Politecnico di Torino
ICT for smart mobility
Laboratory Report 2

Group 3
s306627 Canineo Komar Vitor
s306667 Gamballi Lucca
s308673 Seyedeh Sarah Ghiyasi

Melia Marco
Vassio Luca

Academic Year 2023-2024

1 Prediction using ARIMA models

do not use passive and impersonal form

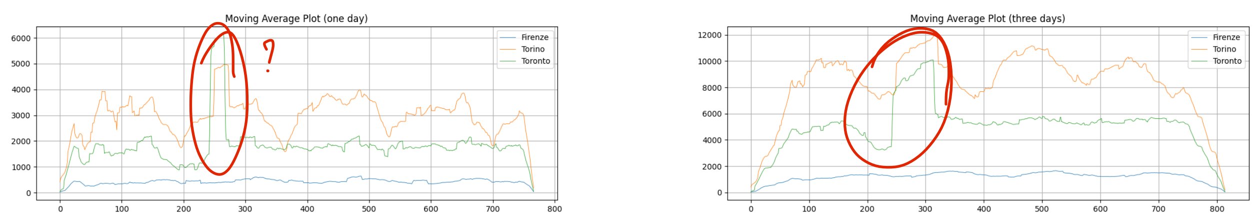
1.1 Extracted Period

In order to choose the month to be analyzed it was important to take into account the seasonality of certain phenomena such as holidays, for instance, December would not be a good candidate to be analyzed due to the Christmas and New Year holidays, for similar reasons August would be a bad candidate, in both cases the process would most likely be nonstationary, making the analysis less accurate. Therefore October was chosen as the month to be analyzed, because there aren't any major events during this month, and seems pretty regular in behavior.

1.2 Missing Values and Filtering

There are currently 736 data points available out of a total of 736. Consequently, there are some missing values. Given that the ARIMA model cannot handle missing data, it is essential to impute these missing values. One effective technique involves propagating the values from the last available sample. Since the number of missing points is minimal, this approach should suffice. Also, it is important to point out that the data was filtered using the same process as in the previous lab. bookings with less than two minutes and more than 3 hours were removed as well as bookings that had the same initial and final address.

1.3 Stationarity



(a) Moving average using one day window

(b) Moving average using three days window

Figure 1: Moving average for different window size

which window?

To check stationarity a moving window approach was used, this can be seen on figure1, both graphs don't show any visible trend, therefore the process can be considered stationary.

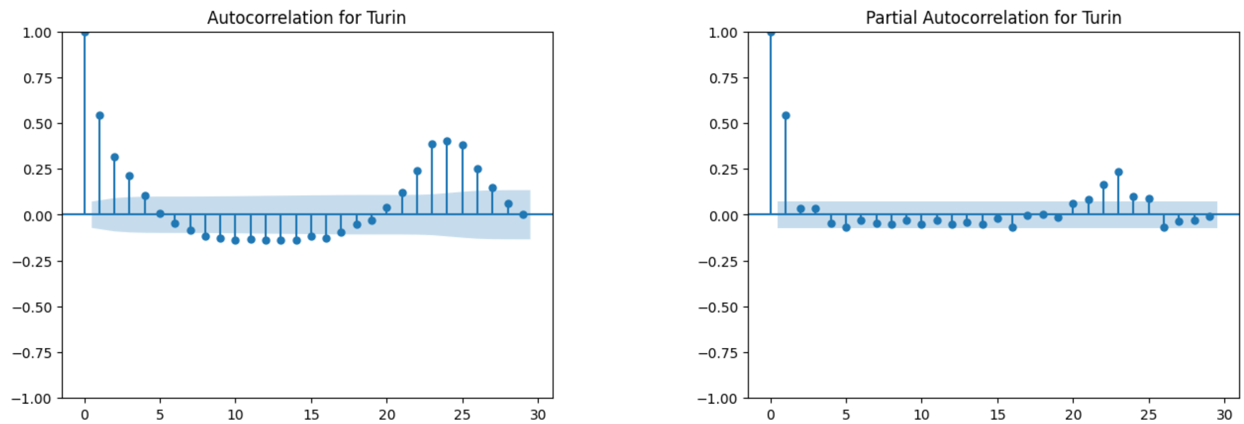
but they show clear outlier(!) why(?)

1.4 Autocorrelation and Partial Autocorrelation

The Autocorrelation and Partial Autocorrelation functions can be seen on figures 2,3 and4.
In Turin, the Partial AutoCorrelation Function (PACF) exhibits a pronounced drop, reaching zero rapidly after q=1. Conversely, the AutoCorrelation Function (ACF) demonstrates a gradual decline, converging to zero around p=5.
The patterns observed in Florence closely mirror those in Turin. In Florence, the PACF experiences a significant decrease, reaching zero swiftly after q=1, while the ACF gradually diminishes, attaining zero around p=5. This parallel behavior suggests similarities in habits between the two Italian cities.
Toronto, however, deviates from the observed patterns in the Italian cities. In Toronto, the ACF follows a different trajectory, declining gently and reaching zero after p=2. This distinctive behavior in Toronto's ACF indicates dissimilarities in the time series dynamics compared to the Italian cities.

