POLITECNICO DI TORINO

ICT FOR SMART MOBILITY

Report of Lab2

Group 8

Pawel Pluciennik **S321347**
Maryam Bigonah **S308977**
Atefeh Khalili Param **S309328**

January 2024

Step 3– Lab#2 - Prediction using ARIMA models

Consider the time series of rentals that you obtained in the previous steps, for each city. Consider a time period of 30 days for which you have data (pay attention to stationarity – are December or January good months?)

The goal of this lab is to experiment with predictions using ARIMA models and to check how the error changes with respect to hyperparameters. For this, you must consider the various parameters in the ARIMA model training, including

- the model parameters (p,d,q)
- the training windows size N (how many past samples are used for training)
- the training policy, i.e., expanding versus sliding windows.

A possible outline of the work could be

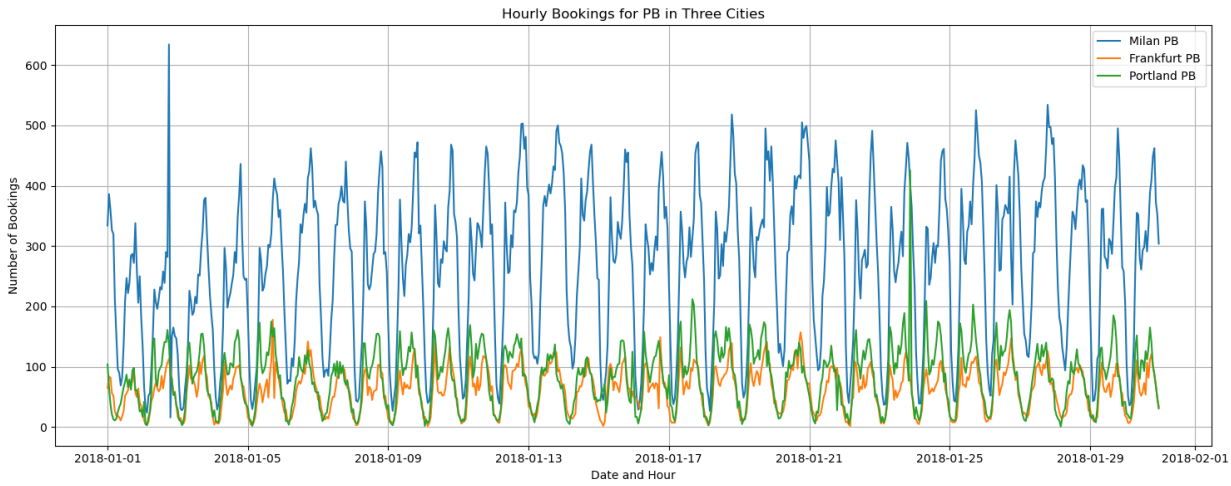
1. For each city, consider the selected period of 30 days. Extract the number of rentals recorded at each hour – after properly filtering the outliers and those bookings that are not rentals.

We chose the period from January 1 to January 31, 2018. We believe that December or January are not the best months to check stationary due to the Christmas holidays, weather conditions, or people's general trips to other cities. For these months, demand for car rentals may be lower than in other months, while they are still important data for training the ARIMA model.

why then choosing January (?)

We have applied two filters. One of them is considering the time which is more than 5 min(300s) and less than 5 hours (18000s). We did so because we believe that trips lasting less than 5 min are considered a mistake by the person renting the car. Longer trips than 5 h should not happen either and if they do exist, they are an exception. It is not worthwhile to rent cars for such a long time in this system. Other companies rent cars for longer periods. The second filtering is to omit the bookings that have similar starting and final addresses. Trips are usually made to a designated destination and do not return to the same place during a single car rental.

better use position than address

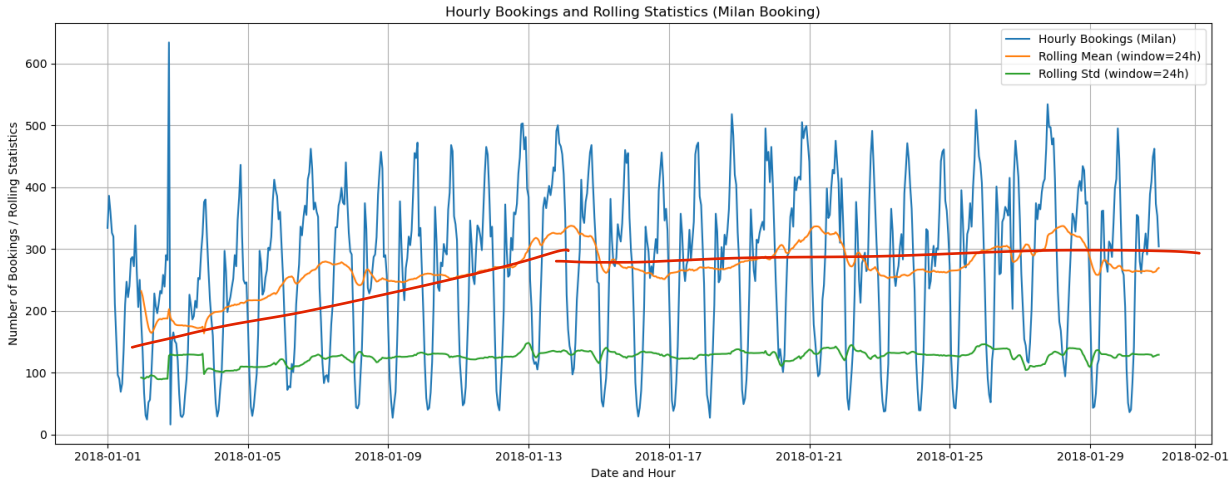


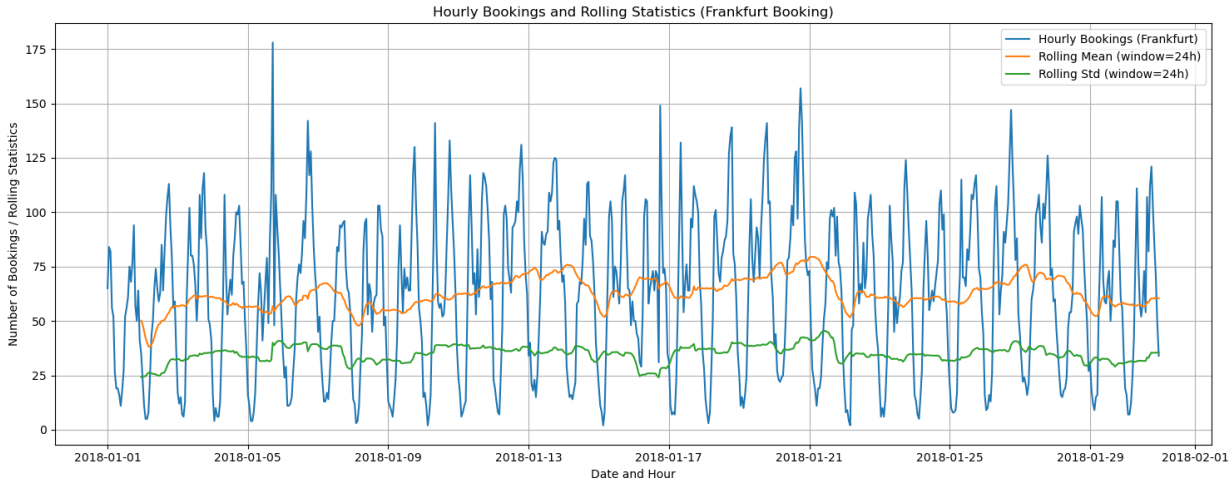
2. Check if there are missing samples (recall – ARIMA models assume a regular time series, with no missing data). In case of missing samples – define a policy for fitting missing data. For instance, use i) the last value,
ii) the average value, or iii) replace with zeros, or iv) replace with average value for the given time bin, etc

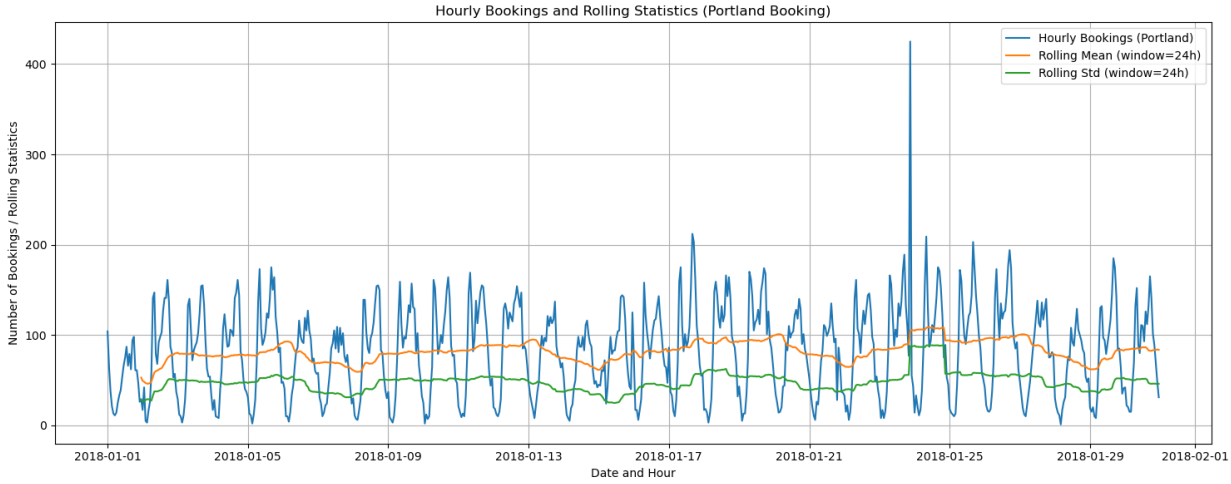
In the Arima model, we should consider a regular time series that has no missing samples. In our case, we split the month's data by hour on each day, which means that for the Army model, we must have continuity of each hour on each day. In the case of missing data, we would have to add it through one of the ways described in the task. After checking, we are not missing any samples, so we can proceed to the next step.

