Politecnico di Torino

Laboratory 2 Report

ICT for Smart Mobility

Group 6

Prediction using ARIMA models

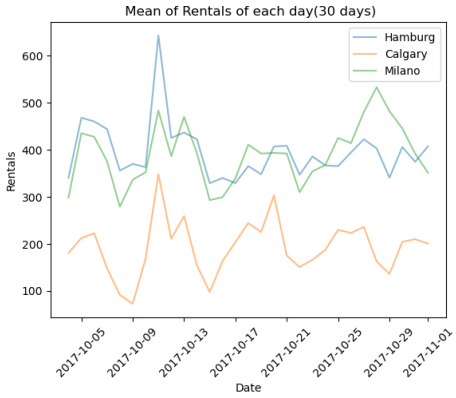
Students:	*Number:*
Shurui Chen	s307957
Abubacarr Ceesay	s310138
Stefano Negri Merlo	s315219

ACADEMIC YEAR 2023/24

Saturday 6th January, 2024

1 Background

The goal of this lab is to train an ARIMA model and use it to predict the number of rentals over a 30 days period.
From the entire dataset at our disposal, we have selected a period with as less missing data as possible and with a stationary behaviour. In particular we are considering the period from 03/10/2917 to 02/11/2017; the figure on the right shows its daily mean. Data right outside this range are too low with respect to its mean, or they are missing.



We have ran many experiments with different parameters and have compared their results by using different kind of errors. The hyperparameters that we have changed include:

- 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.

2 Tasks

2.1 Data extraction

The dataset from the ARIMA model has been constructed by extracting data relative to the three cities of Milano, Hamburg and Calgary during the period mentioned above. Afterwards it has been filtered by removing outliers which rental time is less than 5 min or more than 3 hours and rentals which initial and final position are the same. Finally it has been resampled with a frequency of 1 hour. Table 1 shows the amount of rentals for each city in the dataset.

City	Milano Enjoy	Milano Car2Go	Hamburg Car2Go	Calgary Car2Go
Rentals	734	734	734	721

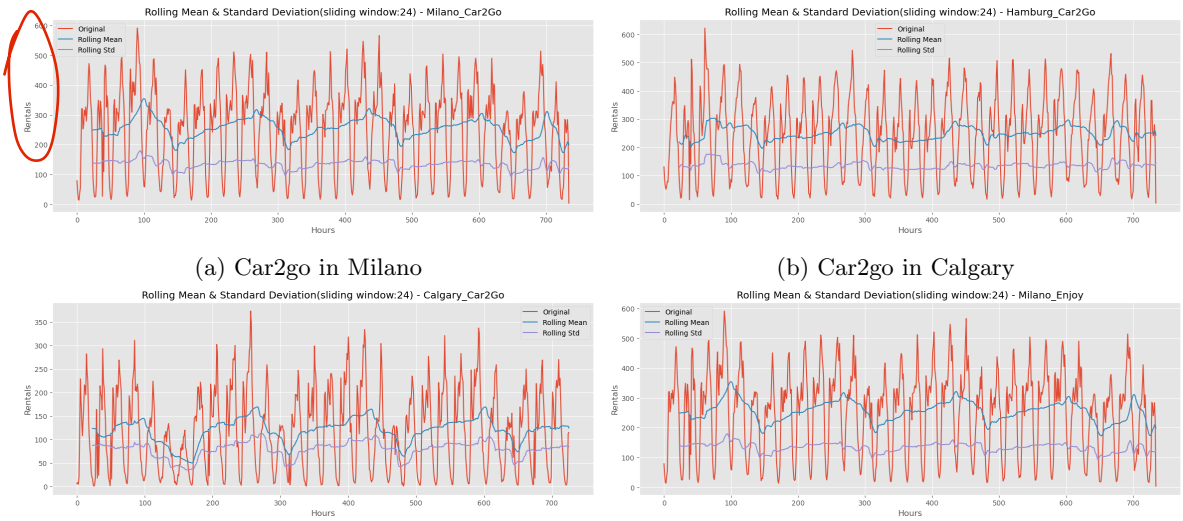
Table 1: Rentals in different cities

2.2 Filling missing data

ARIMA models assume a regular time series with no missing data. We have to check if there are missing samples. Therefore we define a policy for fitting missing data by replace with the average value for the given time bin(previous 24h). It will be more like realistic values because the value changes during the day but similar in the exact hour of each day.

2.3 Checking for stationarity

In order to identify the model to use (ARMA or ARIMA), we have checked the stationarity of the time series. If the model is not stationary we will set the parameter d to a value greater than 0, which decides how many times the process has to be differentiated. Figure 1 shows the mean and standard deviation of the moving average with a windows of 24 hours. The graph shows a periodicity with a length of around one week. Accounting for this cyclical behavior, the mean and the standard deviation remain relatively constant during the period. Following these considerations, we have decided to set $d = 0$.



(a) Car2go in Milano

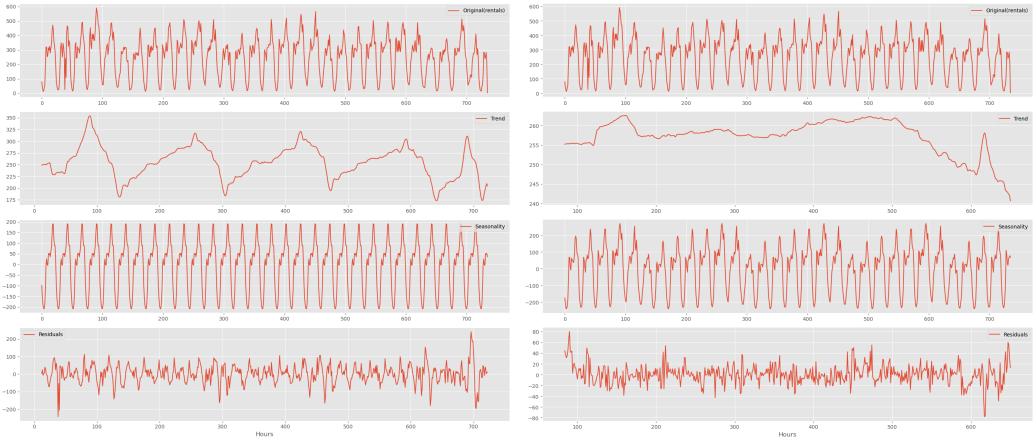
(b) Car2go in Calgary

(c) Car2go in Hamburg

(d) Enjoy in Milano

Figure 1: Rolling Mean & Standard Deviation(sliding window: 24h)

From previous observations we see a cyclical trend in both mean and std, we try to use different sliding windows with 24 hours(1 day) and 168 hours(7 days) to plot the trend decomposition chart. In figure 2, the first picture is the original picture, the second picture is the long-term trend chart, the third picture is the seasonality chart, and the fourth picture is the residual chart. We can see the trend chart is relatively stable and from the seasonality chart that the daily data shows a strong cyclical trend.



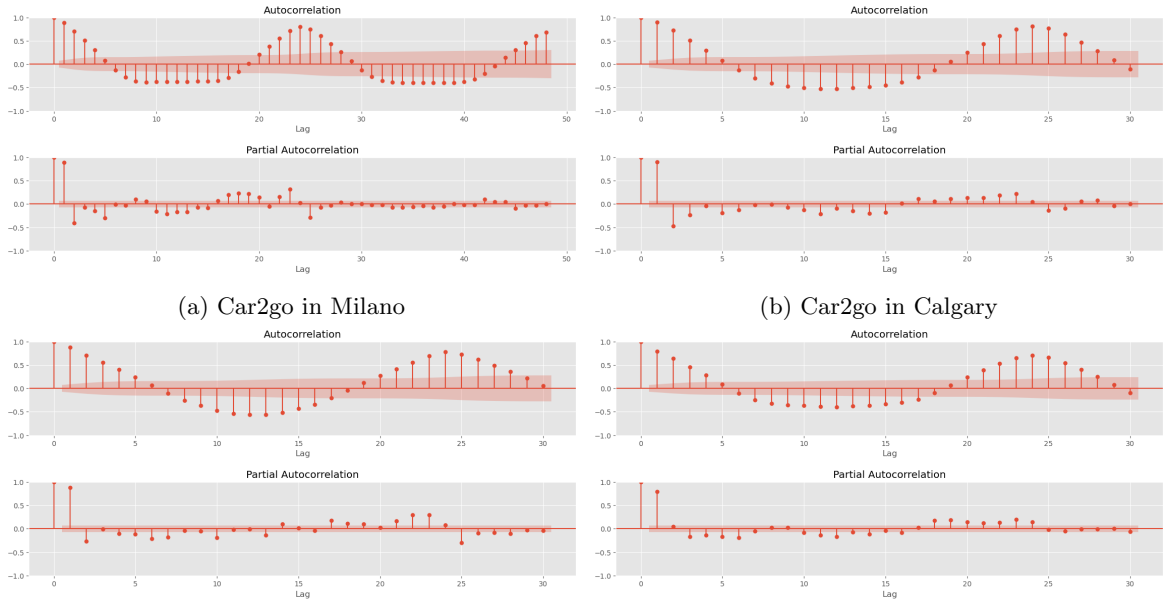
(a) with period: 24h

(b) with period: 168h

Figure 2: Decompose trend, seasonal and residual in Milano Car2Go

2.4 Compute the ACF and PACF. Choose good p and q parameters

Autocorrelation Function (ACF): reflects the correlation between values of the same sequence in different time series, q is the moving average order. Partial Autocorrelation Function (PACF): It is a strict correlation between these two variables (present value and lagged value), eliminating the interference of intermediate random variables. p is the autoregressive order. In figure 3 shows mainly use an ARMA process because $d = 0$. Observing the ACF and PACF, get possible good values of the (p, q) parameters: Milano Car2GO(2,4), Hamburg Car2GO(3,4), Calgary Car2GO(2,5), Milano Enjoy(1,4). From the ACF plots of Car2go in Milano, Calgary and Hamburg, a sine-wave like diagram is observed indicating some seasonality in the time series. Also, a sharp cut-off is observed in the Partial Autocorrelation after a few lags highly indicating that all four observations are AR models.