1.5 Training and Test Datasets

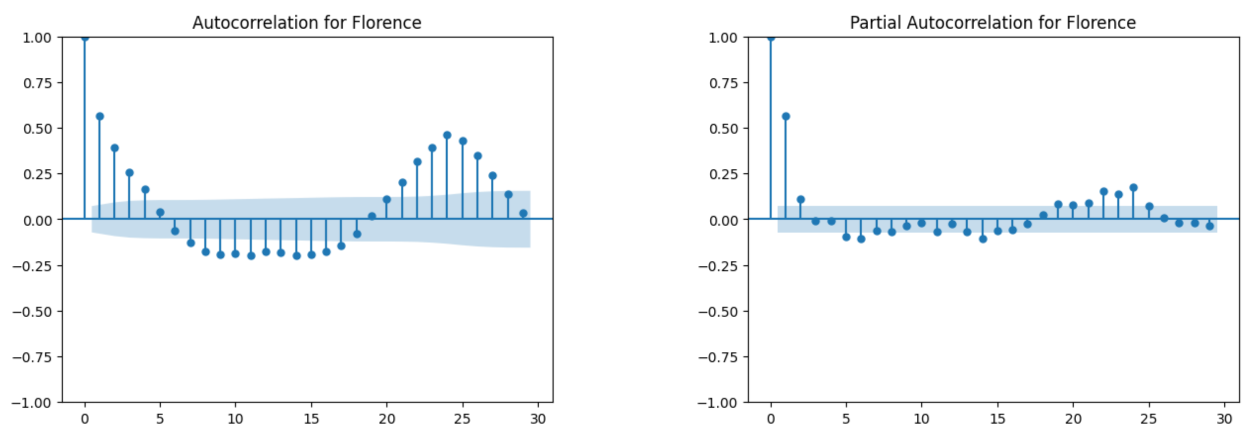
It was decided to use the last week of the month as test dataset, consequently the first tree weeks were used as training. Considering the 31 days of October and (each with 24 hours), this division resulted in a



(a) Autocorrelation for Turin

(b) Partial Autocorrelation for Turin

Figure 2: Autocorrelation and Partial Autocorrelation for Turin



(a) Autocorrelation for Florence

(b) Partial Autocorrelation for Florence

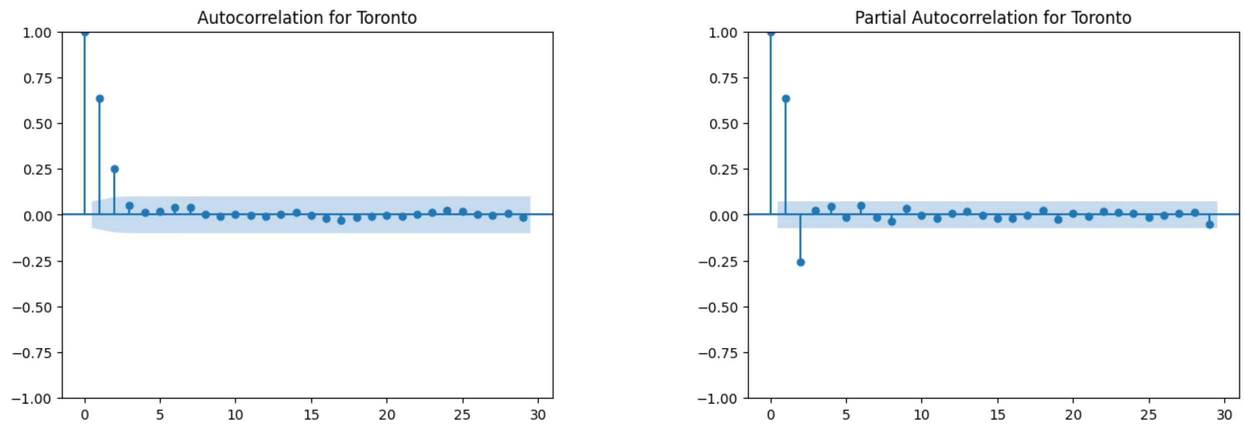
Figure 3: Autocorrelation and Partial Autocorrelation for Florence

training dataset with 504 samples and a test dataset with 240 samples. Which means that the train and test sets represent approximately, 70% and 30% of the data, respectively. This division of the data was adopted in order to use the entirety of available data, additionally it allows to keep a lengthy enough training dataset while not compromising the test set size.

1.6 Model training

Considering the division of the dataset mentioned in Section 1.5, and the fact that the dataset can be considered a stationary process, an ARIMA model of order (2,0,2) was trained for the cities Torino, Firenze and Toronto. The results obtained on the test set can be seen in Table 1. Additionally, Figures 5, 6 and 7 represent, respectively, the comparison between the ground truth and the predicted value. In those figures the ground truth is shown in blue, whereas the predicted values are shown in red, as it is shown the results were very good, with the predicted values closely following the ground truth values. It is also interesting to point out that the predicted values seem shifted to the right in relation to the ground truth, for instance when observing large peaks, this can be explained by the model being influenced by last values, the model assumed that the future will be relatively similar to the past when large values occur this does not hold.

not really ... the error is sizeable
compare with a baseline predictor (?)



(a) Autocorrelation for Toronto

(b) Partial Autocorrelation for Toronto

Figure 4: Autocorrelation and Partial Autocorrelation for Toronto

absolute error? or percentage error?

	Torino	Firenze	Toronto
MAPE	0.8799	0.5878	0.9005

0,8% ? are you sure? This look very low.

Table 1: Mean absolute percentage error for each city.

1.7 Grid Search

1.7.1 Keep the number of training samples fixed, and do a grid search varying (p,d,q) and observe how the error changes.

Considering the same size of training and test sets discussed in Section 1.5, values of $p, q \in [2, 6]$ were considered to perform the Grid Search. Different values of d were not considered due to the fact that the process is stationary, therefore the d can be set to zero. Figure 8 shows the mean absolute percentage error (MAPE) obtained by each model trained and for each city, among Torino, Firenze and Toronto. In Figure 8, models of order that resulted in lower-upper (LU) decomposition errors are represented by -1. d=0

Analysing Figure 8 and considering Firenze, it can be seen that there are a couple of equivalent models. Those are the models of orders (2,0,3), (2,0,4), (3,0,3) and (3,0,4), all of which resulted in a MAPE value of approximately 0.499. Therefore the best option is the model of order (2,0,3) because it produces the same performance metric with less complexity.