3. Check if the time series is stationary or not. Decide accordingly whether to use differencing or not (d=0 or not).

In this step, we should check whether the time series are stationary or not. Stationarity means that the statistical parameters of a time series, such as mean and variance, do not change over time. We plot the rolling mean and standard deviation of the time series for 3 cities in 30 days. As it can be seen from the diagram in Portland there is a small fluctuation of rolling mean and standard deviation for the whole period. Thus the hyperparameter d can be considered as 0. However for the city of Milan from 1st of January till 13th we can observe that the rolling mean has an increasing trend so that the time series can not be considered as stationary. In this situation we can use a different d and d is not 0.


Hourly Bookings and Rolling Statistics (Milan Booking)
Hourly Bookings (Milan)
Rolling Mean (window=24h)
Rolling Std (window=24h)
Number of Bookings / Rolling Statistics
Date and Hour


Hourly Bookings and Rolling Statistics (Frankfurt Booking)
Hourly Bookings (Frankfurt)
Rolling Mean (window=24h)
Rolling Std (window=24h)
Number of Bookings / Rolling Statistics
Date and Hour


Hourly Bookings and Rolling Statistics (Portland Booking)
Hourly Bookings (Portland)
Rolling Mean (window=24h)
Rolling Std (window=24h)
Number of Bookings / Rolling Statistics
Date and Hour

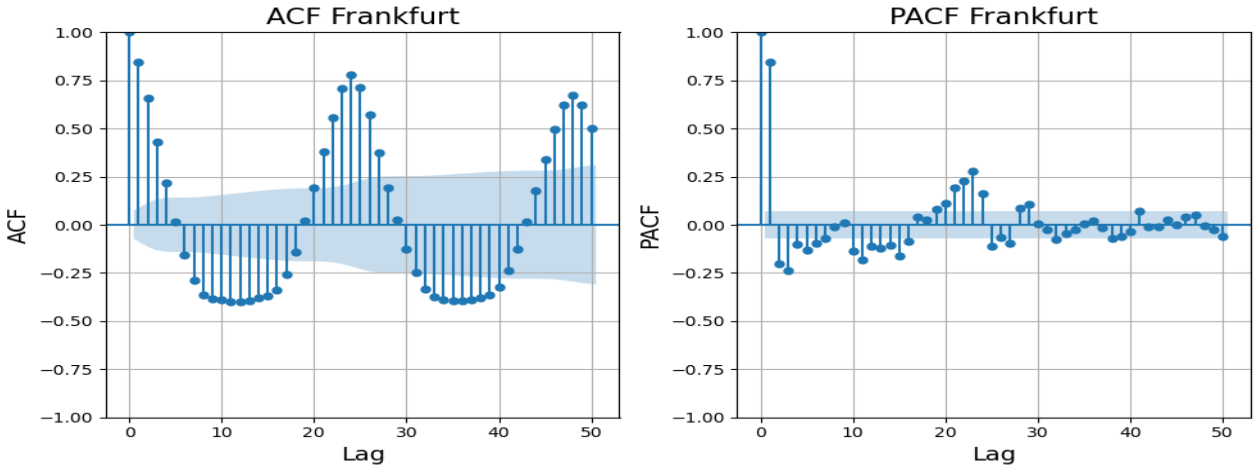
4. Compute the ACF and PACF to observe how they vanish. This is instrumental in guessing possible good values of the p and q parameters, assuming the model is pure AR or pure MA. What if your model is not pure AR or pure MA, i.e., it is an ARMA process?

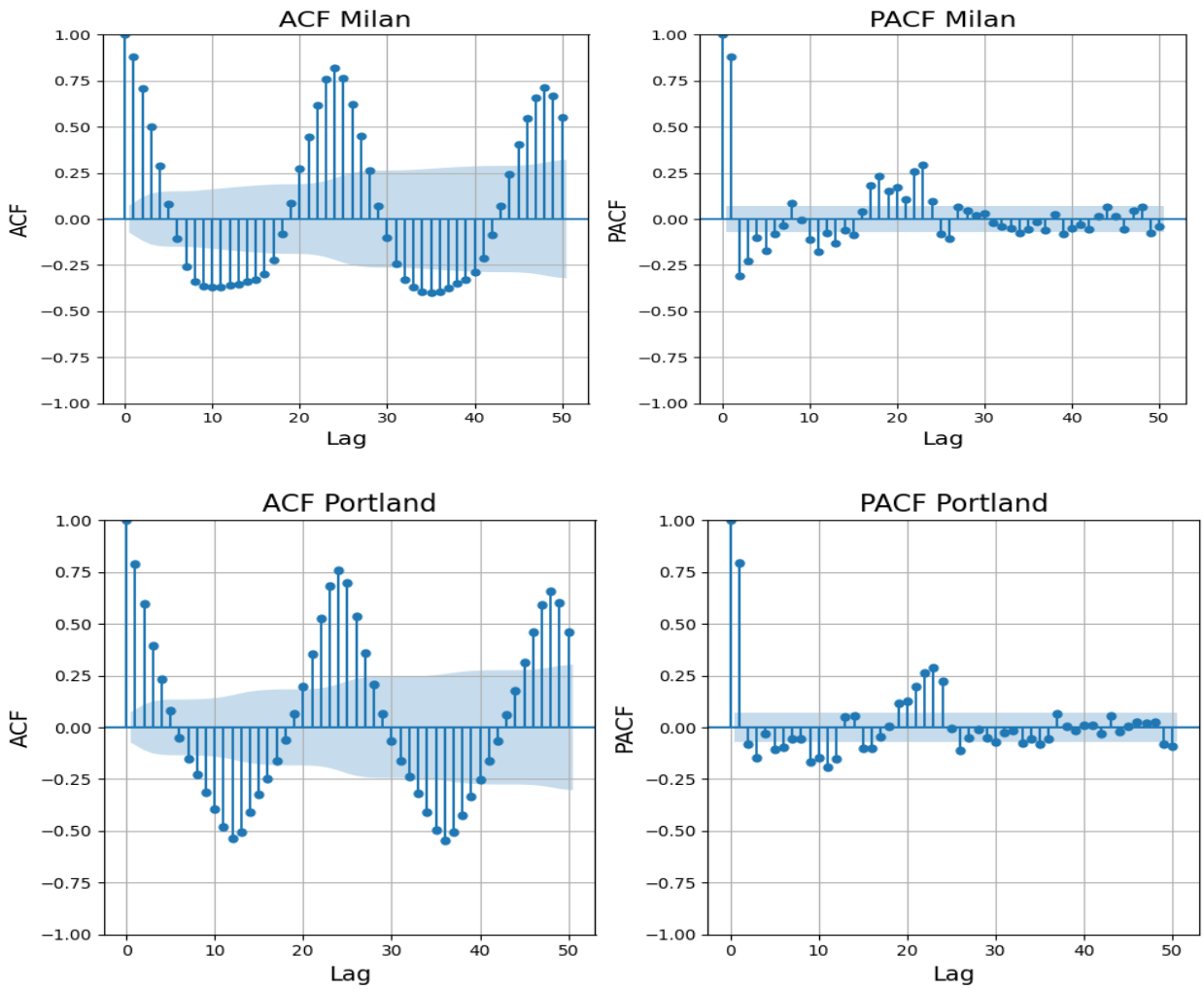
The autocorrelation function (ACF) and partial autocorrelation function (PACF) are tools used in time series analysis.

The ACF plot shows the correlation between a time series and its lagged values. When ACF examines a time series, it acts like a detective trying to understand whether past values have an impact on current ones. If ACF "senses" connections, it means that previous observations impact current ones.

PACF is a "detective tracing a direct line of suspicion." When ACF points out certain connections, PACF indicates which earlier time has the greatest influence on the present. This can be compared to a detective-style investigation, where PACF helps focus on a key suspect.