(a) Car2go in Milano

(b) Car2go in Calgary

(c) Car2go in Hamburg

(d) Enjoy in Milano

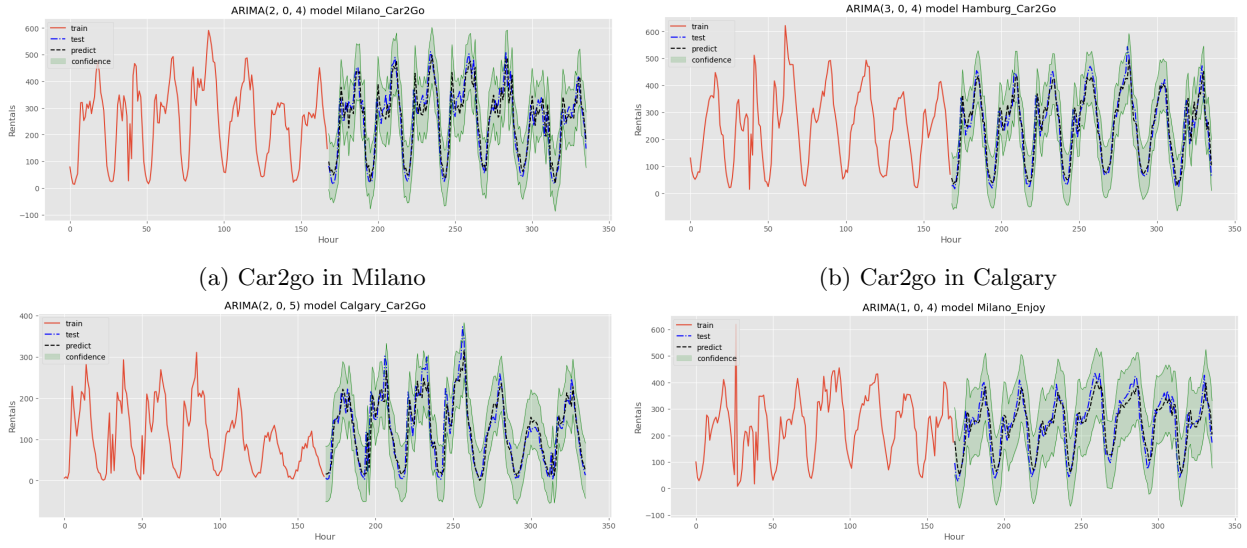
Figure 3: The ACF and PACF in different cities

2.5 Training and Testing

The ARIMA model is trained with a given training set and with given hyperparameters. Its effectiveness is tested by predicting a given test set.

The selected training and the test set correspond respectively to the first and second week of the dataset. Parameters (p, d, q) are the same as those selected in the step above: **Milano Car2GO(2,0,4), Hamburg Car2GO(3,0,4), Calgary Car2GO(2,0,5), Milano Enjoy(1,0,4)**.

The results show in figure 4.



(a) Car2go in Milano

(b) Car2go in Calgary

(c) Car2go in Hamburg

(d) Enjoy in Milano

Figure 4: The ACF and PACF in different cities

The residual is the difference between the actual value and the predicted value. The purpose of the ARIMA model is to minimize the residual error to best fit the data. Once the model has been fitted, we can check if does what we expect and if the assumptions we made are violated. In figure 5, we can observe that the residuals are uncorrelated (bottom right plot) and do not exhibit any obvious seasonality (the top left plot). Also, the residuals and roughly normally distributed with zero mean (top right plot). The Q-Q plot on the bottom left

How can you verify this (?)

shows that the ordered distribution of residuals (dots) roughly follows the linear trend of samples taken from a standard normal distribution with N(0, 1).

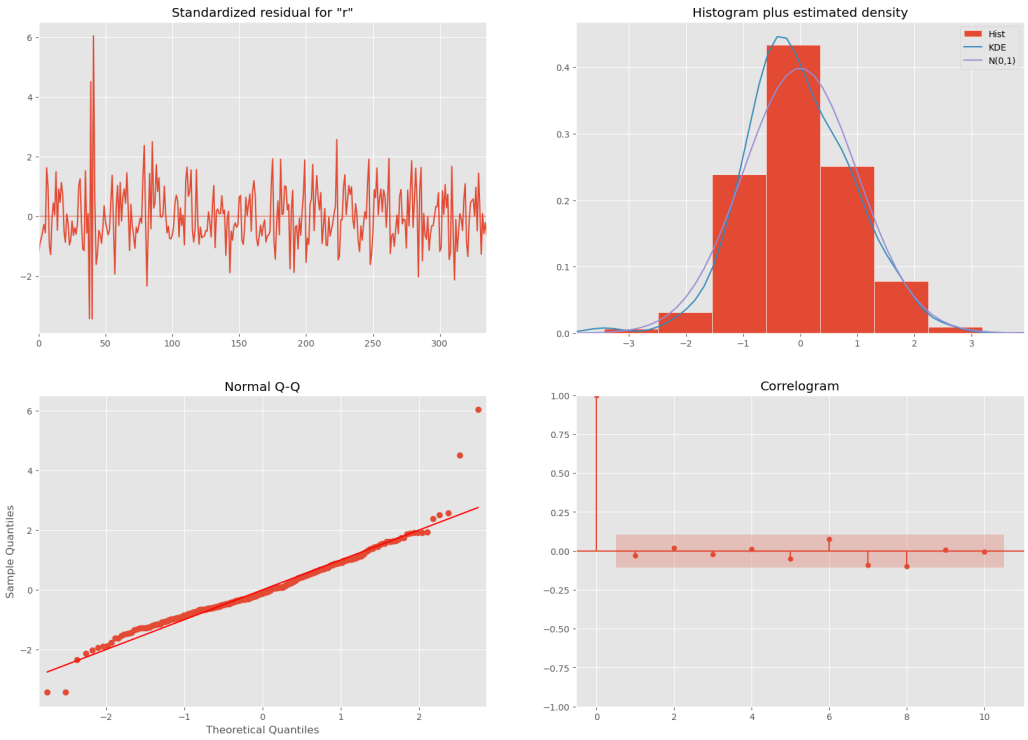


Figure 5: Diagnostic model residual distribution for Milano Car2Go

2.6 Computing the MAE, $MAPE$, MSE and R^2

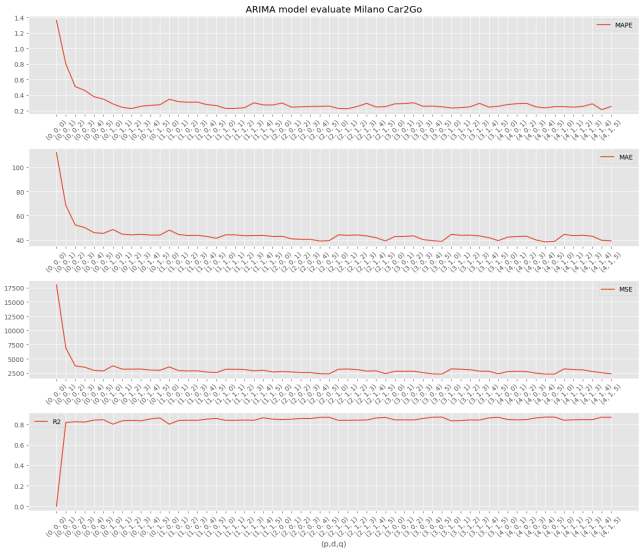
We take the same $N, (p, d, q)$ as previous task to compute the metrics after training and testing the model. Table 2 shows MAE, $MAPE$, MSE and R^2. We can see that the estimated parameters values perform good in Hamburg Car2Go which can also be proved in next steps.

Metrics	MAE	MAPE	MSE	R^2
Milano Car2Go	39.156	0.255	2377.250	0.868
Hamburg Car2Go	31.131	0.199	1651.198	0.907
Calgary Car2Go	22.710	0.528	1091.580	0.847
Milano Enjoy	35.498	0.222	2009.172	0.830

Table 2: Metrics computing

2.7 Check the impact of parameters:

- Keep $N = 168$ (7 days) fixed, and do a grid search varying (p, d, q) and observe how the error changes. Varying parameter p in range $[0, 4]$, d in range $[0, 1]$ and q in range $[0, 5]$. Choose the best (p, d, q) parameter tuple for each city.



better use tables than plots. Also this is not a "curve" but rather separate points

Figure 6: Evaluate different parameters (p,d,q) for MAE,MAPE,MSE,R^2.

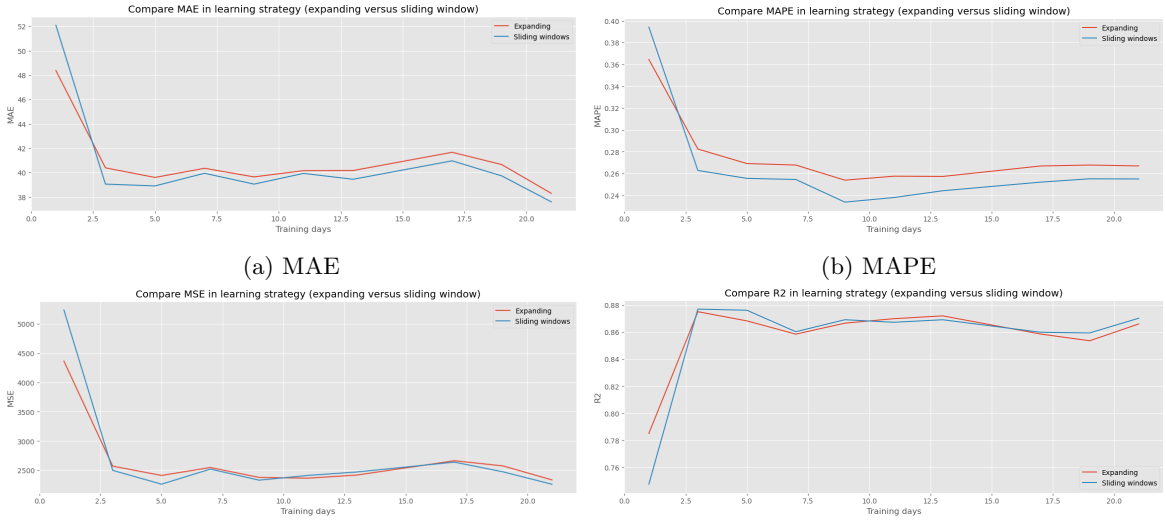
Table 3 shows the best ARIMA orders by varying different orders and comparing the error changes.

Best ARIMA (p,d,q)	MAE	MAPE	MSE	R^2
Milano Car2Go	(4, 0, 4)	(4, 1, 4)	(4, 0, 5)	(4, 0, 5)
Hamburg Car2Go	(3, 0, 4)	(4, 1, 4)	(3, 0, 4)	(3, 0, 4)
Calgary Car2Go	(3, 0, 5)	(4, 0, 5)	(3, 0, 5)	(3, 0, 5)
Milano Enjoy	(3, 0, 2)	(3, 0, 2)	(3, 0, 2)	(3, 0, 2)

Table 3: Finding best ARIMA order by comparing metrics

which one do you choose (?)

- Given the best parameter configuration, change N training samples and the learning strategy (expanding versus sliding window). Always keep the testing set on the same portion of the data. We decide to use 1 week for testing and $N = [1, 3, 5, 7, 9, 11, 13, 17, 19, 21]$ days for training. Typically, when a new data is obtained, the model is retrained to make predictions, which overall gives better predictions at each timestamp. Therefore, we need to compare the two learning strategies to retrain predictions at each timestamp and record the error for model evaluation. Figure 7 shows that the sliding window works a little bit better than expanding window, but this learning strategy may have problems in multi-step prediction (When use a larger order). (?) why (?)



(a) MAE (b) MAPE

(c) MSE (d) R^2

Figure 7: Compare metrics in different learning strategies (expanding versus sliding window)

not clear what the plot reports. what is the take away (?) why you do this analysis (?)

- The change of the relative error w.r.t. the absolute number of rentals is shown in Figure 8. It presents a distribution similar to a decreasing exponential. As the absolute number of rentals increases, the relative error decreases. All cities follow the same behaviour.

  The graph has been obtained by calculating the error for each predicted sample as $abs(predicted - real)/real * 100$. Real values have been gathered in bins for a better evaluation. Errors corresponding to the same bin have been grouped together and replaced with their mean.