The same reasoning can be applied to the city of Toronto. In Figure 8 it can be seen that even more models can be considered equivalent order. Those are the models of orders (2,0,2), (2,0,3), (2,0,6), (3,0,3),

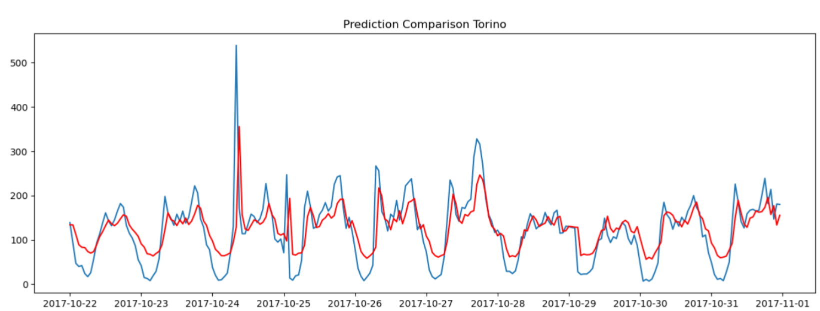
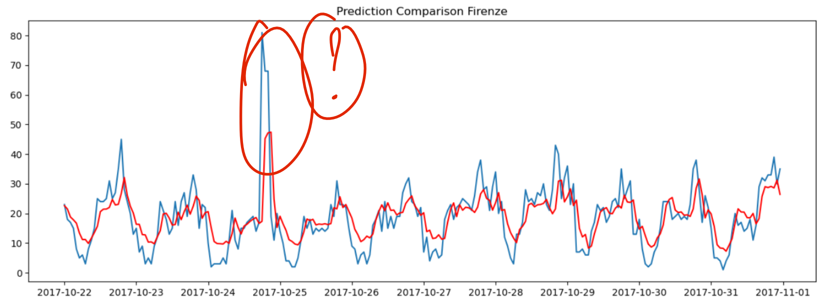


Figure 5: Torino prediction comparison.



why these outliers ?

Figure 6: Firenze prediction comparison.

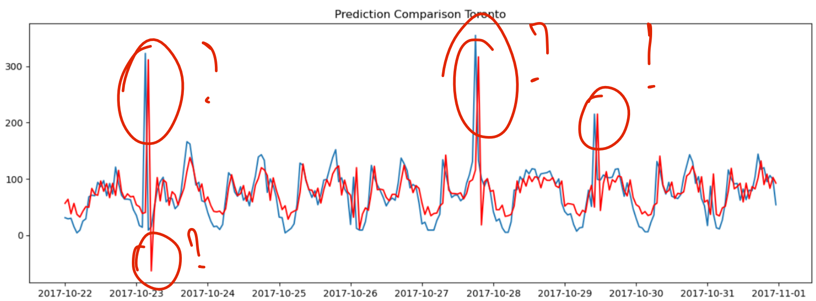


Figure 7: Toronto prediction comparison.

90% or 0.9% ?

(4,0,3) and (4,0,4) all of them resulted in a MAPE value of approximately 0.90. Therefore, applying the logic used to decide the best model for Firenze, the best option for Toronto is the model of order (2,0,2).

Finally, for Torino there is one model that outstands the others, which is the one of order (3,0,5). This model produced a MAPE of approximately 0.62, whereas all other models are less optimal. The closest one stands at a MAPE value of 0.65, while all others have values greater than 0.67.

these are very high ?! why ?

1.7.2 Given the best parameter configuration, change N and the learning strategy

Knowing the best parameter configuration for each city, we proceeded to evaluate the effect of changing the size of the training set (represented by N) and the learning strategy. Two learning strategies were considered, sliding window and expanding window. For what considers N, the array $\mathbf{N} = \begin{bmatrix} 11 & 28 & \dots & 504 \end{bmatrix}$ represent the thirty different values adopted. A full representation of $\mathbf{N}$ can be seen in Equation 1. Figure 9 shows

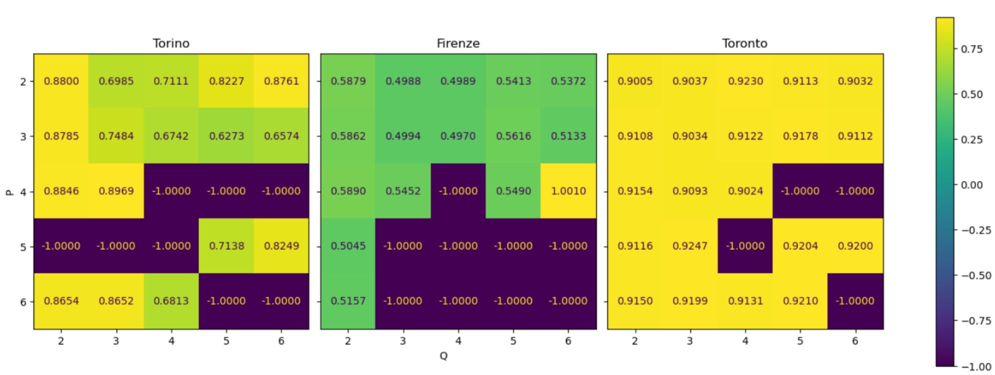
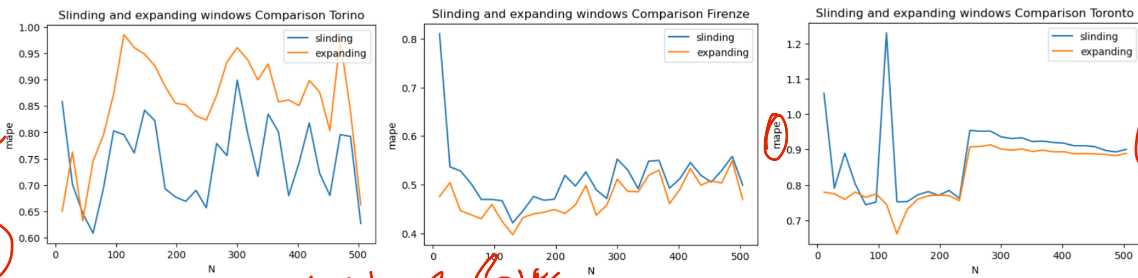