We plotted the ACF and PACF diagrams for the time series of each city to have an initial guess of hyperparameters p (lag order) and q (order of moving average model). If our model is not purely AR or MA we observe a combination of autocorrelation and partial autocorrelation patterns in ACF and PACF. If we want to guess possible values for p and q, we observe that there are 2 spikes at lags 1 and 2 in the diagram of PACF, and the following spikes in the other lags decay. Thus the lag order (p) can be considered as 2. It is worth mentioning that the shaded area in the diagram of ACF and PACF is the confidence interval. In the ACF diagrams, we can observe a repeating pattern in the significant peaks so it is a bit complicated to guess the q value from the lags out of the confidence interval. Here the periodicity is 2 so we suggest the value of the moving average is equal to 2.





5. Decide the number of past samples N to use for training, and how many to use for testing. Given you have 30 days of data (each with 24 hours) you can consider for instance training during the first week of data and testing the prediction in the second week of data.

We should divide our dataset into training and testing sets. It is important to choose the training and testing sets to avoid overfitting. In the training phase, we find a model according to the relation between the input data and output. Then we implement our model on the test samples to evaluate the accuracy of our model. We examined different values of training samples and found that dedicating approximately 60% of all samples to train our model can result in lower error. We also dedicated 3 testing days (72 hours) to test our ARIMA model. The comparison between different values of training days for given (p=2, d=0,q=1) is reported in the following table.

why using these p, q (?)

does it make sense to use % of set size here? time has daily and weekly periodicity...

cities	MAPE (training days 58% of samples)	MAPE (training days= 60% of samples)	MAPE (training days= 62% of samples)	MAPE (training days= 65% of samples)	MAPE (training days= 67% of samples)
Milan	13.53%	12.51%	12.53%	13.15%	13.75%
Portland	19.16%	18.8%	20.85%	19.72%	21.01%
Frankfurt	18.82%	17.89%	16.61%	18.27%	19.23%

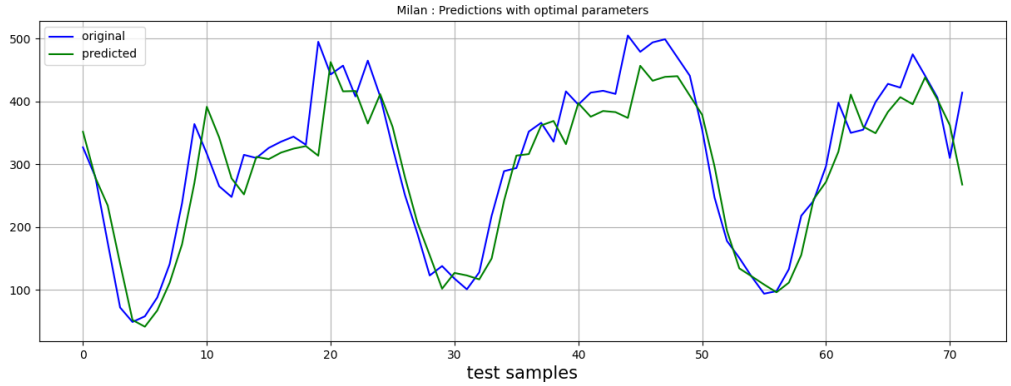
6. Given N, (p,d,q), train a model, and compute the error. Consider the MPE and/or MAPE and/or other metrics – so that you can compare results for different cities (use percentage errors and not absolute errors for this comparison. Absolute errors would obviously be not directly comparable).

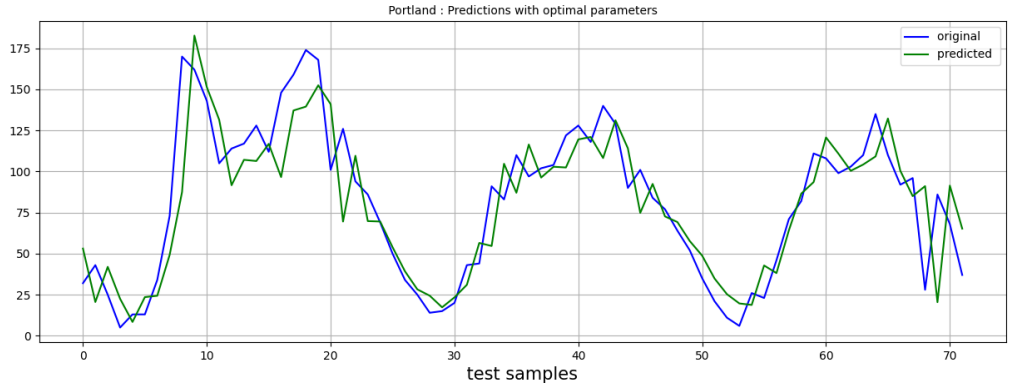
why using p=2 q=2 here

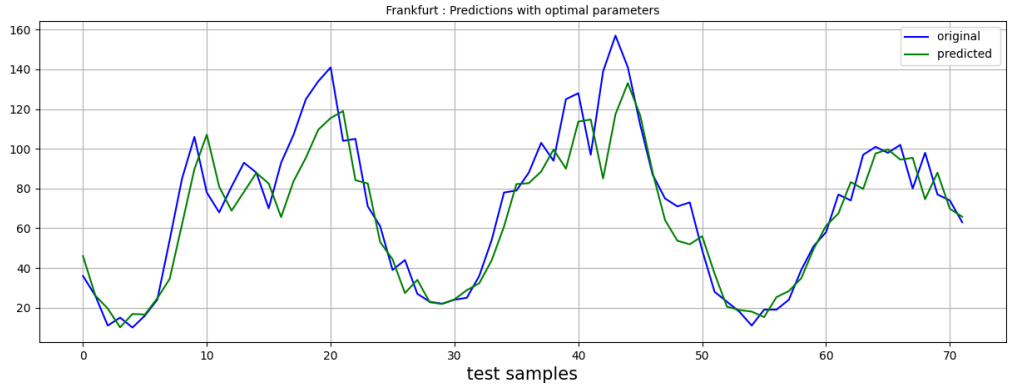
After choosing the number of our training samples we should choose appropriate hyperparameters for our model. Since in the PACF diagram, the number of lags before tailing off is 2, we chose p. Moreover, the value of q is regarded as 2 according to the periodicity of the ACF diagram. We should note that there are no unique values for p,d,q and in the next step we will examine the effect of changing these parameters on our results. In this step, the value of d is considered as 0 since we used no differencing due to the approximate smooth trend of the diagram of the rolling mean.

why changing from p=2 q=1 ?

Cities	**MAPE**	**MSE**	**R2**
Milan	**12.43%**	**2666.19**	**0.84**
Portland	**21.77%**	**560.38**	**0.73**
Frankfurt	**16.51%**	**239.05**	**0.83**