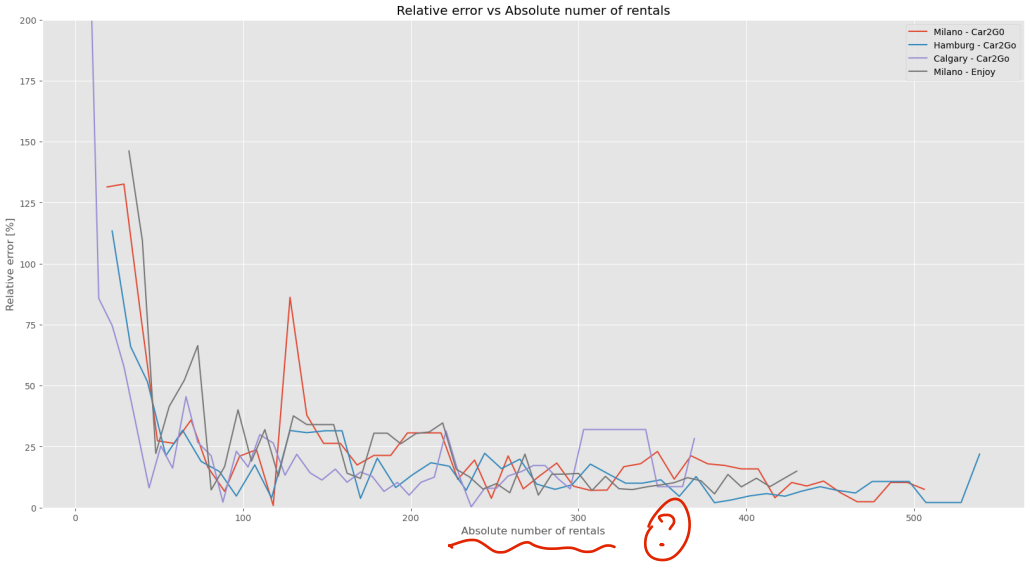


Figure 8: Change of the relative error w.r.t the absolute number of rentals

2.8 Optional Task: time horizon h of the prediction impacts the performance

Use the model to predict the future rentals at t+h, h in [1:24] or more. The number of test samples and predictions are in 168 hours (1 week). Considering previous tasks, we try to use order (2,0,4) for ARIMA model, while the forecast time horizon changes each time (h=[1,24]).
Figure 9 clearly shows that the further in time the prediction goes, the worse is the performance which the predictions are around 250 rentals. This can be seen also from Figure 10: ARIMA model is trained on all the samples of the time series and then forecast 2 future weeks with a confidence interval of 90%. In conclusion, The uncertainty associated to each prediction proportionally increases with the time step of the forecast and the predicted value converges to the mean value of the time series. ARIMA models perform well only at forecasting the next few time horizon, but are less effective at long-term forecasts.

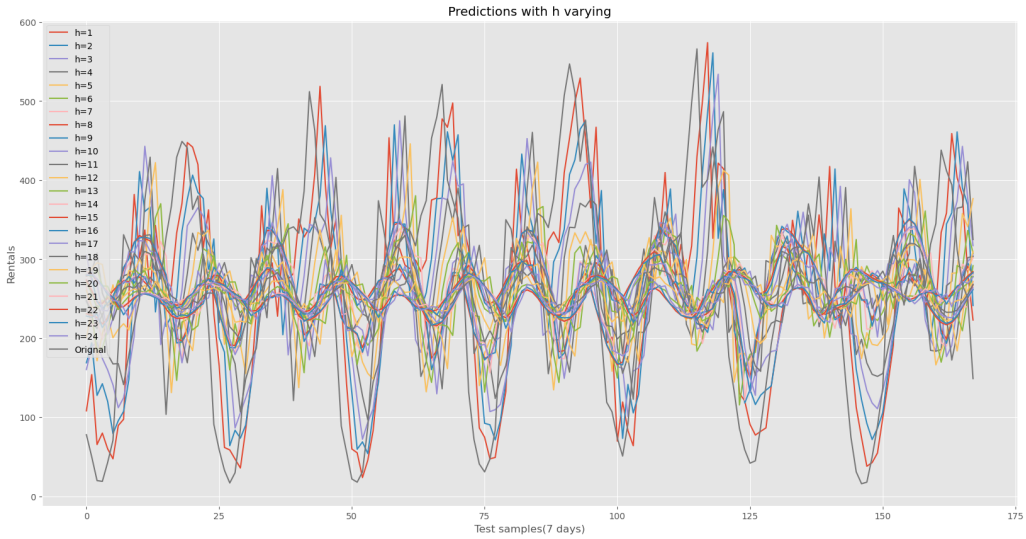


Figure 9: Comparison the predictions of rentals in Milano with varying time horizon.

- complete but some analysis are not clear
- missing the comparison with a baseline model

5 - /5

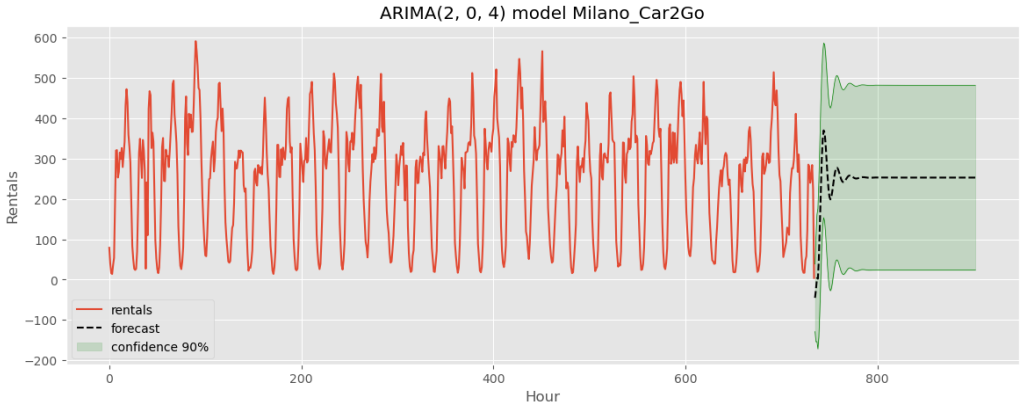


Figure 10: Prediction of future samples with 95% confidence interval.

3 Appendice Codes

In [5]:
```
company = "Car2Go"
rental = "Bookings"
timezones = {
    'Hamburg': pytz.timezone("Europe/Berlin"),
    'Calgary': pytz.timezone("Canada/Mountain"),
    'Milano': pytz.timezone("Europe/Rome"),
}
company_prefix = {
    "Car2Go": "",
    "Enjoy": "enjoy_"
}
```

In [6]:
```
# Connect
client = pm.MongoClient(
    host='bigdatadb.polito.it',
    ssl=True,
    authSource='carsharing',
    username='ictts',
    password='Ict4SM22!',
    tlsAllowInvalidCertificates=True,
)
db = client["carsharing"]
collection = db[f"{company_prefix[company]}Permanent{rental}"]

Car2goPermanentBook = db['PermanentBookings']
Car2goPermanentPark = db['PermanentParkings']
Car2goActiveBook = db['ActiveBookings']
Car2goActivePark = db['ActiveParkings']

EnjoyPermanentBook = db['enjoy_PermanentBookings']
EnjoyPermanentPark = db['enjoy_PermanentParkings']
EnjoyActiveBook = db['enjoy_ActiveBookings']
EnjoyActivePark = db['enjoy_ActiveBookings']
```

In [7]:
```
city = ['Hamburg','Calgary','Milano']
```

In [8]:
```
def get_all(city):
    data = list(collection.find({
        "city": city,
    }, ("city", "init_date", "final_date")))
    return DataFrame(data)
```

In [9]:
```
city_data = {
    city: get_all(city)
    for city in timezones.keys()
}
```

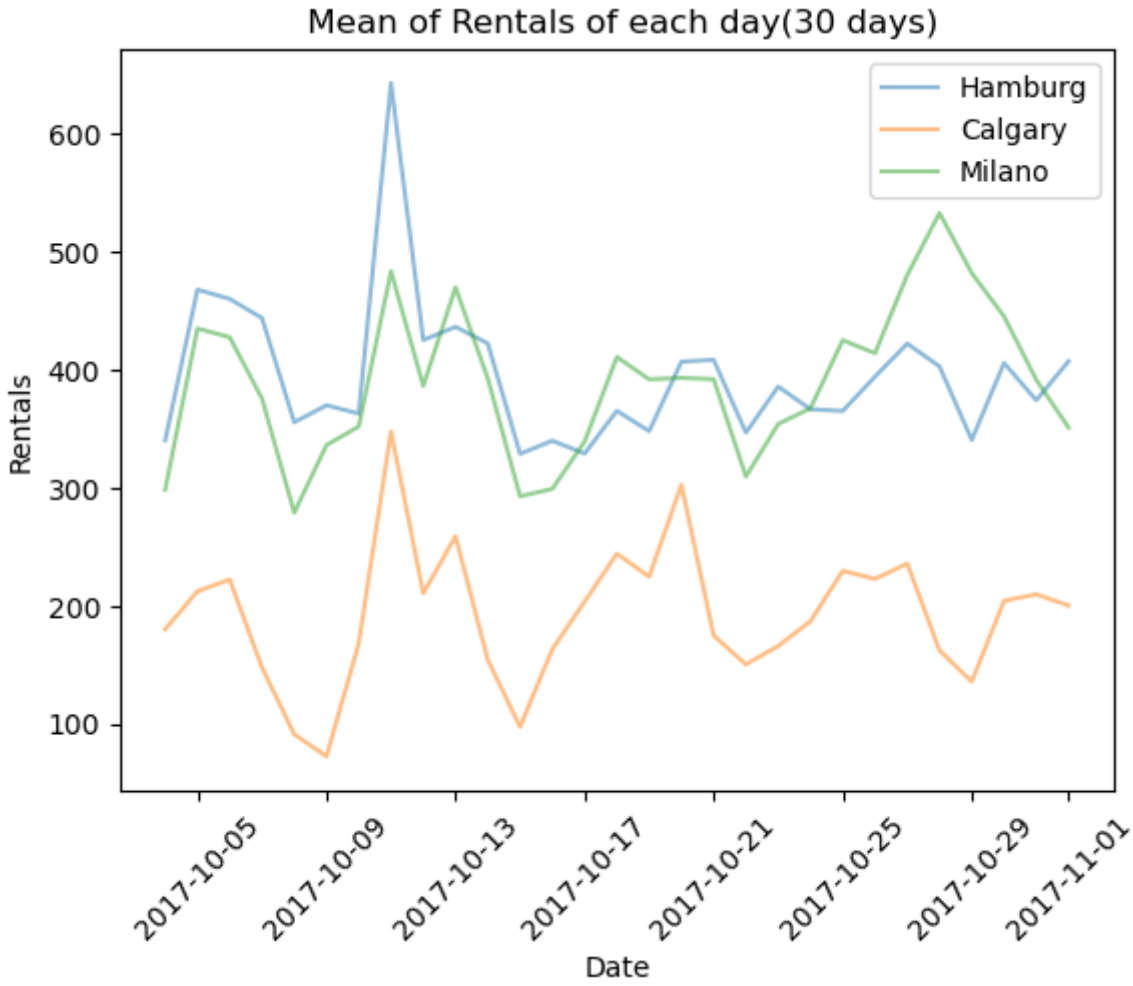
In [10]:
```
copy_data = {
    city: df.copy()
    for city, df in city_data.items()
}
```