Figure 8: Grid Search results.

the result on the test set for Torino, Firenze and Toronto. It is worth mentioning that when evaluating the performance over all values of N adopted, the test set was fixed for each city.



(a) Torino window comparison. (b) Firenze window comparison. (c) Toronto window comparison.

Figure 9: Slidind and expanding windows comparison.

why there is no trend here (?)

again errors are suspiciously too high or low

1.7.3 Compare results for the different cities. How the relative error changes w.r.t. the absolute number of rentals?

Analysing Figure 9a, it can be seen that for almost all values of N the expanding window learning strategy produces a greater error than the sliding one. The only values of N for which the sliding is performing worse are the ones that are closer to zero. Additionally, it can be seen that the smaller MAPE values occurs when considering the sliding window and when N is approximately 50. This result is an indicative that, for Torino, the optimal N value is in the vicinity of $N = 50$ and sliding window is the best learning strategy.

Considering Figure 9b, it can be seen that for all values of N the sliding results in a MAPE value greater or equal than the ones that result from an expanding window strategy. Furthermore, it can be seen that the smaller MAPE values occurs when considering the expanding window and when N is close to 120. This result is an indicative that, for Firenze, the optimal N value is in the vicinity of $N = 120$ and expanding window is the best learning strategy. Additionaly, it can be seen that in general the difference between the two strategies is smaller in the case of Firenze than in the case of Torino.

From Figure 9c it can be seen that for values of N greater than 200 both strategies follow a similar behaviour. For these values of N the expanding window slightly outperforms the sliding window. For values smaller than 200, it can be seen that the curves do not behave in the same way. Moreover, it can be seen that the smaller MAPE values occurs when considering the expanding window and when N is, once again, close to 120. This result is an indicative that, for Toronto, the optimal N value is in the vicinity of $N = 120$ and expanding window is the best learning strategy. Finally, when analysing the sliding window from Figures 9b and 9c, it can be seen that for certain values of N there is a peak in the MAPE value. Take $N = 113$ in Figure 9c as an example. A possible explanation for this behavior is that the model has overfitting issues for those values of N.

(?)

- results are not consistent with the typical errors seen in the class and by other groups. check if there are any problems and what type of error do you use
- no time horizon comparison
- no baseline comparison

RESUBMIT after fixing

2 Appendix

2.1 Values of N

Equation 1 shows all values adopted for N,

$$\mathbf{N} = \begin{bmatrix} 11 & 28 & 45 & 62 & 79 & 96 & 113 & 130 & 147 & 164 \\ 181 & 198 & 215 & 232 & 249 & 266 & 283 & 300 & 317 & 334 \\ 351 & 368 & 385 & 402 & 419 & 436 & 453 & 470 & 487 & 504 \end{bmatrix}. \quad (1)$$

2.2 Code

```
""" lab2

    authors :
        Lucca Gamballi
        Sarah Ghiyasi Seyedeh
        Vitor Canineo Komar

    ARIMA models
"""


import random as rnd
from statsmodels.graphics.tsaplots import plot_acf
from statsmodels.graphics.tsaplots import plot_pacf
from statsmodels.tsa.arima.model import ARIMA
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from matplotlib import colors
import pymongo as pm
import pprint
from datetime import datetime
from sklearn.metrics import mean_squared_error
from sklearn.metrics import mean_absolute_error
from sklearn.metrics import r2_score
from sklearn.metrics import mean_absolute_percentage_error
from sklearn.metrics import ConfusionMatrixDisplay
import seaborn as sn

get_ipython().run_line_magic('matplotlib', 'inline')

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

PermanentBookings = db['PermanentBookings']
ActiveBookings = db['ActiveBookings']

PermanentParkings = db['PermanentParkings']
ActiveParkings = db['ActiveParkings']

enjoy_PermanentBookings = db['enjoy_PermanentBookings']
enjoy_ActiveBookings = db['enjoy_ActiveBookings']

enjoy_PermanentParkings = db['enjoy_PermanentParkings']
enjoy_ActiveParkings = db['enjoy_ActiveParkings']