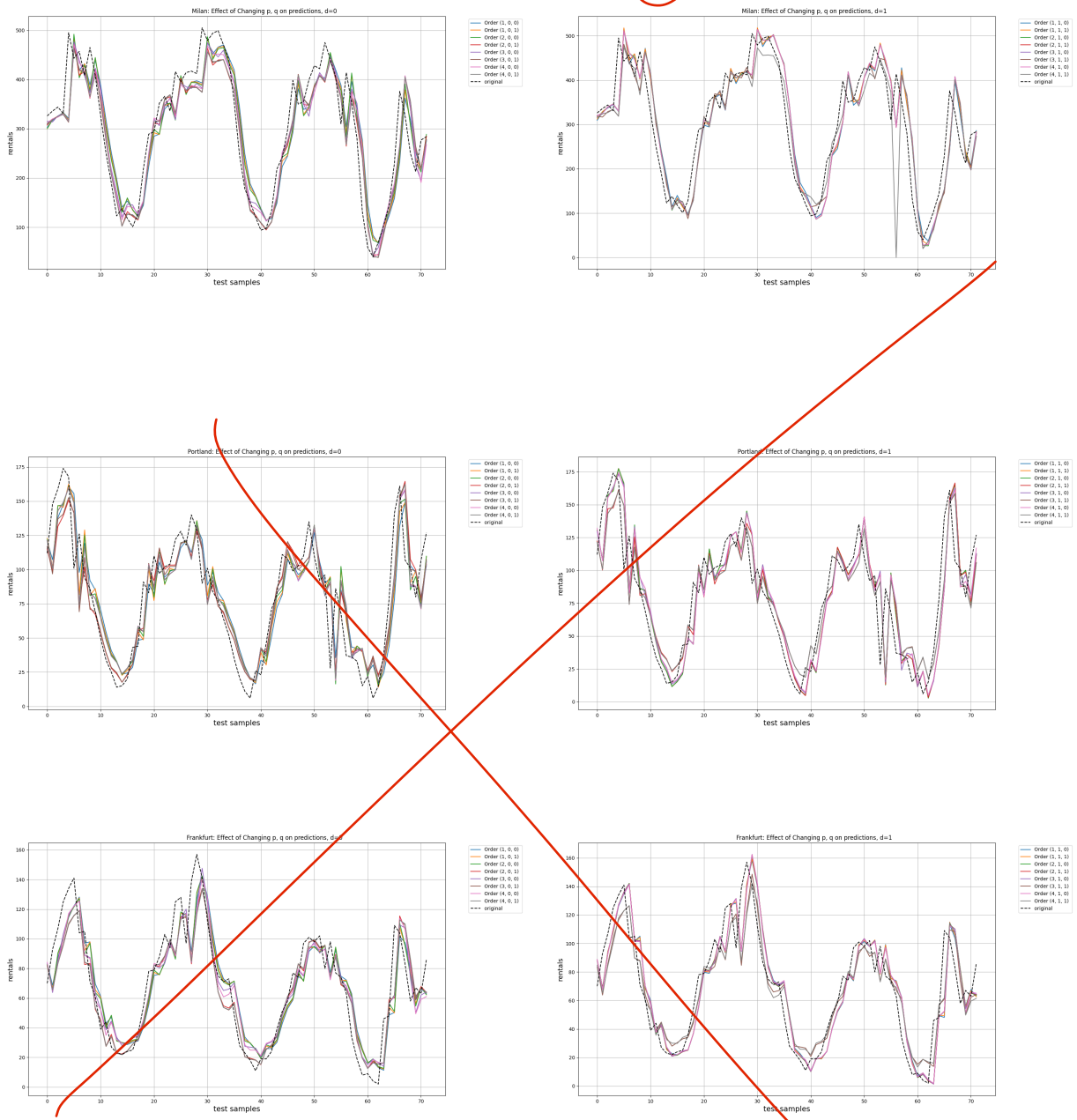


7. Now check the impact of parameters:

a. Keep N fixed, and do a grid search varying (p,d,q) and observe how the error changes. Choose the best (p,d,q) parameter tuple for each city. Justify your choice.

In this step, we want to evaluate the effect of changing different parameters of p,d, q on the ARIMA model. We examined the values of p in the range of [1,2,3,4] and q in the range of [0,1] For 3 cities the effect of changing these values on the diagram of predictions are implemented. First, we selected d as 0 then we changed the value of d to 1 and plotted the effect of it on

predictions. The reason that we plotted the changing of d to 1 in a different figure was to have a clear and transparent view of the diagrams.



no need to show these plots.
what are these supposed to tell!

In the following tables, we obtained MAPE, MSE, and R2 according to different values of p and q. The value of d is considered to be 0 in these tables.

Milan

	p	q	d	MAPE	MSE	R2
0	1.0	0.0	0.0	15.149216	3457.709343	0.794603
1	1.0	1.0	0.0	14.745509	3286.846813	0.804753
2	2.0	0.0	0.0	14.464674	3189.455886	0.810538
3	2.0	1.0	0.0	13.023853	2887.919794	0.828450
4	3.0	0.0	0.0	13.264317	2980.655868	0.822941
5	3.0	1.0	0.0	12.937042	2892.314634	0.828189
6	4.0	0.0	0.0	13.264085	2981.804735	0.822873
7	4.0	1.0	0.0	12.897231	2892.069725	0.828203
8	5.0	0.0	0.0	12.954574	2838.016410	0.831414
9	5.0	1.0	0.0	13.020075	2909.088051	0.827193

Frankfurt

	p	q	d	MAPE	MSE	R2
0	1.0	0.0	0.0	20.952096	331.104783	0.778179
1	1.0	1.0	0.0	20.701393	326.176051	0.781481
2	2.0	0.0	0.0	20.522113	323.085151	0.783551
3	2.0	1.0	0.0	18.864957	292.363021	0.804133
4	3.0	0.0	0.0	19.875943	301.974433	0.797694
5	3.0	1.0	0.0	18.642957	290.374525	0.805466
6	4.0	0.0	0.0	20.107205	300.608682	0.798609
7	4.0	1.0	0.0	18.653516	290.453205	0.805413
8	5.0	0.0	0.0	19.791120	299.599906	0.799285
9	5.0	1.0	0.0	18.570973	289.762410	0.805876

are all the digit significant ? up

portland

	p	q	d	MAPE	MSE	R2
0	1.0	0.0	0.0	23.427429	544.976187	0.721903
1	1.0	1.0	0.0	24.122615	584.566618	0.701701
2	2.0	0.0	0.0	23.949231	587.901002	0.699999
3	2.0	1.0	0.0	20.830416	495.345382	0.747230
4	3.0	0.0	0.0	23.200013	556.556186	0.715994
5	3.0	1.0	0.0	21.473066	525.362141	0.731912

In the next step, we want to observe the effect of d on the measurement errors. The reason why we displayed these results in different tables was due to the running time. Evaluation of the large dimension of all of these parameters while running the code takes much time. ?

Milan

repetition ?

	p	q	d	MAPE	MSE	R2
0	3.0	1.0	0.0	12.937042	2892.314634	0.828189
1	3.0	2.0	0.0	12.672383	2785.321847	0.834545
2	3.0	3.0	0.0	12.713716	2746.471016	0.836852
3	3.0	1.0	1.0	14.743843	3347.056978	0.801176
4	3.0	2.0	1.0	15.062413	3305.705540	0.803632
5	3.0	3.0	1.0	13.864538	2877.573139	0.829065

why changing d ?

why you change q only for p = 3 ?

The table for Milan shows that the optimum values for this city are (p=3, d=0, q=2)

Frankfurt

	p	q	d	MAPE	MSE	R2
0	3.0	1.0	0.0	18.642957	290.374525	0.805466
1	3.0	2.0	0.0	18.610167	290.312389	0.805507
2	3.0	3.0	0.0	18.586202	290.156019	0.805612
3	3.0	1.0	1.0	19.979724	302.389328	0.797416
4	3.0	2.0	1.0	20.415184	313.510096	0.789966
5	3.0	3.0	1.0	21.294976	358.075157	0.760110

The table for Frankfurt shows that the optimum values for this city are (p=3, d=0, q=3)

Portland

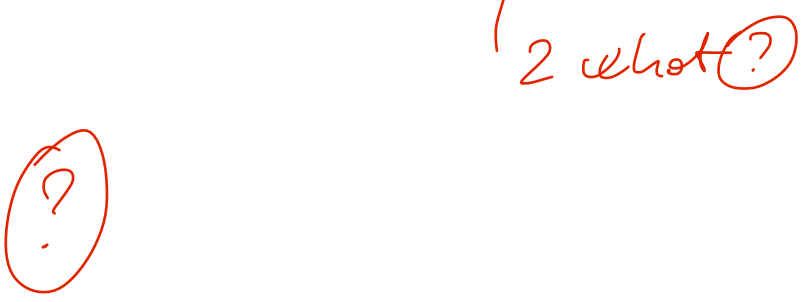
	p	q	d	MAPE	MSE	R2
0	3.0	1.0	0.0	21.473066	525.362141	0.731912
1	3.0	2.0	0.0	21.454237	525.870924	0.731653
2	3.0	3.0	0.0	20.261981	481.322990	0.754385
3	3.0	1.0	1.0	23.204561	557.419031	0.715554
4	3.0	2.0	1.0	22.611530	627.097441	0.679998
5	3.0	3.0	1.0	20.809394	474.178887	0.758031

The table for Portland shows that the optimum values for this city are (p=3, d=0, q=3)

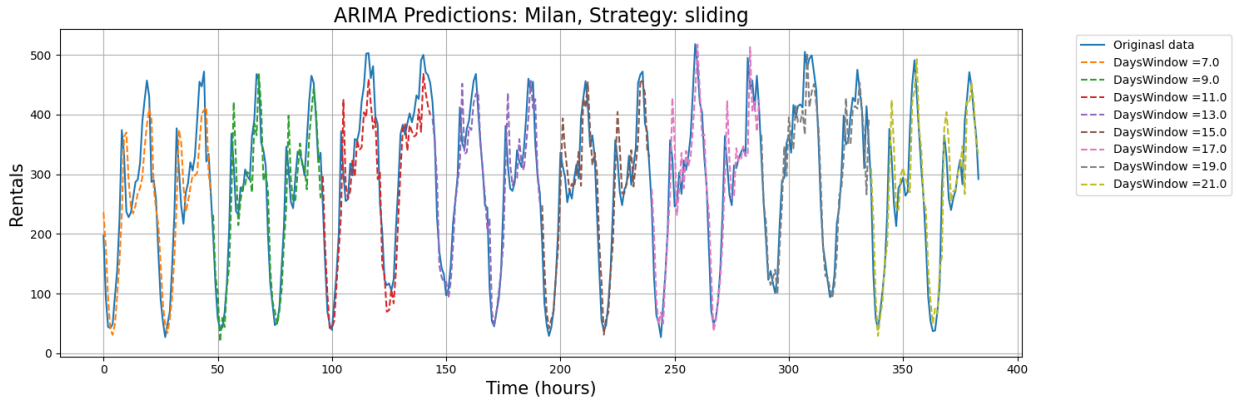
For simplicity, we select the values of (p=3, d=0, q=3) for all cities in the following steps.