In [11]:
```
resampled = {
    city: df.set_index("init_date").resample('1H').count()
    for city, df in copy_data.items()
}
```

In [12]:
```
rolling = {
    city: df.resample('1D').mean()
    for city, df in resampled.items()
}
```

In [13]:
```
start = datetime.datetime(2017, 10, 3)
month = datetime.timedelta(days=30)
choice = {
    city: df[(start < df.index) & (df.index < start + month)]
    for city, df in rolling.items()
}
plt.xticks(rotation=45)
i=0
for df in choice.values():
    plt.plot(df.index, df["final_date"],label= city[i], alpha=0.5)
    i+=1
plt.xlabel('Date')
plt.ylabel('Rentals')
plt.title(' Mean of Rentals of each day(30 days)')
plt.legend()
# 6, 10
```

Out[13]:
```
<matplotlib.legend.Legend at 0x7f1b0cd96350>
```



Data preprocessing

In [14]:
```
#2017.10.3-2017.11.2
Start = time.mktime((datetime.datetime(2017, 10, 3, 0, 0)).timetuple())
End = time.mktime((datetime.datetime(2017, 11, 3, 0, 0)).timetuple())
```

In [15]:
```
#filter
def pipeline_rental(city,zone):
    addTime = zone*60*60
    pipeline = [{"$match":
                    {"$and": [{"city": city},
                              {"init_time": {"$gte": Start + addTime}},
                              {"init_time": {"$lte": End + addTime}}
                              ]}
                }, {"$project": {
                    "_id": 0,
                    "durationBook": {"$divide": [{"$subtract": ["$final_time
                    "hour": {"$floor": {"$divide": ["$init_time", 3600]}},
                    "moved": {"$ne": [{"$arrayElemAt": ["$origin_destinatior
                                       {"$arrayElemAt": ["$origin_destinatior
                    }
                },{"$match":
                          {"$and": [{"moved": True},
                                    {"durationBook": {"$gte": 5}},
                                    {"durationBook": {"$lte": 180}}
                                    ]}
                },
                {"$group": {
```

```
                    "_id":  "$hour",
                    "totalBooking": {"$sum": 1}}
                },{"$sort": {
                    "_id.hour": 1
                    }
                }
            ]
    return pipeline
```

In [16]:
```
for city in city:

    if city == 'Milano':
        pipeline_Milano = pipeline_rental(city,-1)
        Milano_Car2Go_rentals = Car2goPermanentBook.aggregate(pipeline_Milar
        Milano_Enjoy_rentals = EnjoyPermanentBook.aggregate(pipeline_Milano)
        list_Milano_Car2Go = list(Milano_Car2Go_rentals)
        list_Milano_Enjoy = list(Milano_Enjoy_rentals)

    if city == 'Hamburg':
        pipeline_Hamburg_car2go = pipeline_rental(city,-1)
        Hamburg_Car2Go_rentals = Car2goPermanentBook.aggregate(pipeline_Hamb
        list_Hamburg_Car2Go = list(Hamburg_Car2Go_rentals)

    if city == 'Calgary':
        pipeline_Calgary_car2go = pipeline_rental(city,7)
        Calgary_Car2Go_rentals = Car2goPermanentBook.aggregate(pipeline_Calg
        list_Calgary_Car2Go = list(Calgary_Car2Go_rentals)
```

In [17]:
```
print("the rentals of Milano, Enjoy, Hamburg, Calgary:", len(list_Milano_Car
```

```
the rentals of Milano, Enjoy, Hamburg, Calgary: 734 734 734 721
```

In [18]:
```
#define a policy for fitting missing data: using the previous hour data
def MissingData_Processing(inputdata,city):

    df = pd.DataFrame(inputdata)
    df.columns = ['hour', 'rentals']
    df = df.sort_values(by=['hour'])
    for i in range(len(df)-1):
        Hour, NextHour = df.iloc[i]['hour'], df.iloc[i+1]['hour']
        if Hour != NextHour - 1:
            sum = 0
            for j in range(24):
                sum = sum + df.iloc[i-j]['rentals']
            new_item = {'hour': Hour + 1, 'rentals': sum//24}
            df = pd.concat([df.iloc[:i+1], pd.DataFrame([new_item]), df.iloc

    df.to_csv(city + '.csv', index=False)
```

In [19]:
```
Milano_Car2Go = MissingData_Processing(list_Milano_Car2Go, 'Milano_Car2Go')
Milano_Enjoy = MissingData_Processing(list_Milano_Enjoy, 'Milano_Enjoy')
Hamburg_Car2Go = MissingData_Processing(list_Hamburg_Car2Go, 'Hamburg_Car2Go
Calgary_Car2Go = MissingData_Processing(list_Calgary_Car2Go, 'Calgary_Car2Go
```

Stationary check

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

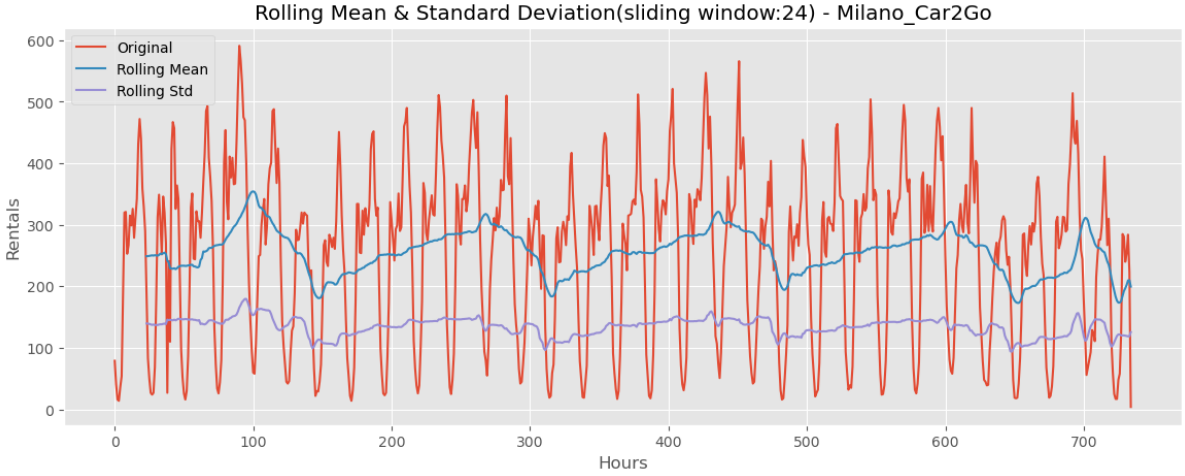
Recall the definition, if the mean or variance changes over time then it's non-stationary, thus an intuitive way is to plot rolling statistics. We can also make an autocorrelation plot, as non-stationary time series often shows very slow decay

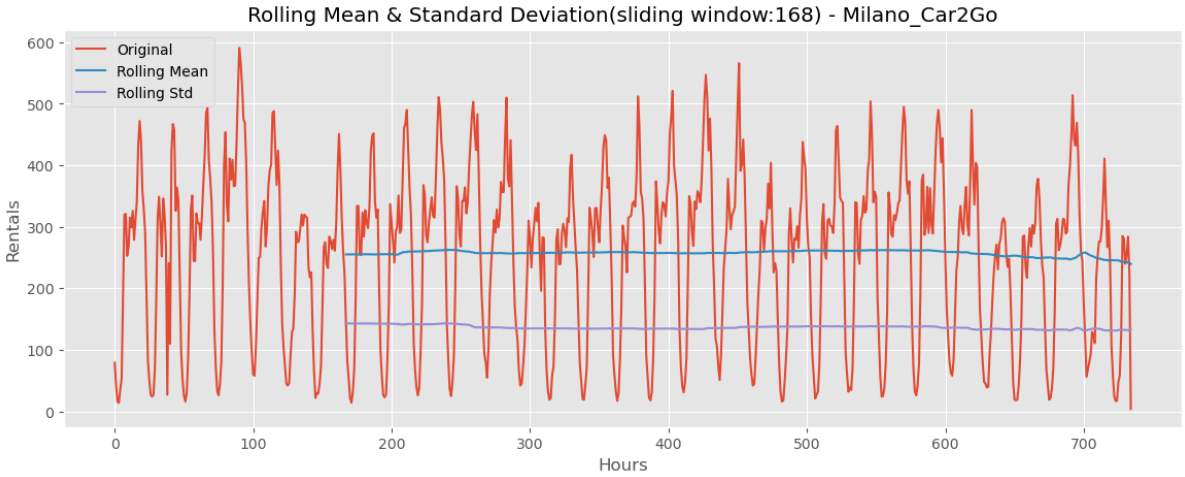
In [20]:
```
# Plot style Defaults
plt.rcParams['figure.figsize'] = (20.0, 10.0)
plt.rcParams.update({'font.size': 12})
plt.style.use('ggplot')
```

In [21]:
```
import warnings
from statsmodels.tsa.arima.model import ARIMA  #构建ARIMA模型时候会使用的包
from statsmodels.tsa.seasonal import seasonal_decompose
from statsmodels.tsa.stattools import adfuller #平稳性检验，单位根检验
from statsmodels.graphics.tsaplots import plot_acf, plot_pacf  #求取自相关系数
```

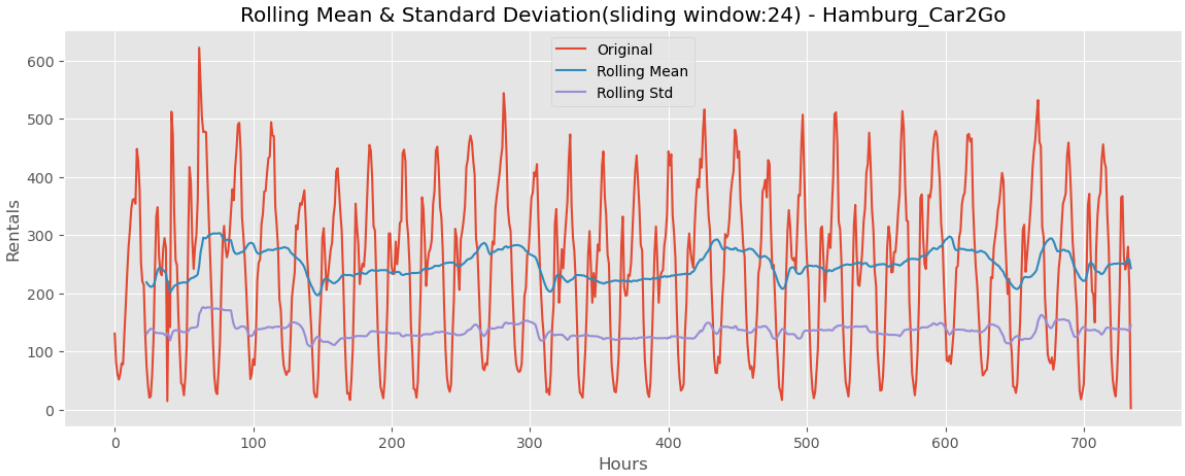
In [22]:
```
def draw_trend(data, size, city):
    '''
    time series trend, size: average moving 绘制原始趋势及移动平均的水平和波动
    '''
    value = data['rentals']
    f = plt.figure(facecolor='white',figsize=(14,5))
    # mean
    rol_mean = value.rolling(window=size).mean()
    # standard diviation
    rol_std = value.rolling(window=size).std()

    value.plot(label='Original')
    rol_mean.plot(label='Rolling Mean')
    rol_std.plot(label='Rolling Std')
    plt.xlabel('Hours')
    plt.ylabel('Rentals')
    plt.legend(loc='best')
    plt.title('Rolling Mean & Standard Deviation(sliding window:'+str(size)+
    plt.show()
```

