(min_col[-1])

# Plot the chart that evaluates the best model
```

```
horizon_values = list(hor_win.keys())
for column in range(0, max(hor_win.values()) + 1):
    x_values = [horizon for horizon, col in hor_win.items() if col == column]
    y_values = [column] * len(x_values)
    plt.plot(x_values, y_values, marker='o', label=f'p = {column}', color=colors[column])

plt.yticks(range(0, max(hor_win.values()) + 1))
plt.xticks(horizon_values)
plt.title("Value of p with lowest MAPE according to h")
plt.xlabel("h")
plt.ylabel("p")
plt.legend()
plt.grid(linestyle='--', linewidth=0.5)
plt.show()
```