```

do not compare strings use locations

```
def make_hour_df_filter(collection, city):
    aggregation = collection.aggregate([{
                                          '$match' : {'city': city,
                                                      'init_address': {
                                                                        '$ne': '$final_address'   # 'init_address' not equal to 'final_address'
                                                                    }
                                                      }
                                        },
                                        {
                                        '$project' : {'duration': {'$subtract': ['$final_time', '$init_time']},
                                                      'hour': {'$hour': '$init_date'},
                                                      'year': {'$year': '$init_date'},
                                                      "timestamp":{ "$dateToString":{"format":"%Y-%m-%d-%H", "date":"$init_date"}}}
                                        },
                                        {
                                                    '$match': {
                                                        'duration': {
                                                                    '$gte': 60,         # Greater than or equal to 120 seconds
                                                                    '$lte': 3600 * 3    # Less than or equal to 3600*3 seconds
                                                                }
                                                    }
                                        },
                                        {'$group' : {
                                                    '_id' : '$timestamp',
                                                    'count' : {'$sum': 1},
                                                    'avg' : {'$avg': 1}
                                                    }
                                        }])

    hours = []
    bookings = []
    for el in aggregation:
        hours.append(el['_id'])
        bookings.append(el['count'])
    df = pd.DataFrame({'hours':hours, 'count':bookings})
    df['hours'] = pd.to_datetime(df['hours'], format="%Y-%m-%d-%H")
    df = df.sort_values('hours')
    mask = (df['hours'].dt.year == 2017) & (df['hours'].dt.month == 10)
    df = df[mask]
    return df

hourDfFirenze = make_hour_df_filter(enjoy_PermanentBookings, 'Firenze')
hourDfTorino = make_hour_df_filter(PermanentBookings, 'Torino')
hourDfToronto = make_hour_df_filter(PermanentBookings, 'Toronto')


# ### 2 Checking for missing values
len(hourDfTorino)
# There are missing values

def fill_missing_values(df):
    df = (df.set_index('hours')
      .reindex(pd.date_range('2017-10-01', '2017-11-1', freq='h'))
      .rename_axis(['hours'])).fillna(method='ffill').iloc[:-1]
    return df

hourDfFirenze = fill_missing_values(hourDfFirenze)
hourDfTorino = fill_missing_values(hourDfTorino)
hourDfToronto = fill_missing_values(hourDfToronto)

```

```
plt.figure(figsize=(12, 4))
plt.plot(hourDfFirenze.index, hourDfFirenze['count'], label='Firenze', linewidth=0.6)
plt.plot(hourDfTorino.index, hourDfTorino['count'], label='Torino', linewidth=0.6)
plt.plot(hourDfToronto.index, hourDfToronto['count'], label='Toronto', linewidth=0.6)
plt.title('Bookings per Hour (Filtered)')
plt.tight_layout()
plt.legend()
plt.grid()
plt.show()
```

what does the convolve method do ?

```
plt.figure(figsize=(12, 4))
plt.plot(np.convolve(hourDfFirenze['count'], np.ones(6)), label='Firenze', linewidth=0.6)
plt.plot(np.convolve(hourDfTorino['count'], np.ones(6)), label='Torino', linewidth=0.6)
plt.plot(np.convolve(hourDfToronto['count'], np.ones(6)), label='Toronto', linewidth=0.6)
plt.legend()
plt.title('Moving Average Plot (six hours)')
plt.grid()
plt.show()

plt.figure(figsize=(12, 4))
plt.plot(np.convolve(hourDfFirenze['count'], np.ones(24)), label='Firenze', linewidth=0.6)
plt.plot(np.convolve(hourDfTorino['count'], np.ones(24)), label='Torino', linewidth=0.6)
plt.plot(np.convolve(hourDfToronto['count'], np.ones(24)), label='Toronto', linewidth=0.6)
plt.legend()
plt.title('Moving Average Plot (one day)')
plt.grid()
plt.show()

plt.figure(figsize=(12, 4))
plt.plot(np.convolve(hourDfFirenze['count'], np.ones(24*3)), label='Firenze', linewidth=0.6)
plt.plot(np.convolve(hourDfTorino['count'], np.ones(24*3)), label='Torino', linewidth=0.6)
plt.plot(np.convolve(hourDfToronto['count'], np.ones(24*3)), label='Toronto', linewidth=0.6)
plt.legend()
plt.title('Moving Average Plot (three days)')
plt.grid()
plt.show()

plt.figure(figsize=(12, 4))
plt.plot(np.convolve(hourDfFirenze['count'], np.ones(24*7)), label='Firenze', linewidth=0.6)
plt.plot(np.convolve(hourDfTorino['count'], np.ones(24*7)), label='Torino', linewidth=0.6)
plt.plot(np.convolve(hourDfToronto['count'], np.ones(24*7)), label='Toronto', linewidth=0.6)
plt.legend()
plt.title('Moving Average Plot (seven days)')
plt.grid()
plt.show()

plot_acf(hourDfTorino['count'], title='Autocorrelation for Turin')

plot_pacf(hourDfTorino['count'], title='Partial Autocorrelation for Turin')

plot_acf(hourDfFirenze['count'], title='Autocorrelation for Florence')

plot_pacf(hourDfFirenze['count'], title='Partial Autocorrelation for Florence')

plot_acf(hourDfToronto['count'], title='Autocorrelation for Toronto')

plot_pacf(hourDfToronto['count'], title='Partial Autocorrelation for Toronto')


p=2; d=0; q=2
modelTorino = ARIMA(hourDfTorino['count'], order=(p,d,q))
modelTorino = modelTorino.fit()

modelFirenze = ARIMA(hourDfFirenze['count'], order=(p,d,q))
modelFirenze = modelFirenze.fit()
```

```

modelToronto = ARIMA(hourDfToronto['count'], order=(p,d,q))
modelToronto = modelToronto.fit()

fig = plt.figure(figsize=(15,5))
plt.plot(hourDfTorino['count'])
plt.plot(modelTorino.fittedvalues, color='red')
plt.title('Prediction Comparison Torino')

fig = plt.figure(figsize=(15,5))
plt.plot(hourDfFirenze['count'])
plt.plot(modelFirenze.fittedvalues, color='red')
plt.title('Prediction Comparison Firenze')

fig = plt.figure(figsize=(15,5))
plt.plot(hourDfToronto['count'])
plt.plot(modelToronto.fittedvalues, color='red')
plt.title('Prediction Comparison Toronto')
```