In [23]:
```
df_Milano_Car2Go = pd.read_csv('Milano_Car2Go.csv')
draw_trend(df_Milano_Car2Go,24,'Milano_Car2Go')
draw_trend(df_Milano_Car2Go,168,'Milano_Car2Go')
```

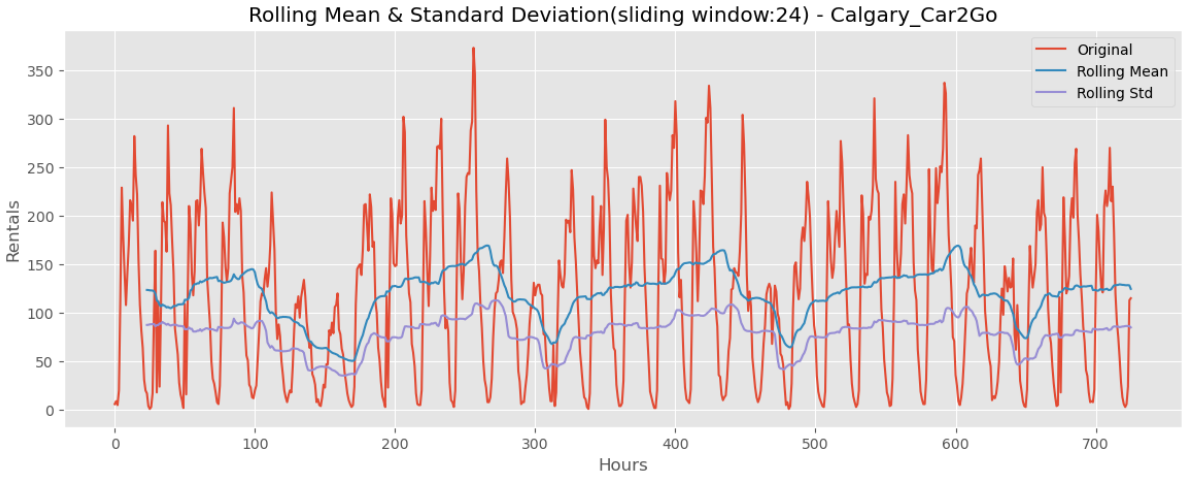




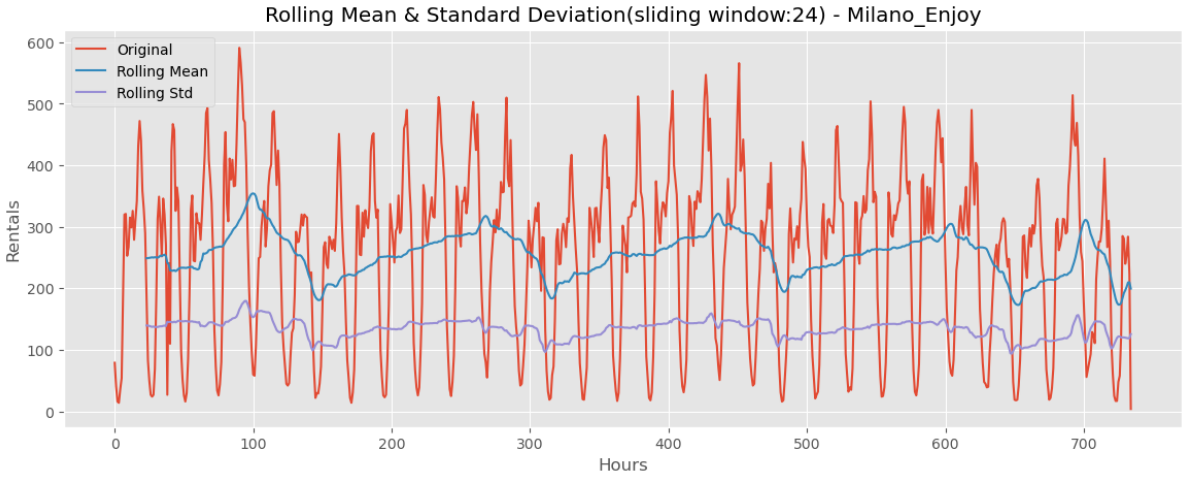
In [24]:
```
df_Hamburg_Car2Go = pd.read_csv('Hamburg_Car2Go.csv')
draw_trend(df_Hamburg_Car2Go,24,'Hamburg_Car2Go')
```



In [25]:
```
df_Calgary_Car2Go = pd.read_csv('Calgary_Car2Go.csv')
draw_trend(df_Calgary_Car2Go,24, 'Calgary_Car2Go')
```

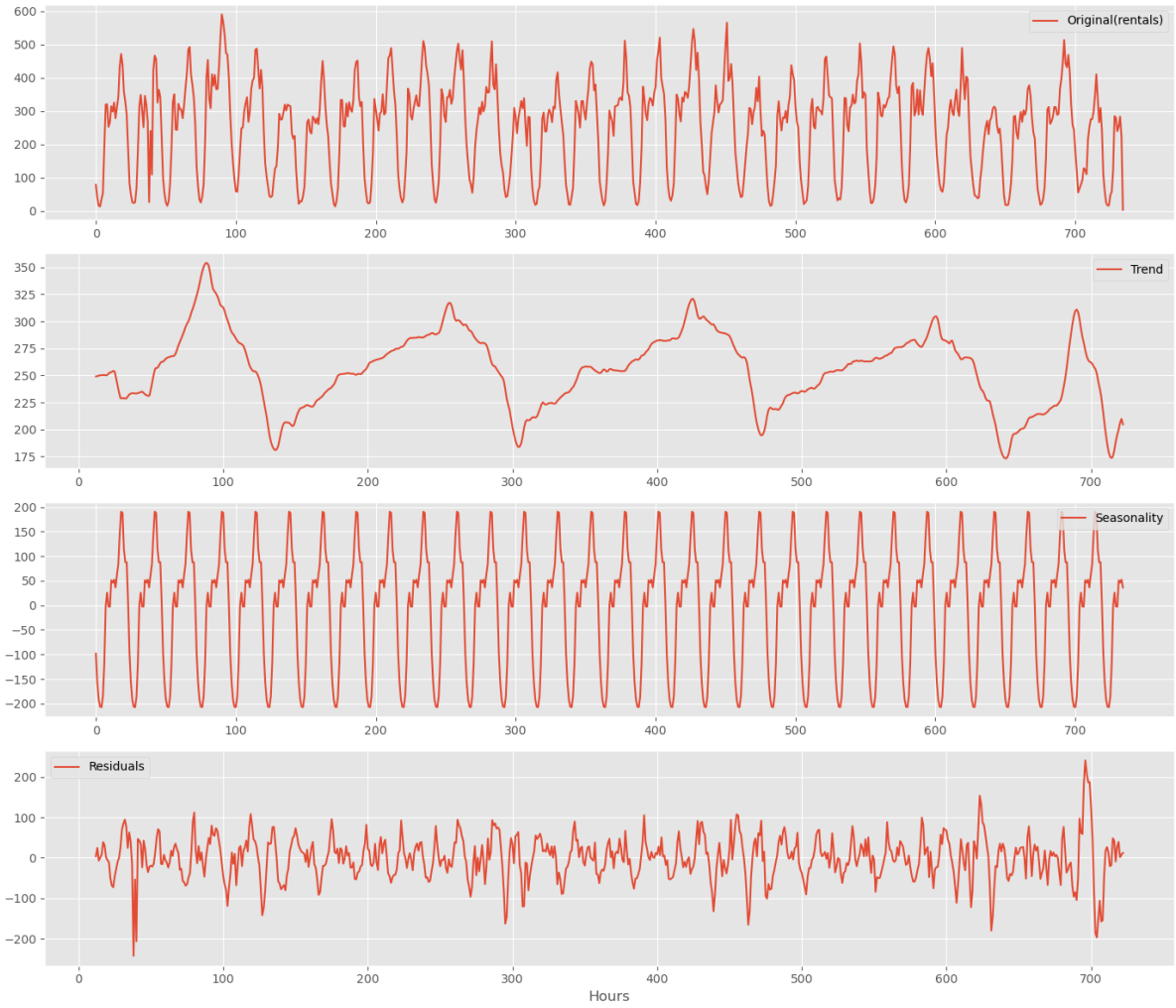


In [26]:
```
df_Milano_Enjoy = pd.read_csv('Milano_Enjoy.csv')
draw_trend(df_Milano_Car2Go,24,'Milano_Enjoy')
```



In [27]:
```
def decompose(data,p):
    '''
    时间序列趋势分解的函数，timeseries是所需要分析的时序数据,p是需要确认的周期性的期数
    '''
    value = data['rentals']
    # including trend, seasonal and residual
    decomposition = seasonal_decompose(value,period=p)

    trend = decomposition.trend
    seasonal = decomposition.seasonal
    residual = decomposition.resid
    plt.figure(facecolor='white',figsize=(14,12))
    plt.subplot(411)
    plt.plot(value, label='Original({})'.format(value.name))
    plt.legend(loc='best')
    plt.subplot(412)
    plt.plot(trend, label='Trend')
    plt.legend(loc='best')
    plt.subplot(413)
    plt.plot(seasonal,label='Seasonality')
    plt.legend(loc='best')
    plt.subplot(414)
    plt.plot(residual, label='Residuals')
    plt.legend(loc='best')
    plt.xlabel('Hours')
    plt.tight_layout()
    plt.show()
    return trend , seasonal, residual
```

In [28]:
```
decompose(df_Milano_Car2Go,p=24)
```


Original(rentals)
Trend
Seasonality
Residuals
Hours

Out[28]:
```
(0     NaN
 1     NaN
 2     NaN
 3     NaN
 4     NaN
       ..
 730   NaN
 731   NaN
 732   NaN
 733   NaN
 734   NaN
 Name: trend, Length: 735, dtype: float64,
 0      -98.438965
 1     -157.323687
 2     -191.961187
 3     -206.406613
 4     -207.264372
          ...
 730     -2.866383
 731     50.976289
 732     46.220063
 733     52.186035
 734     36.299924
 Name: seasonal, Length: 735, dtype: float64,
 0     NaN
 1     NaN
 2     NaN
 3     NaN
 4     NaN
       ..
 730   NaN
 731   NaN
 732   NaN
 733   NaN
 734   NaN
 Name: resid, Length: 735, dtype: float64)
```