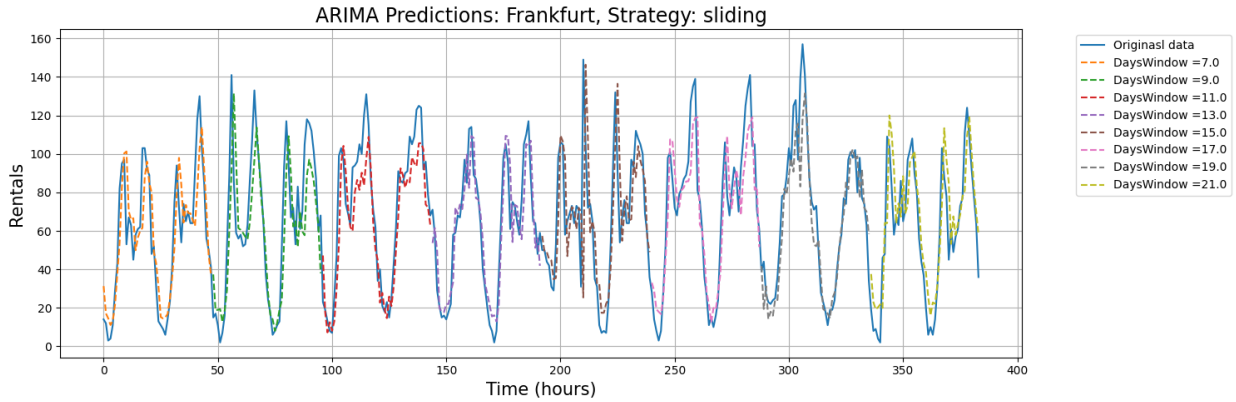
b. Given the best parameter configuration, change N and the learning strategy (expanding versus sliding window). Always keep the testing set on the same portion of the data. For instance, if you use 1 week for testing, you can use from 1 day to 3 weeks for training.

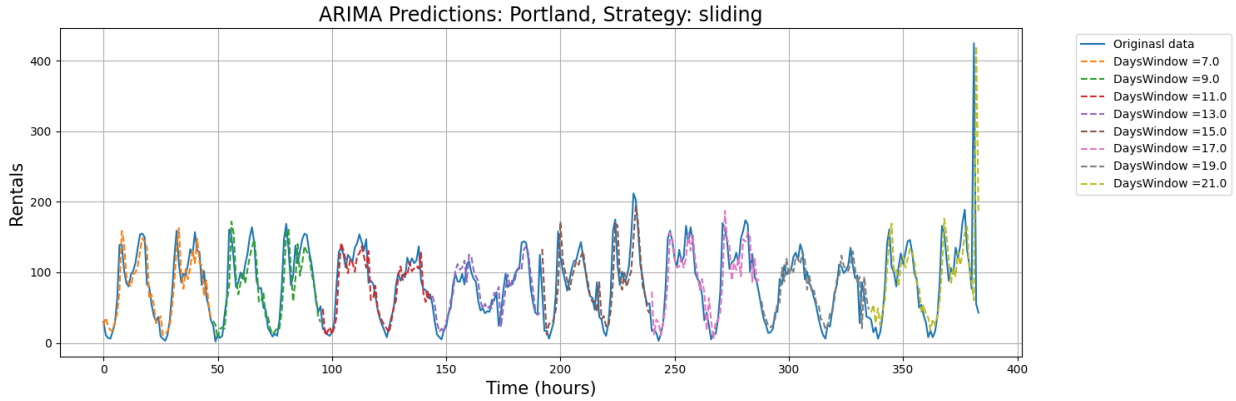
In this step, we want to visualize the effect of sliding and expanding windows on our model. We start with the 7 days of the training window and evaluate the results as we have a sliding or expanding strategy. The test size is 2 in this step and the parameters of p, d, and q are considered the obtained optimum values in the previous step. (3,0,3)

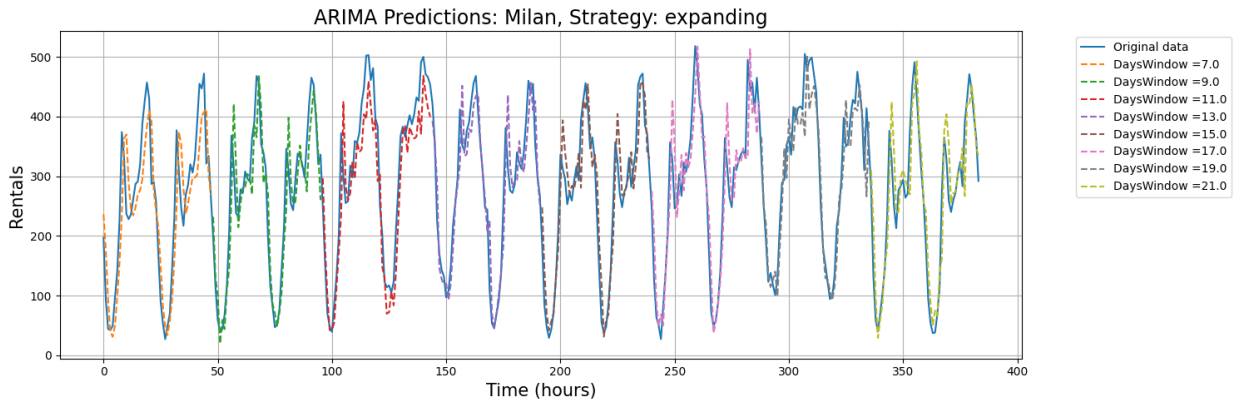


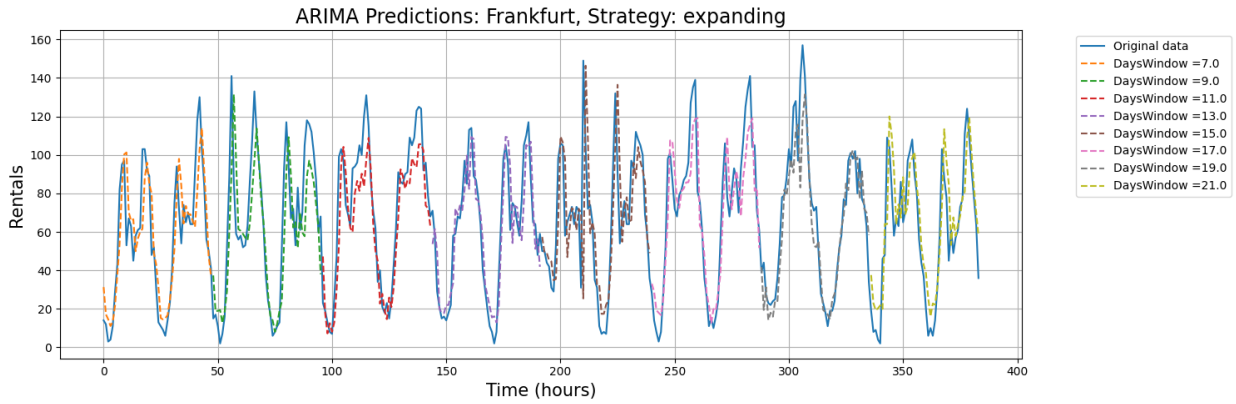
what does this figure show (?)

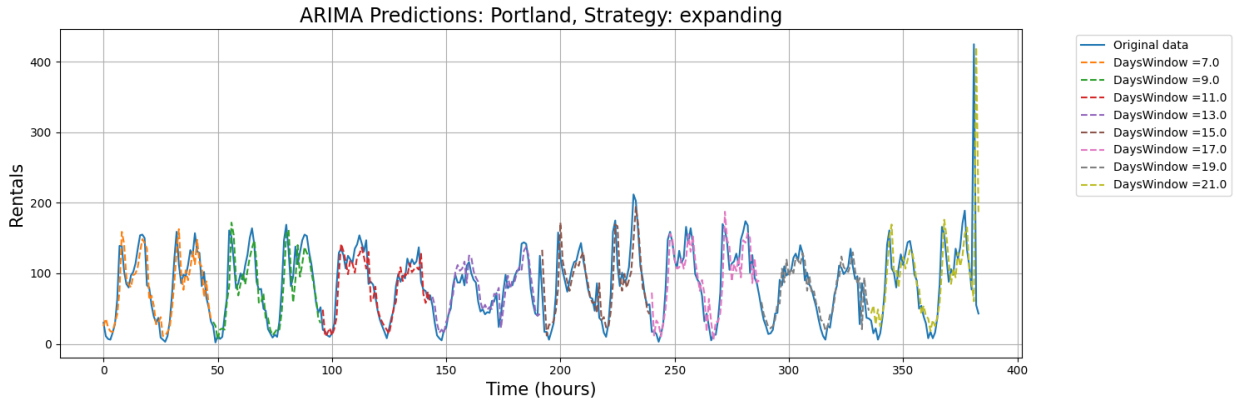












c. Compare results for the different cities. How does the relative error change w.r.t. the absolute number of rentals?

Milan:sliding

	N[hours]	N[days]	MAPE	MSE	R2
0	168	7	16.294444	3130.803356	0.814875
1	336	14	13.035034	2326.837778	0.861362
2	504	21	12.987714	2573.068757	0.844382

Milan:expanding

	N[hours]	N[days]	MAPE	MSE	R2
0	168	7	16.294444	3130.803356	0.814875
1	336	14	13.035034	2326.837778	0.861362
2	504	21	12.987714	2573.068757	0.844382

changes are significant !