MAPE (?)

```

mean_squared_error(hourDfTorino['count'], modelTorino.fittedvalues), mean_absolute_error(hourDfTorino['count'], modelTorino.fittedvalues)
mean_squared_error(hourDfFirenze['count'], modelFirenze.fittedvalues), mean_absolute_error(hourDfFirenze['count'], modelFirenze.fittedvalues)
mean_squared_error(hourDfToronto['count'], modelToronto.fittedvalues), mean_absolute_error(hourDfToronto['count'], modelToronto.fittedvalues)

def trainTestSplit(df, split):
    train = df[0:split]
    test = df[split:df.shape[0]]

    return train, test

def walkFowardValidation(train, test, order, windowType='sliding'):
    history = [x for x in train]
    predictions = []

    for i, testSample in enumerate(test):
        model = ARIMA(history, order=order)
        model_fit = model.fit()
        output = model_fit.forecast()
        yhat = output[0]
        predictions.append(yhat)
        history.append(testSample)
        if windowType=='sliding':
          history.pop(0)

    return predictions

trainSamples = 3*7*24 #first 3 week of samples
N = trainSamples

trainTorino, testTorino = trainTestSplit(hourDfTorino['count'], trainSamples)
trainFirenze, testFirenze = trainTestSplit(hourDfFirenze['count'], trainSamples)
trainToronto, testToronto = trainTestSplit(hourDfToronto['count'], trainSamples)

print(hourDfTorino['count'].shape, hourDfFirenze['count'].shape, hourDfToronto['count'].shape)
print(trainTorino.shape, trainFirenze.shape, trainToronto.shape)
print(testTorino.shape, testFirenze.shape, testToronto.shape)

p=2; d=0; q=2
order = (p,d,q)
predsTorino = walkFowardValidation(trainTorino, testTorino, order, windowType='sliding')
predsFirenze = walkFowardValidation(trainFirenze, testFirenze, order, windowType='sliding')
predsToronto = walkFowardValidation(trainToronto, testToronto, order, windowType='sliding')
```

```python

fig = plt.figure(figsize=(15,5))
plt.plot(testTorino, label='truth')
plt.plot(testTorino.index, predsTorino, color='red', label='prediction')
plt.title('Prediction Comparison Torino')

fig = plt.figure(figsize=(15,5))
plt.plot(testFirenze, label='truth')
plt.plot(testFirenze.index, predsFirenze, color='red', label='prediction')
plt.title('Prediction Comparison Firenze')

fig = plt.figure(figsize=(15,5))
plt.plot(testToronto, label='truth')
plt.plot(testToronto.index, predsToronto, color='red', label='prediction')
plt.title('Prediction Comparison Toronto')

print(mean_squared_error(testTorino, predsTorino), mean_absolute_error(testTorino, predsTorino), mean_absolute_percentage_error(testTorino, predsTorino))
print(mean_squared_error(testFirenze, predsFirenze), mean_absolute_error(testFirenze, predsFirenze), mean_absolute_percentage_error(testFirenze, predsFirenze))
print(mean_squared_error(testToronto, predsToronto), mean_absolute_error(testToronto, predsToronto), mean_absolute_percentage_error(testToronto, predsToronto))

P = [i for i in range(2,7)]
Q = [i for i in range(2,7)]
d = 0

resultsTorino = []
resultsFirenze = []
resultsToronto = []
for p in P:
  for q in Q:
    order = (p,d,q)
    print(order)
    predsTorino = walkFowardValidation(trainTorino, testTorino, order, windowType='sliding')
    mse = mean_squared_error(testTorino, predsTorino)
    mae = mean_absolute_error(testTorino, predsTorino)
    mape = mean_absolute_percentage_error(testTorino, predsTorino)
    resultsTorino.append((order, mse, mae, mape))

    predsFirenze = walkFowardValidation(trainFirenze, testFirenze, order, windowType='sliding')
    mse = mean_squared_error(testFirenze, predsFirenze)
    mae = mean_absolute_error(testFirenze, predsFirenze)
    mape = mean_absolute_percentage_error(testFirenze, predsFirenze)
    resultsFirenze.append((order, mse, mae, mape))

    predsToronto = walkFowardValidation(trainToronto, testToronto, order, windowType='sliding')
    mse = mean_squared_error(testToronto, predsToronto)
    mae = mean_absolute_error(testToronto, predsToronto)
    mape = mean_absolute_percentage_error(testToronto, predsToronto)
    resultsToronto.append((order, mse, mae, mape))

## considering TORINO
## restart from order 5,0,5 due to LU decomposition error on orders
##    4,0,4 --- 4,0,5 --- 4,0,6
##    5,0,2 --- 5,0,3 --- 5,0,4
## orders 6,0,5 and 6,0,6 also had LU decomposition error

P = [i for i in range(4,7)]
Q = [i for i in range(2,7)]
d = 0

for p in P:
```

```python
  for q in Q:
      if p == 4 and q < 7:
          pass
      elif p == 5 and q < 5:
          pass
      else:
        order = (p,d,q)
        print(order)
        predsTorino = walkFowardValidation(trainTorino, testTorino, order, windowType='sliding')
        mse = mean_squared_error(testTorino, predsTorino)
        mae = mean_absolute_error(testTorino, predsTorino)
        mape = mean_absolute_percentage_error(testTorino, predsTorino)