In [29]:
```python
def teststationarity(ts):
    '''
    原假设是不平稳
    '''
    dftest = adfuller(ts)
    ##原假设是不平稳
    # 对上述函数求得的值进行语义描述
    dfoutput = pd.Series(dftest[0:4], index=['Test Statistic','p-value','#La
    for key,value in dftest[4].items():
        dfoutput['Critical Value (%s)'%key] = value
    return dfoutput


print('------ Original data stationarity test ----------------')
print(teststationarity(df_Milano_Car2Go['rentals']))

print('------ Differenced data stationarity test ----------------')
teststationarity(df_Milano_Car2Go['rentals'].diff(1).dropna())

#当p-value>0.05，则要进行差分运算！否之直接跳转到模型选择与定阶。
```

```
------ Original data stationarity test ----------
Test Statistic                  -3.518532
p-value                          0.007524
#Lags Used                      19.000000
Number of Observations Used    715.000000
Critical Value (1%)             -3.439529
Critical Value (5%)             -2.865591
Critical Value (10%)            -2.568927
dtype: float64
------ Differenced data stationarity test ----------
```

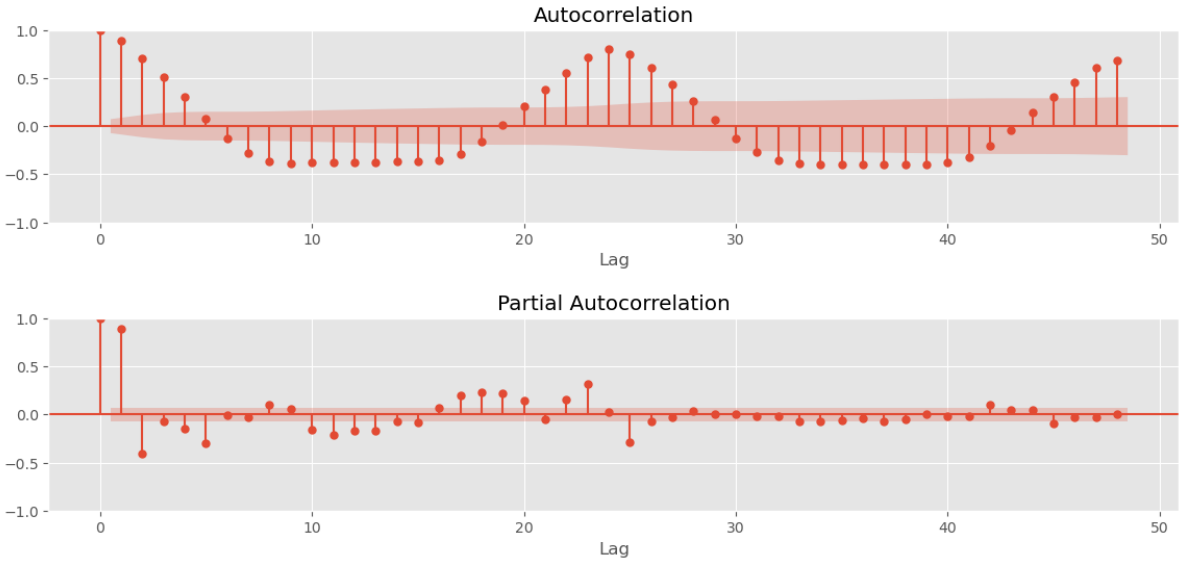
Out[29]:
```
Test Statistic                -1.493357e+01
p-value                        1.355010e-27
#Lags Used                     2.000000e+01
Number of Observations Used    7.130000e+02
Critical Value (1%)           -3.439555e+00
Critical Value (5%)           -2.865602e+00
Critical Value (10%)          -2.568933e+00
dtype: float64
```

ACF & PACF

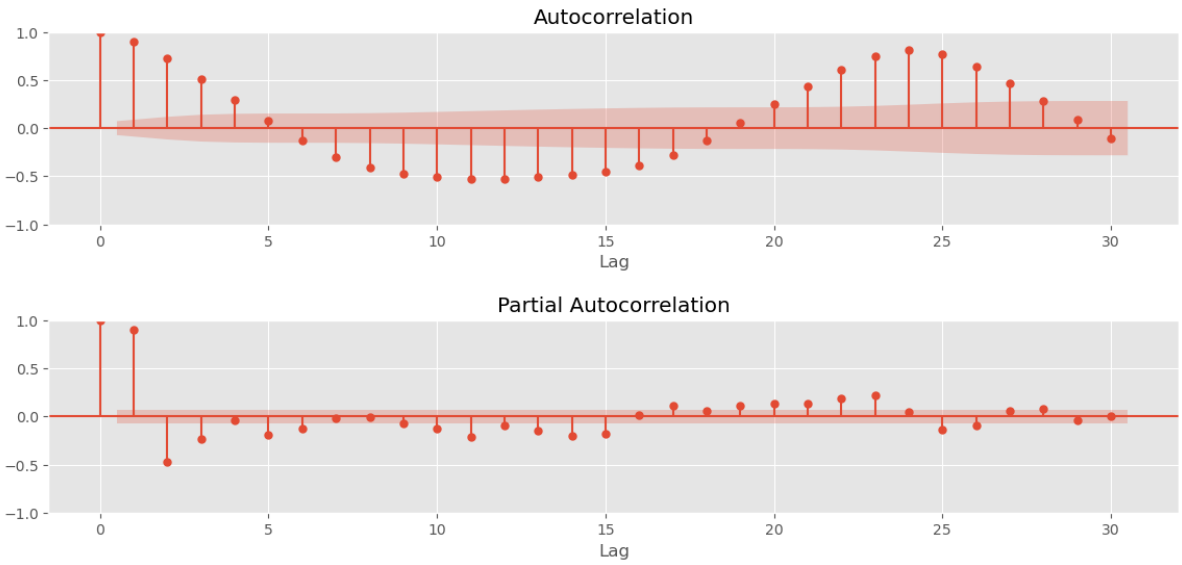
Step 4 ACF&PACF to select p & q

In [130…
```
def draw_acf_pacf(ts,lags):
    f = plt.figure(facecolor='white',figsize=(14,6))
    ax1 = f.add_subplot(211)
    plot_acf(ts,ax=ax1,lags=lags)
    plt.xlabel('Lag')
    ax2 = f.add_subplot(212)
    plot_pacf(ts,ax=ax2,lags=lags)
    plt.xlabel('Lag')
    plt.subplots_adjust(hspace=0.5)
    plt.show()

    # import statsmodels.tsa.stattools as st
    # model = st.arma_order_select_ic(ts, max_ar=5, max_ma=5, ic=['aic', 'bi
    # print('BIC to get (p,q):' + str(model.bic_min_order))
```

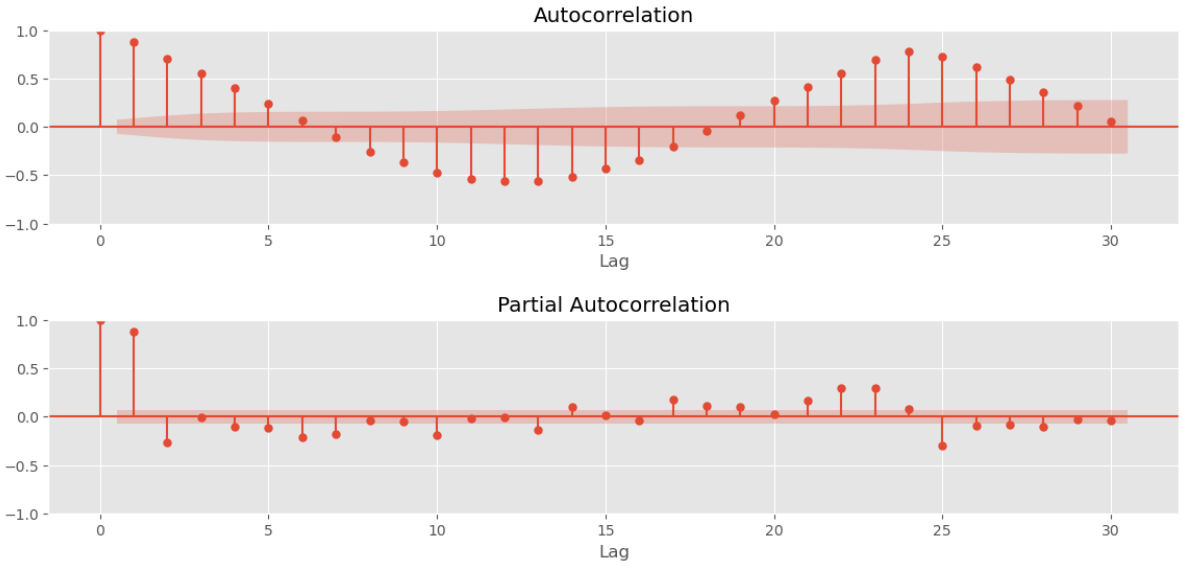
In [131…
```
draw_acf_pacf(df_Milano_Car2Go['rentals'],48)
```



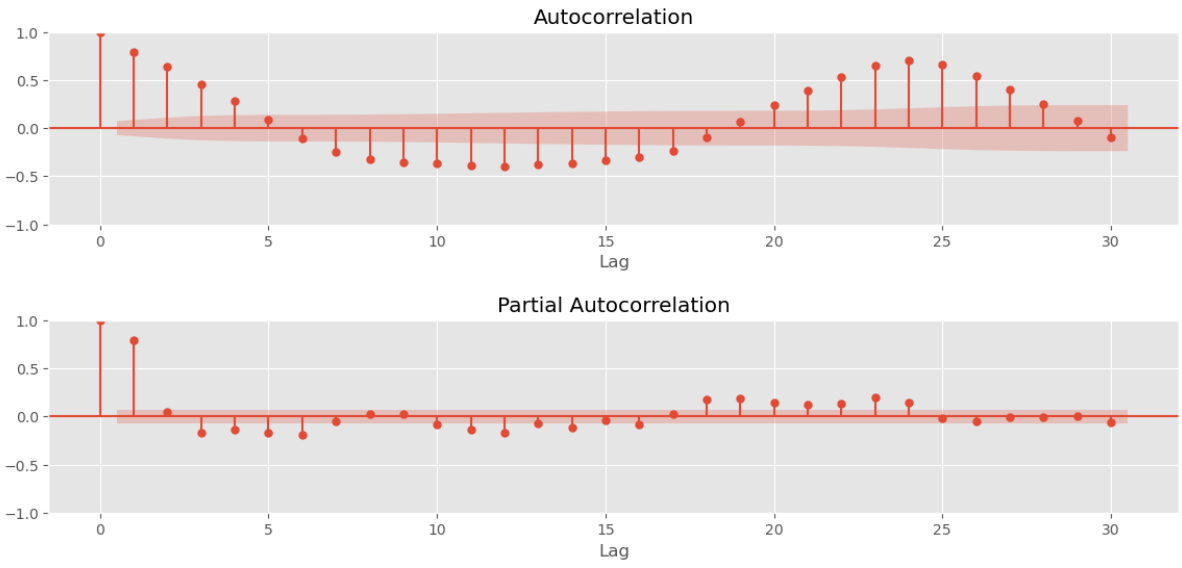
In [132…
```
draw_acf_pacf(df_Hamburg_Car2Go['rentals'],30)
```



In [133…
```
draw_acf_pacf(df_Calgary_Car2Go['rentals'],30)
```



In [134…
```
draw_acf_pacf(df_Milano_Enjoy['rentals'],30)
```



From ACF&PACF: ARMA Milano(2,4) Hamburg(3,4) Calgary(2,5) MilanoEnjoy(1,4)