Frankfurt:Sliding

	N[hours]	N[days]	MAPE	MSE	R2
0	168	7	20.325567	305.120532	0.761334
1	336	14	20.065718	369.238913	0.717506
2	504	21	19.054880	241.133874	0.797754

Frankfurt:expanding

	N[hours]	N[days]	MAPE	MSE	R2
0	168	7	20.325567	305.120532	0.761334
1	336	14	20.065718	369.238913	0.717506
2	504	21	19.054880	241.133874	0.797754

Portland:Sliding

	N[hours]	N[days]	MAPE	MSE	R2
0	168	7	20.893236	465.283446	0.789677
1	336	14	21.849366	659.960670	0.701210
2	504	21	29.428819	2437.062652	0.260473

Portland:expanding

	N[hours]	N[days]	MAPE	MSE	R2
0	168	7	20.893236	465.283446	0.789677
1	336	14	21.849366	659.960670	0.701210
2	504	21	29.428819	2437.062652	0.260473

In this step, we evaluated the error according to different values of N for both strategies. From the results, we can note that there is no significant difference in the error between the two strategies. Moreover, the error for Milan is lower than the error for the other 2 cities.

why?

- Very confused and with a lot of useless choices
- missing baseline comparison
- no impact of h

3--/5

Appendixes

1.

```
import pymongo as pm
import pprint
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from datetime import datetime
start_date = datetime(2017, 12, 31)
end_date = datetime(2018, 1, 31)
def query(city):
    return {
        "city": city,
        "init_date": {"$gte": start_date, "$lte": end_date}
        }

def dataMongoDB(collection, temp_query):
    return pd.DataFrame(list(collection.find(temp_query)))

#def filter_query(city):
    return {
        "city": city,
        "init_date": {"$gte": start_date, "$lte": end_date},
        "$and": [
            {
                "$expr": {
                    "$gt": [
                        {"$subtract": ["$final_time", "$init_time"]},
                        300
                    ]
                }
            },  # <-- Add a comma here
            {
                "$expr": {
                    "$lt": [
                        {"$subtract": ["$final_time", "$init_time"]},
                        18000  # less than 5 hours (18000 sec)
                    ]
                }
            }
```

```
        ]
    }

def filter_query(city):
    return {
        "city": city,
        "init_date": {"$gte": start_date, "$lte": end_date},
        "$and": [
            {
                "$expr": {
                    "$gt": [
                        {"$subtract": ["$final_time", "$init_time"]},
                        300
                    ]
                }
            },
            {
                "$expr": {
                    "$lt": [
                        {"$subtract": ["$final_time", "$init_time"]},
                        18000  # less than 5 hours (18000 sec)
                    ]
                }
            },
            {
                "initial_address": {"$ne": "$final_address"}  # Add this
condition
            }
        ]
    }



def calculate_and_plot_cdf(data, label):
    sorted_data = np.sort(data)
    cdf = np.arange(len(sorted_data)) / float(len(sorted_data))
    plt.plot(sorted_data, cdf, label=label)

def aggregate_city_data(df):
    df['date_hour'] = df['init_date'].dt.floor('H')
```

```
    return df.groupby('date_hour').size()

client = pm.MongoClient('bigdatadb.polito.it',
                        ssl=True,
                        authSource = 'carsharing',
                        username = 'ictts',
                        password ='Ict4SM22!',
                        tlsAllowInvalidCertificates=True)

db = client['carsharing']
PB_enjoy_collection = db['enjoy_PermanentBookings']
PB_collection = db['PermanentBookings']
PP_enjoy_collection = db['enjoy_PermanentParkings']
PP_collection = db['PermanentParkings']

m = "Milano"
f = "Frankfurt"
p = "Portland"
df_milan_PB = dataMongoDB(PB_enjoy_collection, filter_query(m))
df_portland_PB = dataMongoDB(PB_collection, filter_query(p))
df_frankfurt_PB = dataMongoDB(PB_collection, filter_query(f))

df_milan_PB['duration'] = (df_milan_PB['final_time']
-df_milan_PB['init_time']) / 60
df_frankfurt_PB['duration'] = (df_frankfurt_PB['final_time']
-df_frankfurt_PB['init_time']) / 60
df_portland_PB['duration'] = (df_portland_PB['final_time']
-df_portland_PB['init_time']) / 60


MilanDataPB = df_milan_PB['duration'].values
FrankfurtDataPB = df_frankfurt_PB['duration'].values
PortlandDataPB = df_portland_PB['duration'].values

hourly_counts_milan_PB = aggregate_city_data(df_milan_PB)
hourly_counts_frankfurt_PB = aggregate_city_data(df_frankfurt_PB)
hourly_counts_portland_PB = aggregate_city_data(df_portland_PB)

plt.figure(figsize=(15, 6))
```

```
plt.plot(hourly_counts_milan_PB.index,
hourly_counts_milan_PB.values,label='Milan PB')
plt.plot(hourly_counts_frankfurt_PB.index,hourly_counts_frankfurt_PB.value
s, label='Frankfurt PB')
plt.plot(hourly_counts_portland_PB.index,hourly_counts_portland_PB.values,
label='Portland PB')
plt.xlabel('Date and Hour')
plt.ylabel('Number of Bookings')
plt.title('Hourly Bookings for PB in Three Cities')
plt.legend()
plt.tight_layout()
plt.grid(True)
plt.show()



#calculate_and_plot_cdf(MilanDataPB, 'MilanPermanentBookings')
#calculate_and_plot_cdf(FrankfurtDataPB, 'FrankfurtPermanentBookings')
#calculate_and_plot_cdf(PortlandDataPB, 'PortlandPermanentBookings')

#plt.xlabel('Time duration')
#plt.xscale('log')
#plt.ylabel('CDF')
#plt.title('Three cities CDF/Duration after filter')
#plt.legend()
#plt.grid(True)
#plt.show()









#ii=df_milan_PB.index[ df_milan_PB [" init_address "] == df_milan_PB
["final_address " ]]. tolist ()
#df_milan_PB_secondfilter = df_milan_PB . drop (ii)
```

2.

```
import pandas as pd


def aggregate_city_data(df):
    df['init_date'] = pd.to_datetime(df['init_date'])
    df['date_hour'] = df['init_date'].dt.floor('H')
    result = df.groupby('date_hour').size()
    return result


def check_continuous_data(df, city_name):
    agg_data = aggregate_city_data(df)
    #  missing hours is for searching lost hours during 1 month. It checks
which indexes are not present in this period.
    missing_hours = pd.date_range(start=agg_data.index.min(),
end=agg_data.index.max(), freq='H').difference(
        agg_data.index)
    #  if there are some missing hours variable missing hours won't be
empty
    if len(missing_hours) == 0:
        print(f"Data for {city_name} is continuous for all hours.")
    else:
        print(f"Missing data for {city_name} at hours: {missing_hours}")


df_milan_filtered_PB = pd.read_csv('milan_filtered.csv')
df_portland_filtered_PB = pd.read_csv('portland_filtered.csv')
df_frankfurt_filtered_PB = pd.read_csv('frankfurt_filtered.csv')

check_continuous_data(df_milan_filtered_PB, 'Milan')
check_continuous_data(df_portland_filtered_PB, 'Portland')
check_continuous_data(df_frankfurt_filtered_PB, 'Frankfurt')
```

3.

```
import pymongo as pm
import pprint
```

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from datetime import datetime
start_date = datetime(2018, 1, 1)
end_date = datetime(2018, 1, 31)
def query(city):
    return {
        "city": city,
        "init_date": {"$gte": start_date, "$lte": end_date}
        }

def dataMongoDB(collection, temp_query):
    return pd.DataFrame(list(collection.find(temp_query)))

#def filter_query(city):
    return {
        "city": city,
        "init_date": {"$gte": start_date, "$lte": end_date},
        "$and": [
            {
                "$expr": {
                    "$gt": [
                        {"$subtract": ["$final_time", "$init_time"]},
                        300
                    ]
                }
            },  # <-- Add a comma here
            {
                "$expr": {
                    "$lt": [
                        {"$subtract": ["$final_time", "$init_time"]},
                        18000  # less than 5 hours (18000 sec)
                    ]
                }
            }
        ]
    }

def filter_query(city):
```

```
    return {
        "city": city,
        "init_date": {"$gte": start_date, "$lte": end_date},
        "$and": [
            {
                "$expr": {
                    "$gt": [
                        {"$subtract": ["$final_time", "$init_time"]},
                        300
                    ]
                }
            },
            {
                "$expr": {
                    "$lt": [
                        {"$subtract": ["$final_time", "$init_time"]},
                        18000  # less than 5 hours (18000 sec)
                    ]
                }
            },
            {
                "initial_address": {"$ne": "$final_address"}  # Add this
condition
            }
        ]
    }

def plot_rolling_statistics(data, window_size):
    rolling_mean = data.rolling(window=window_size).mean()
    rolling_std = data.rolling(window=window_size).std()

    plt.figure(figsize=(15, 6))
    plt.plot(data, label='Hourly Bookings (Milan)')
    plt.plot(rolling_mean, label=f'Rolling Mean (window={window_size})')
    plt.plot(rolling_std, label=f'Rolling Std (window={window_size})')

    plt.xlabel('Date and Hour')
    plt.ylabel('Number of Bookings / Rolling Statistics')
    plt.title('Hourly Bookings and Rolling Statistics (Milan Booking)')
    plt.legend()
```