        resultsTorino.append((order, mse, mae, mape))

## considering FIRENZE
## restart from order 4,0,5 due to LU decomposition error on orders
##    4,0,4
P = [i for i in range(4,7)]
Q = [i for i in range(2,7)]
d = 0

for p in P:
  for q in Q:
      if p == 4 and q < 5:
          pass
      else:
        order = (p,d,q)
        print(order)
        predsFirenze = walkFowardValidation(trainFirenze, testFirenze, order, windowType='sliding')
        mse = mean_squared_error(testFirenze, predsFirenze)
        mae = mean_absolute_error(testFirenze, predsFirenze)
        mape = mean_absolute_percentage_error(testFirenze, predsFirenze)
        resultsFirenze.append((order, mse, mae, mape))

## considering FIRENZE
## restart from order 6,0,2 due to LU decomposition error on orders
##    5,0,3 --- 5,0,4 --- 5,0,5 --- 5,0,6
## from 6,0,3 all orders result in LU decomposition error
P = [i for i in range(5,7)]
Q = [i for i in range(2,7)]
d = 0

for p in P:
  for q in Q:
      if p == 5 and q < 7:
          pass
      else:
        order = (p,d,q)
        print(order)
        predsFirenze = walkFowardValidation(trainFirenze, testFirenze, order, windowType='sliding')
        mse = mean_squared_error(testFirenze, predsFirenze)
        mae = mean_absolute_error(testFirenze, predsFirenze)
        mape = mean_absolute_percentage_error(testFirenze, predsFirenze)
        resultsFirenze.append((order, mse, mae, mape))

## considering TORONTO
## restart from order 4,0,6 due to LU decomposition error on orders
##    4,0,5 --- 4,0,6
##    5,0,2 --- 5,0,3 --- 5,0,4
P = [i for i in range(4,7)]
Q = [i for i in range(2,7)]
d = 0
```

```
for p in P:
  for q in Q:
      if p == 4 and q < 7:
          pass
      elif p == 5 and q < 5:
          pass
      else:
        order = (p,d,q)
        print(order)
        predsToronto = walkFowardValidation(trainToronto, testToronto, order, windowType='sliding')
        mse = mean_squared_error(testToronto, predsToronto)
        mae = mean_absolute_error(testToronto, predsToronto)
        mape = mean_absolute_percentage_error(testToronto, predsToronto)
        resultsToronto.append((order, mse, mae, mape))

ordersListsTorino = []
for index, element in enumerate(resultsTorino):
    ordersListsTorino.append(element[0])

ordersListsFirenze = []
for element in resultsFirenze:
    ordersListsFirenze.append(element[0])

ordersListsToronto = []
for element in resultsToronto:
    ordersListsToronto.append(element[0])

metricTorino = np.zeros((5,5))
metricFirenze = np.zeros((5,5))
metricToronto = np.zeros((5,5))
elTorino = 0
elFirenze = 0
elToronto = 0
for p in range(2,7):
  for q in range(2,7):
      if (p,d,q) in ordersListsTorino:
        metricTorino[p-2][q-2] = resultsTorino[elTorino][3]
        elTorino += 1
      else:
        metricTorino[p-2][q-2] = -1

      if (p,d,q) in ordersListsFirenze:
        metricFirenze[p-2][q-2] = resultsFirenze[elFirenze][3]
        elFirenze += 1
      else:
        metricFirenze[p-2][q-2] = -1

      if (p,d,q) in ordersListsToronto:
        metricToronto[p-2][q-2] = resultsToronto[elToronto][3]
        elToronto += 1
      else:
        metricToronto[p-2][q-2] = -1

data = [
    {"city": "Torino", "results": metricTorino},
    {"city": "Firenze", "results": metricFirenze},
    {"city": "Toronto", "results": metricToronto},
]


f, axes = plt.subplots(1, 3, figsize=(15, 5), sharey='row')

for i, element in enumerate(data):
```

```
    disp = ConfusionMatrixDisplay(element['results'], display_labels=[2,3,4,5,6])
    disp.plot(ax=axes[i], values_format=".4f")
    disp.ax_.set_title(element['city'])
    disp.im_.colorbar.remove()
    disp.ax_.set_xlabel('')
    if i!=0:
        disp.ax_.set_ylabel('')
    else:
        disp.ax_.set_ylabel('P')

f.text(0.4, 0.05, 'Q', ha='left')
plt.subplots_adjust(wspace=0.40, hspace=0.1)

f.tight_layout()
f.colorbar(disp.im_, ax=axes)
plt.show()


# from grid search
bestOrderTorino = (3,0,5)
bestOrderFirenze = (2,0,3)
bestOrderToronto = (2,0,2)

trainSamples = 3*7*24 #first 3 week of samples
N = trainSamples
trainTorino, fixedTest = trainTestSplit(hourDfTorino['count'], trainSamples) #set fixed test
     set

mseSliding = []
mseExpanding = []

maeSliding = []
maeExpanding = []

mapeSliding = []
mapeExpanding = []
N = [i for i in range(trainSamples, 1, -trainSamples//(30))] #range(start, stop, step)
for n in N:
  print(n)
  trainTorino, testTorino = trainTestSplit(hourDfTorino['count'], n)
  predsTorinoSliding = walkFowardValidation(trainTorino, fixedTest, bestOrderTorino,
    windowType='sliding')
  predsTorinoExpanding = walkFowardValidation(trainTorino, fixedTest, bestOrderTorino,
    windowType='expanding')

  mseSliding.append(mean_squared_error(fixedTest, predsTorinoSliding))
  mseExpanding.append(mean_squared_error(fixedTest, predsTorinoExpanding))