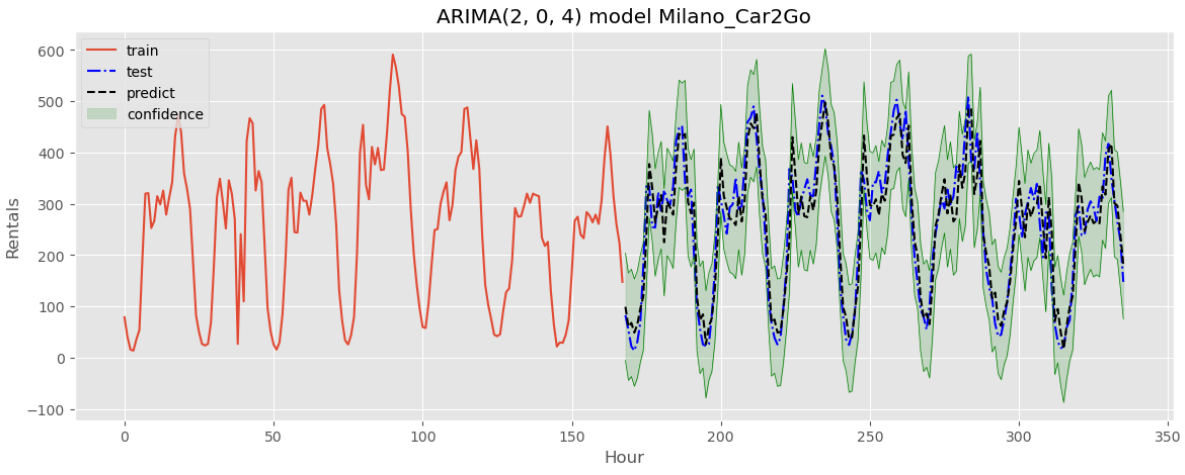
Step 5 Model fitting and testing

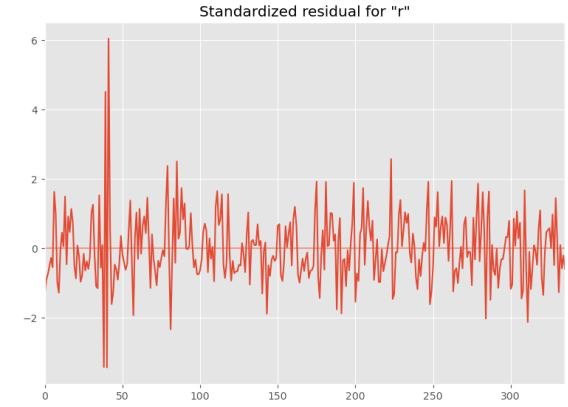
In [136…

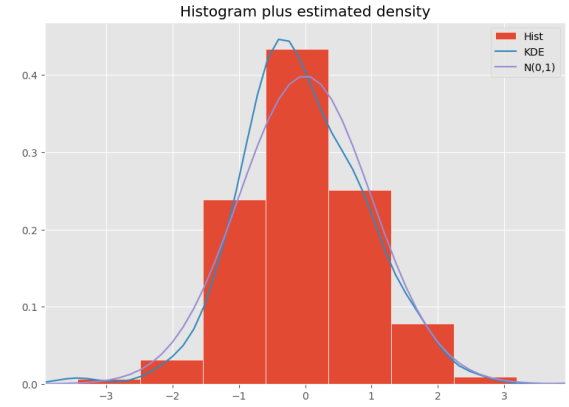
```
# Arima N:1 week
def model_fitting(data,city,N,order):
    n=N*7*24
    TrainData=data.iloc[:n]
    TestData=data.iloc[n:2*n]
    ARIMA_model = ARIMA(data['rentals'][:2*n], order=order)  #ARIMA model
    ARIMA_result = ARIMA_model.fit()
    predict_df = ARIMA_result.get_prediction(n) #predict(start=n,end=2*n,dyr
    confident = predict_df.conf_int(alpha=0.05)
    result=predict_df.predicted_mean
    fig=plt.figure(figsize=(14,5))
    plt.plot(TrainData['rentals'].index,TrainData['rentals'],label='train')
    plt.plot(TestData['rentals'].index,TestData['rentals'],'-.b',label='test
    plt.plot(result.index, result,'--k',label='predict')
    plt.plot(confident.index, confident['lower rentals'], 'g',marker='.',mar
    plt.plot(confident.index, confident['upper rentals'], 'g',marker='.',mar
    plt.fill_between(confident.index,confident['lower rentals'],confident['u
    plt.ylabel('Rentals')
    plt.xlabel('Hour')
    plt.title('ARIMA'+str(order)+' model '+city)
    plt.legend()

    ARIMA_result.plot_diagnostics(figsize=(20, 14))
    plt.show()
    # error = pd.DataFrame(ARIMA_result.resid).iloc[n:2*n]
    # error.plot(legend=False)
    # plt.title('ARIMA residual error for '+city)
    # plt.xlabel('Hour')

    #print(mape(TestData['rentals'],result))
    return TestData,result
```

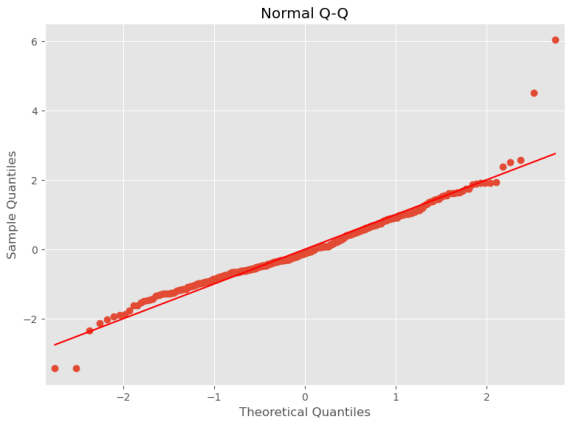
In [120…

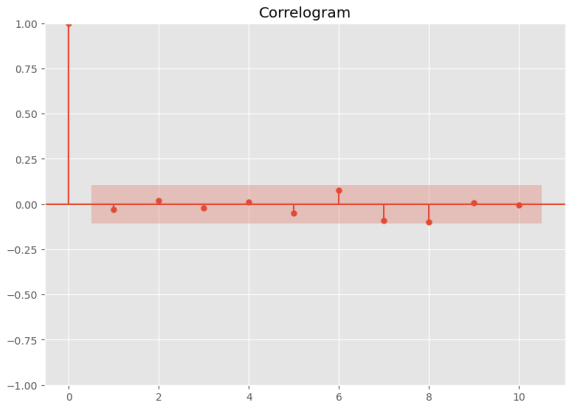
```
TestData_Milano_Car2Go,predict_Milano_Car2Go = model_fitting(df_Milano_Car2C
```







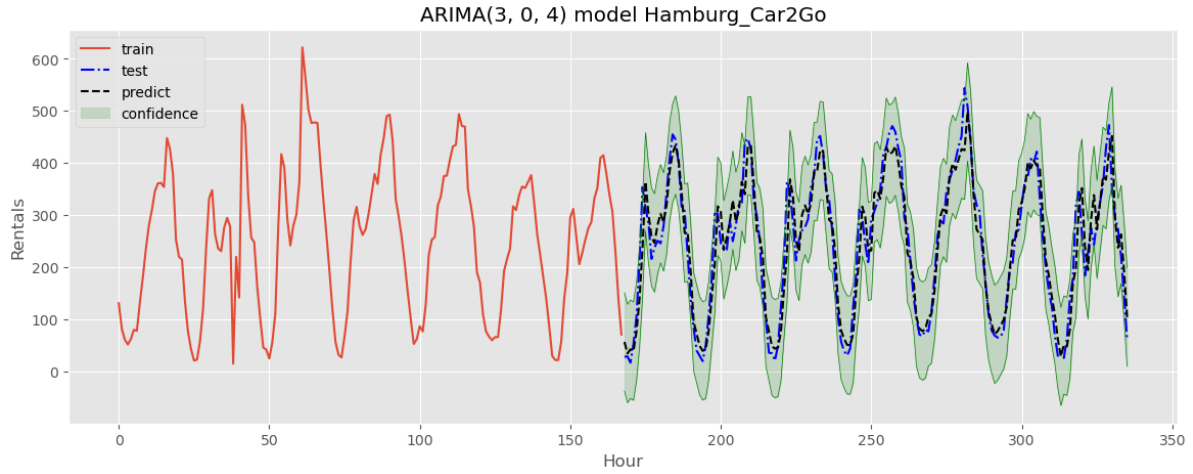


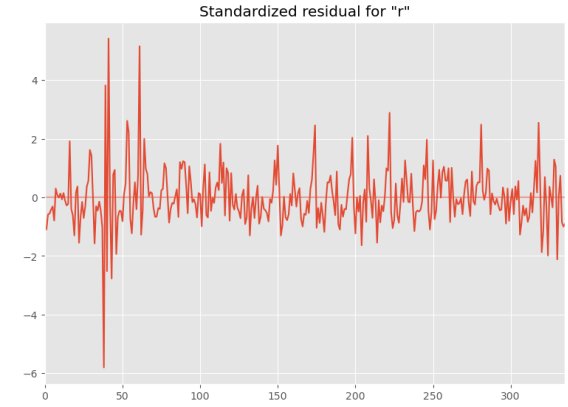


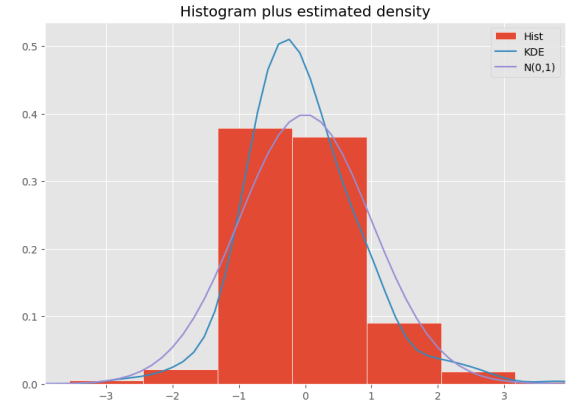
In [121…

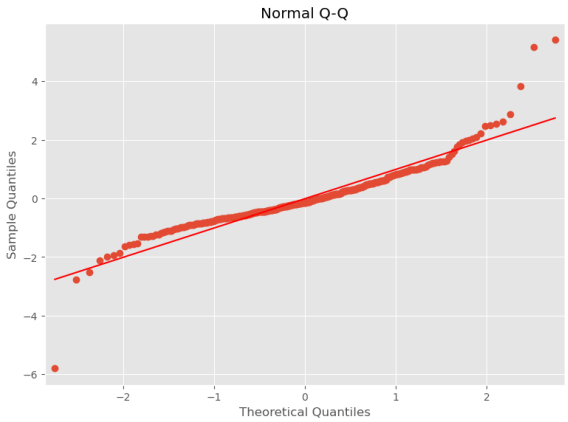
```
TestData_Hamburg_Car2Go,predict_Hamburg_Car2Go = model_fitting(df_Hamburg_Ca
```