```
    plt.tight_layout()
    plt.grid(True)
    plt.show()



def calculate_and_plot_cdf(data, label):
    sorted_data = np.sort(data)
    cdf = np.arange(len(sorted_data)) / float(len(sorted_data))
    plt.plot(sorted_data, cdf, label=label)

def aggregate_city_data(df):
    df['date_hour'] = df['init_date'].dt.floor('H')
    return df.groupby('date_hour').size()

client = pm.MongoClient('bigdatadb.polito.it',
                        ssl=True,
                        authSource = 'carsharing',
                        username = 'ictts',
                        password ='Ict4SM22!',
                        tlsAllowInvalidCertificates=True)

db = client['carsharing']
PB_enjoy_collection = db['enjoy_PermanentBookings']
PB_collection = db['PermanentBookings']
PP_enjoy_collection = db['enjoy_PermanentParkings']
PP_collection = db['PermanentParkings']

m = "Milano"
f = "Frankfurt"
p = "Portland"
df_milan_PB = dataMongoDB(PB_enjoy_collection, filter_query(m))
df_portland_PB = dataMongoDB(PB_collection, filter_query(p))
df_frankfurt_PB = dataMongoDB(PB_collection, filter_query(f))

df_milan_PB['duration'] = (df_milan_PB['final_time']
-df_milan_PB['init_time']) / 60
df_frankfurt_PB['duration'] = (df_frankfurt_PB['final_time']
-df_frankfurt_PB['init_time']) / 60
```

```
df_portland_PB['duration'] = (df_portland_PB['final_time']
-df_portland_PB['init_time']) / 60


MilanDataPB = df_milan_PB['duration'].values
FrankfurtDataPB = df_frankfurt_PB['duration'].values
PortlandDataPB = df_portland_PB['duration'].values

hourly_counts_milan_PB = aggregate_city_data(df_milan_PB)
hourly_counts_frankfurt_PB = aggregate_city_data(df_frankfurt_PB)
hourly_counts_portland_PB = aggregate_city_data(df_portland_PB)

window_size = 24
plot_rolling_statistics(hourly_counts_milan_PB, window_size)
#plot_rolling_statistics(hourly_counts_portland_PB, window_size)
#plot_rolling_statistics(hourly_counts_frankfurt_PB, window_size)







#calculate_and_plot_cdf(MilanDataPB, 'MilanPermanentBookings')
#calculate_and_plot_cdf(FrankfurtDataPB, 'FrankfurtPermanentBookings')
#calculate_and_plot_cdf(PortlandDataPB, 'PortlandPermanentBookings')

#plt.xlabel('Time duration')
#plt.xscale('log')
#plt.ylabel('CDF')
#plt.title('Three cities CDF/Duration after filter')
#plt.legend()
#plt.grid(True)
#plt.show()
```

```
#ii=df_milan_PB.index[ df_milan_PB [" init_address "] == df_milan_PB
["final_address " ]]. tolist ()
#df_milan_PB_secondfilter = df_milan_PB . drop (ii)
```

4.

```
def calculate_and_plot_cdf(data, label):
    sorted_data = np.sort(data)
    cdf = np.arange(len(sorted_data)) / float(len(sorted_data))
    plt.plot(sorted_data, cdf, label=label)

def aggregate_city_data(df):
    df['date_hour'] = df['init_date'].dt.floor('H')
    return df.groupby('date_hour').size()

client = pm.MongoClient('bigdatadb.polito.it',
                        ssl=True,
                        authSource = 'carsharing',
                        username = 'ictts',
                        password ='Ict4SM22!',
                        tlsAllowInvalidCertificates=True)

def plot_acf_pacf(data_timeseries):
    plt.figure(figsize=(10, 5))

    # Plot ACF
    plt.subplot(1, 2, 1)
    plot_acf(data_timeseries, lags=50, ax=plt.gca())
    plt.xlabel ('lags', fontsize =14)
    plt.ylabel ('ACF', fontsize =14)
    plt.title(' ACF(Milan)', fontsize =16)
    plt . grid ()

    # Plot PACF
    plt.subplot(1, 2, 2)
    plot_pacf(data_timeseries, lags=50, ax=plt.gca())
    plt.xlabel ('lags', fontsize =14)
    plt.ylabel ('PACF', fontsize =14)
```

```
    plt.title('PACF (Milan)', fontsize =16)
    plt . grid ()
    plt.tight_layout()
    plt.show()

db = client['carsharing']
PB_enjoy_collection = db['enjoy_PermanentBookings']
PB_collection = db['PermanentBookings']
PP_enjoy_collection = db['enjoy_PermanentParkings']
PP_collection = db['PermanentParkings']

m = "Milano"
f = "Frankfurt"
p = "Portland"
df_milan_PB = dataMongoDB(PB_enjoy_collection, filter_query(m))
df_portland_PB = dataMongoDB(PB_collection, filter_query(p))
df_frankfurt_PB = dataMongoDB(PB_collection, filter_query(f))

df_milan_PB['duration'] = (df_milan_PB['final_time']
-df_milan_PB['init_time']) / 60
df_frankfurt_PB['duration'] = (df_frankfurt_PB['final_time']
-df_frankfurt_PB['init_time']) / 60
df_portland_PB['duration'] = (df_portland_PB['final_time']
-df_portland_PB['init_time']) / 60


MilanDataPB = df_milan_PB['duration'].values
FrankfurtDataPB = df_frankfurt_PB['duration'].values
PortlandDataPB = df_portland_PB['duration'].values

hourly_counts_milan_PB = aggregate_city_data(df_milan_PB)
hourly_counts_frankfurt_PB = aggregate_city_data(df_frankfurt_PB)
hourly_counts_portland_PB = aggregate_city_data(df_portland_PB)


plot_acf_pacf(hourly_counts_milan_PB)
```

5 , 6

```
def train(dataseries,order, strategy, test_days):
    datapercentage_training = 0.62
```

```
    X = dataseries.values. astype ( float )
    N = int( len(X)* datapercentage_training)
    test_len = 24* test_days
    predictions = np. zeros ((1,test_len ))

    train = X[0:N]
    test = X[N: N+test_len]
    metrics = {"MAE": None," MAPE ": None , "MSE":None , "R2": None }
    history = [ x for x in train ]
    for t in range ( test_len ):
        model = ARIMA(history, order=order)
        model_fit = model.fit()
        output = model_fit . forecast ()
        yhat = output [0]

        predictions [0][t] = yhat

        history . append ( test [t])

        if strategy == " sliding ":
            history = history [1:]




    plt.figure ( figsize = (15 ,5))
    plt . grid ()
    plt . plot (X[N:N+ test_len ],color = 'blue', label = 'original ')
    plt . plot ( predictions[0] , color = 'green',label = 'predicted ')
    plt . legend ()
    plt . xlabel (" test samples ", fontsize = 15)
    plt . title (f" Frankfurt : Predictions with optimal parameters
",fontsize = 10)
    MAE = mean_absolute_error(test, predictions[0])
    MAPE = mean_absolute_error ( test , predictions [0]) / test . mean ()
*100
    MSE = mean_squared_error ( test , predictions[0] )
    R2 = r2_score ( test , predictions[0])
    metrics ["MAE"] , metrics [" MAPE "], metrics [" MSE"], metrics ["R2"]
= MAE, MAPE , MSE , R2
```

```
    print(f'MAE: {MAE:.2f}')
    print(f'MAPE: {MAPE:.2f}%')
    print(f'MSE: {MSE:.2f}')
    print(f'R2: {R2:.2f}')
    return metrics



order = (2,0,2)
train(hourly_counts_frankfurt_PB, order, "sliding", 3)
plt.show()
```

7.a

```
def train(dataseries, order, strategy, test_days):
    datapercentage_training = 0.62
    X = dataseries.values.astype(float)
    N = int(len(X) * datapercentage_training)
    test_len = 24 * test_days
    predictions = np.zeros((1, test_len))

    train = X[0:N]
    test = X[N: N + test_len]
    metrics = {"MAE": None, " MAPE ": None, "MSE": None, "R2": None}
    history = [x for x in train]
    for t in range(test_len):
        model = ARIMA(history, order=order)
        model_fit = model.fit()
        output = model_fit.forecast()
        yhat = output[0]
        predictions[0][t] = yhat

        history.append(test[t])

        if strategy == " sliding ":
            history = history[1:]