  maeSliding.append(mean_absolute_error(fixedTest, predsTorinoSliding))
  maeExpanding.append(mean_absolute_error(fixedTest, predsTorinoExpanding))

  mapeSliding.append(mean_absolute_percentage_error(fixedTest, predsTorinoSliding))
  mapeExpanding.append(mean_absolute_percentage_error(fixedTest, predsTorinoExpanding))

fig = plt.figure(figsize=(6,4))
plt.plot(N, mseSliding, label='slinding')
plt.plot(N, mseExpanding, label='expanding')
plt.title('Slinding and expanding windows Comparison Torino')
plt.legend()
plt.xlabel('N')
plt.ylabel('mse')
plt.show()

fig = plt.figure(figsize=(6,4))
plt.plot(N, maeSliding, label='slinding')
```

```
plt.plot(N, maeExpanding, label='expanding')
plt.title('Slinding and expanding windows Comparison Torino')
plt.legend()
plt.xlabel('N')
plt.ylabel('mae')
plt.show()

fig = plt.figure(figsize=(6,4))
plt.plot(N, mapeSliding, label='slinding')
plt.plot(N, mapeExpanding, label='expanding')
plt.title('Slinding and expanding windows Comparison Torino')
plt.legend()
plt.xlabel('N')
plt.ylabel('mape')
plt.show()


# In[266]:


trainSamples = 3*7*24 #first 3 week of samples
N = trainSamples
trainFirenze, fixedTest = trainTestSplit(hourDfFirenze['count'], trainSamples) #set fixed
    test set

mseSliding = []
mseExpanding = []

maeSliding = []
maeExpanding = []

mapeSliding = []
mapeExpanding = []
N = [i for i in range(trainSamples, 1, -trainSamples//(30))]
for n in N:
  print(n)
  trainFirenze, testFirenze = trainTestSplit(hourDfFirenze['count'], n)
  predsFirenzeSliding = walkFowardValidation(trainFirenze, fixedTest, (2,0,3), windowType='
    sliding')
  predsFirenzeExpanding = walkFowardValidation(trainFirenze, fixedTest, (2,0,3), windowType=
    'expanding')

  mseSliding.append(mean_squared_error(fixedTest, predsFirenzeSliding))
  mseExpanding.append(mean_squared_error(fixedTest, predsFirenzeExpanding))

  maeSliding.append(mean_absolute_error(fixedTest, predsFirenzeSliding))
  maeExpanding.append(mean_absolute_error(fixedTest, predsFirenzeExpanding))

  mapeSliding.append(mean_absolute_percentage_error(fixedTest, predsFirenzeSliding))
  mapeExpanding.append(mean_absolute_percentage_error(fixedTest, predsFirenzeExpanding))

fig = plt.figure(figsize=(6,4))
plt.plot(N, mseSliding, label='slinding')
plt.plot(N, mseExpanding, label='expanding')
plt.title('Slinding and expanding windows Comparison Firenze')
plt.legend()
plt.xlabel('N')
plt.ylabel('mse')
plt.show()

fig = plt.figure(figsize=(6,4))
plt.plot(N, maeSliding, label='slinding')
plt.plot(N, maeExpanding, label='expanding')
plt.title('Slinding and expanding windows Comparison Firenze')
plt.legend()
```

```
plt.xlabel('N')
plt.ylabel('mae')
plt.show()

fig = plt.figure(figsize=(6,4))
plt.plot(N, mapeSliding, label='slinding')
plt.plot(N, mapeExpanding, label='expanding')
plt.title('Slinding and expanding windows Comparison Firenze')
plt.legend()
plt.xlabel('N')
plt.ylabel('mape')
plt.show()

trainSamples = 3*7*24 #first 3 week of samples
N = trainSamples
trainToronto, fixedTest = trainTestSplit(hourDfToronto['count'], trainSamples) #set fixed
    test set

mseSliding = []
mseExpanding = []

maeSliding = []
maeExpanding = []

mapeSliding = []
mapeExpanding = []
N = [i for i in range(trainSamples, 1, -trainSamples//(30))] #range(start, stop, step)
for n in N:
  trainToronto, testToronto = trainTestSplit(hourDfToronto['count'], n)
  predsTorontoSliding = walkFowardValidation(trainToronto, fixedTest, bestOrderToronto,
    windowType='sliding')
  predsTorontoExpanding = walkFowardValidation(trainToronto, fixedTest, bestOrderToronto,
    windowType='expanding')

  mseSliding.append(mean_squared_error(fixedTest, predsTorontoSliding))
  mseExpanding.append(mean_squared_error(fixedTest, predsTorontoExpanding))

  maeSliding.append(mean_absolute_error(fixedTest, predsTorontoSliding))
  maeExpanding.append(mean_absolute_error(fixedTest, predsTorontoExpanding))

  mapeSliding.append(mean_absolute_percentage_error(fixedTest, predsTorontoSliding))
  mapeExpanding.append(mean_absolute_percentage_error(fixedTest, predsTorontoExpanding))

fig = plt.figure(figsize=(6,4))
plt.plot(N, mseSliding, label='slinding')
plt.plot(N, mseExpanding, label='expanding')
plt.title('Slinding and expanding windows Comparison Toronto')
plt.legend()
plt.xlabel('N')
plt.ylabel('mse')
plt.show()

fig = plt.figure(figsize=(6,4))
plt.plot(N, maeSliding, label='slinding')
plt.plot(N, maeExpanding, label='expanding')
plt.title('Slinding and expanding windows Comparison Toronto')
plt.legend()
plt.xlabel('N')
plt.ylabel('mae')
plt.show()

fig = plt.figure(figsize=(6,4))
plt.plot(N, mapeSliding, label='slinding')
plt.plot(N, mapeExpanding, label='expanding')
plt.title('Slinding and expanding windows Comparison Toronto')
```

```
plt.legend()
plt.xlabel('N')
plt.ylabel('mape')
plt.show()
```