    MAE = mean_absolute_error(test, predictions[0])
    MAPE = mean_absolute_error(test, predictions[0]) / test.mean() * 100
    MSE = mean_squared_error(test, predictions[0])
```

```
    R2 = r2_score(test, predictions[0])
    metrics["MAE"], metrics[" MAPE "], metrics[" MSE"], metrics["R2"] =
MAE, MAPE, MSE, R2
    print(f'MAE: {MAE:.2f}')
    print(f'MAPE: {MAPE:.2f}%')
    print(f'MSE: {MSE:.2f}')
    print(f'R2: {R2:.2f}')
    return MAE, MAPE, MSE, R2, predictions , X , N , test_len


def plot_arima_parameter_effect(dataseries, strategy, test_days):
    p_values = range(1, 5)
    d_values = range(1, 2)
    q_values = range(0, 2)

    results = []

    plt.figure(figsize=(15, 10))
    for i, p in enumerate(p_values):
        for j, d in enumerate(d_values):
            for k, q in enumerate(q_values):
                order = (p, d, q)
                MAE, MAPE, MSE, R2, predictions, X , N , test_len =
train(dataseries, order, strategy, test_days)
                results.append({'order': order, 'MAE': MAE, 'MAPE': MAPE,
'MSE': MSE, 'R2': R2, 'predictions': predictions})
                plt . grid ()

                plt . plot ( predictions[0] , label=f"Order {order} ")
                plt . legend ()




    plt.grid()
    plt . plot (X[N:N+ test_len] , color = 'black', linestyle='--', label
= 'original ')
    plt.legend(bbox_to_anchor=(1.05, 1), loc='upper left')
```

```
    plt.suptitle(f"Frankfurt: Effect of Changing p, q on predictions,
d=1")
    plt . xlabel (" test samples ", fontsize = 15)
    plt.ylabel("rentals",fontsize=14)
    plt.tight_layout()
    plt.show()



plot_arima_parameter_effect(hourly_counts_frankfurt_PB,
strategy="sliding", test_days=3)
```

7. displaying error (We ran this code in jupyter to obtain the error tables according to p, q, d)

```
def train(dataseries, order, strategy, test_days):
    datapercentage_training = 0.62
    X = dataseries.values.astype(float)
    N = int(len(X) * datapercentage_training)
    test_len = 24 * test_days
    predictions = np.zeros((1, test_len))

    train = X[0:N]
    test = X[N: N + test_len]

    history = [x for x in train]
    for t in range(test_len):
        model = ARIMA(history, order=order)
        model_fit = model.fit()
        output = model_fit.forecast()
        yhat = output[0]
        predictions[0][t] = yhat

        history.append(test[t])

        if strategy == " sliding ":
            history = history[1:]
```

```
    MAPE = mean_absolute_error(test, predictions[0]) / test.mean() * 100
    MSE = mean_squared_error(test, predictions[0])
    R2 = r2_score(test, predictions[0])

    return MAPE, MSE, R2, predictions , X , N , test_len


def plot_arima_parameter_effect(dataseries, strategy, test_days):
    p_values = range(3, 4)
    d_values = range(0, 1)
    q_values = range(3, 4)

    results_df = pd.DataFrame(columns=['p', 'q', 'd', 'MAPE', 'MSE',
'R2'])



    plt.figure(figsize=(15, 10))
    for i, p in enumerate(p_values):
        for j, d in enumerate(d_values):
            for k, q in enumerate(q_values):
                order = (p, d, q)
                MAPE, MSE, R2, predictions, X , N , test_len =
train(dataseries, order, strategy, test_days)
                results_df = results_df.append({'p': p, 'q': q, 'd': d,
'MAPE': MAPE, 'MSE': MSE, 'R2': R2}, ignore_index=True)
                plt . grid ()

                plt . plot ( predictions[0] , label=f"Order {order} ")
                plt . legend ()




    plt.grid()
    plt . plot (X[N:N+ test_len] , color = 'black', linestyle='--', label
= 'original ')
    plt.legend(bbox_to_anchor=(1.05, 1), loc='upper left')
    plt.suptitle(f"Portland: Effect of Changing p, q on predictions, d=1")
```

```
    plt . xlabel (" test samples ", fontsize = 15)
    plt.ylabel("rentals",fontsize=14)
    plt.tight_layout()
    plt.show()

    display(Markdown("Frankfurt"))
    display(results_df)

    return results_df



plot_arima_parameter_effect(hourly_counts_frankfurt_PB,
strategy="sliding", test_days=3)
```

7. Expanding and sliding , error

```
import pymongo as pm
import pprint
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from datetime import datetime
from statsmodels.tsa.arima.model import ARIMA
from sklearn.metrics import mean_absolute_percentage_error
from sklearn.metrics import mean_squared_error
from sklearn.metrics import mean_absolute_error
from sklearn.metrics import r2_score

start_date = datetime(2018, 1, 1)
end_date = datetime(2018, 1, 31)

def query(city):
    return {
        "city": city,
        "init_date": {"$gte": start_date, "$lte": end_date}
    }

def calculate_and_plot_cdf(data, label):
    sorted_data = np.sort(data)
```

```
    cdf = np.arange(len(sorted_data)) / float(len(sorted_data))
    plt.plot(sorted_data, cdf, label=label)

def aggregate_city_data(df):
    df['init_date'] = pd.to_datetime(df['init_date'])
    df['date_hour'] = df['init_date'].dt.floor('H')
    result = df.groupby('date_hour').size()
    return result

m = "Milano"
f = "Frankfurt"
p = "Portland"
df_milan_PB = pd.read_csv('milan_filtered.csv')
df_portland_PB = pd.read_csv('portland_filtered.csv')
df_frankfurt_PB = pd.read_csv('frankfurt_filtered.csv')

df_milan_PB['duration'] = (df_milan_PB['final_time'] -
df_milan_PB['init_time']) / 60
df_frankfurt_PB['duration'] = (df_frankfurt_PB['final_time'] -
df_frankfurt_PB['init_time']) / 60
df_portland_PB['duration'] = (df_portland_PB['final_time'] -
df_portland_PB['init_time']) / 60

MilanDataPB = df_milan_PB['duration'].values
FrankfurtDataPB = df_frankfurt_PB['duration'].values
PortlandDataPB = df_portland_PB['duration'].values

hourly_counts_milan_PB = aggregate_city_data(df_milan_PB)
hourly_counts_frankfurt_PB = aggregate_city_data(df_frankfurt_PB)
hourly_counts_portland_PB = aggregate_city_data(df_portland_PB)

def arima_window_effect(data, order, strategy, window_init, window_end,
test_days):
    X = data.values.astype(float)
    N_list = list(np.arange(window_init, window_end, test_days) * 24)
    test_len = 24 * test_days
    predictions = np.zeros((len(N_list), test_len))
    error_arima = pd.DataFrame(columns=['N[ hours ]', 'N[ days ]', 'MAPE
', 'MSE ', 'R2 ', 'Predictions'])
    plt.figure(figsize=(15, 5))
```

```
    plt.plot(X[N_list[0]: N_list[-1] + test_len], label='Original data')
    j = 0
    i = 0
    for step, N in enumerate(N_list):
        train_size = int(N)
        print(N)
        if strategy == 'expanding ':
            print(f'Testing ARIMA - expanding window step : {N / 24}
Days')
            train = X[0:train_size]
            test = X[train_size: train_size + test_len]
            history = [x for x in train]
        else:
            print(f" Testing ARIMA - sliding window : {step}")
            i = step * test_len
            train = X[i: train_size]
            test = X[train_size: train_size + test_len]
            history = [x for x in train]

            if len(test) < test_len:
                print(len(test), test_len)
                test_len = len(test)

            for t in range(test_len):
                model = ARIMA(history, order=order)
                model_fit = model.fit()
                output = model_fit.forecast()
                yhat = output[0]
                predictions[N_list.index(N)][t] = yhat
                history.append(test[t])

        x_val = np.arange(j, test_len + j, 1)
        j += test_len
        plt.plot(x_val, predictions[N_list.index(N)], label=f'DaysWindow
={N / 24}', linestyle='dashed')

        mape = mean_absolute_error(test, predictions[N_list.index(N)]) /
test.mean() * 100
        mse = mean_squared_error(test, predictions[N_list.index(N)])
        r2 = r2_score(test, predictions[N_list.index(N)])
```

```
        error_arima = error_arima.append({'N[ hours ]': int(N), 'N[ days
]': int(N / 24),'MAPE ': mape, 'MSE ': mse, 'R2 ': r2,'Predictions':
predictions[N_list.index(N)]}, ignore_index=True)

    plt.grid()
    plt.xlabel('Time (hours)', fontsize=15)
    plt.ylabel('Rentals', fontsize=15)
    plt.title(f'ARIMA Predictions: Portland, Strategy: {strategy}',
fontsize=17)
    plt.legend(bbox_to_anchor=(1.05, 1), loc='upper left')
    plt.tight_layout()
    plt.show()


    return error_arima
```

To obtain the results for error we run the code in jupyter and add the below code:

```
display(error_arima)
return error_arima
```