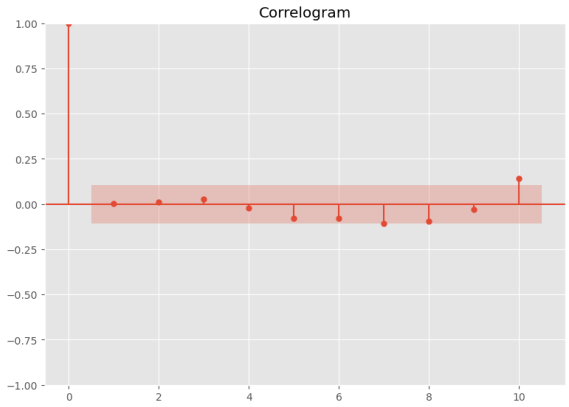
```
/opt/conda/lib/python3.7/site-packages/statsmodels/base/model.py:606: Conve
rgenceWarning: Maximum Likelihood optimization failed to converge. Check ml
e_retvals
  ConvergenceWarning)
```







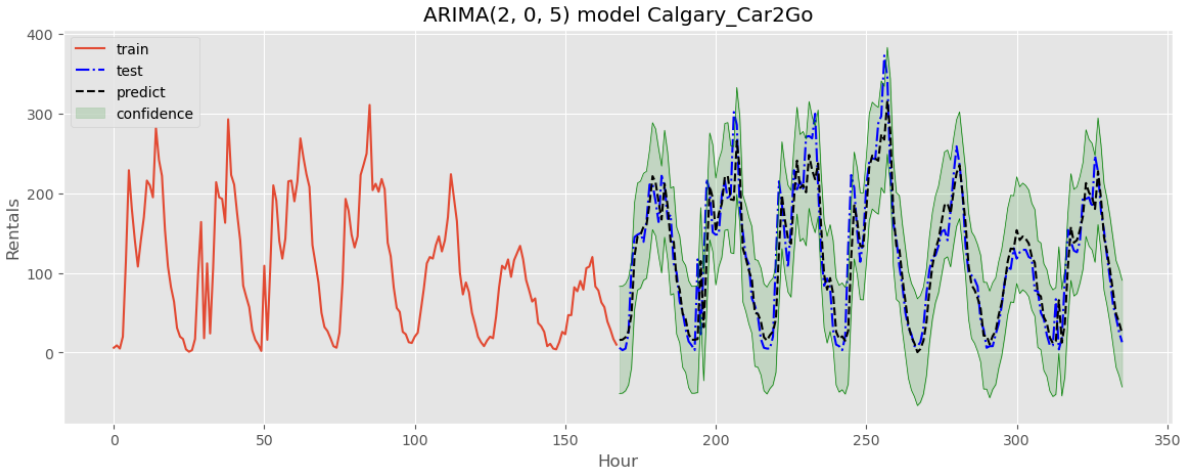


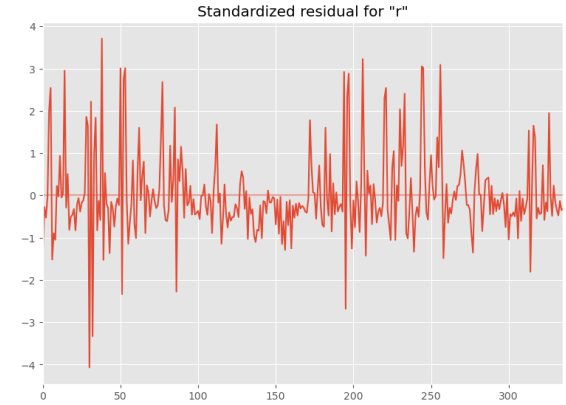


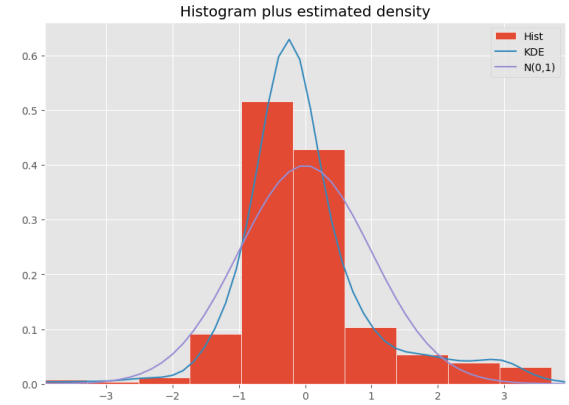
In [122…

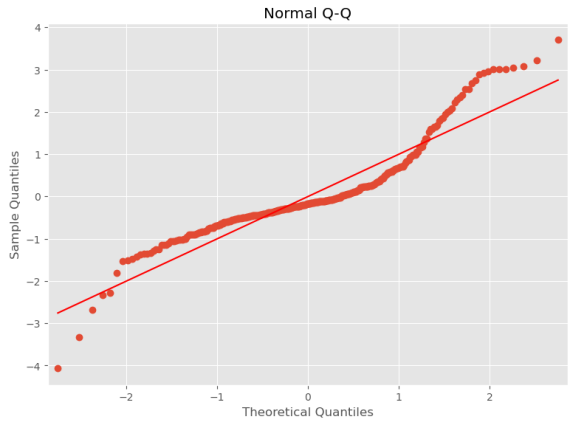
```
TestData_Calgary_Car2Go,predict_Calgary_Car2Go = model_fitting(df_Calgary_Ca
```

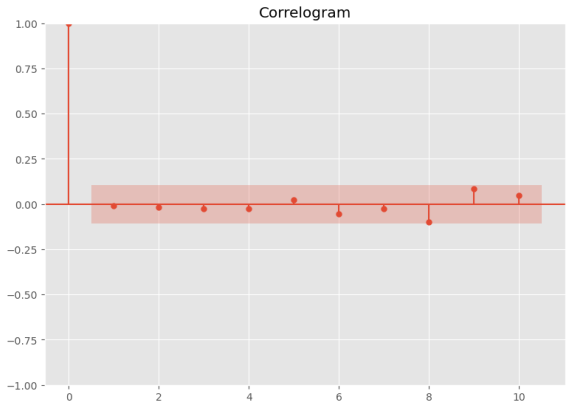
```
/opt/conda/lib/python3.7/site-packages/statsmodels/base/model.py:606: Conve
rgenceWarning: Maximum Likelihood optimization failed to converge. Check ml
e_retvals
  ConvergenceWarning)
```





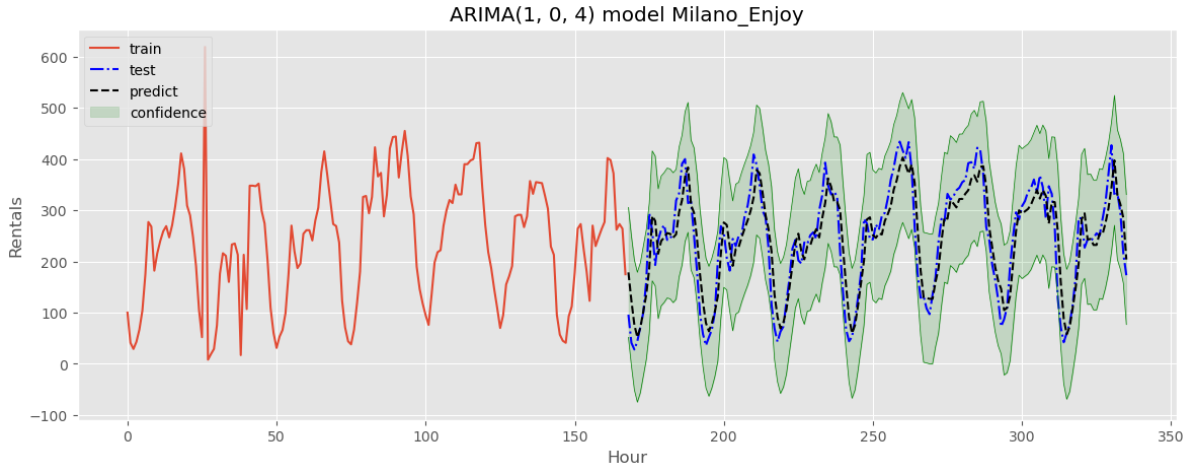


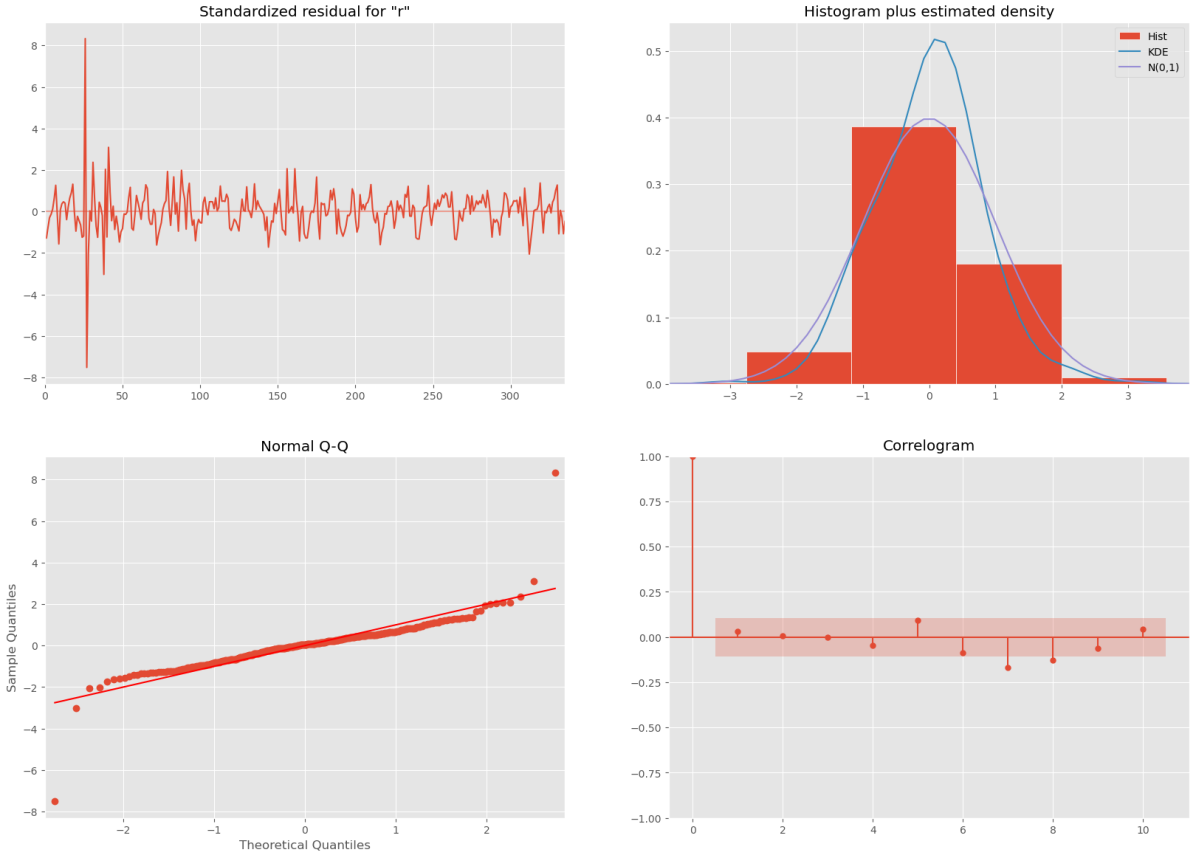




In [123…

```
TestData_Milano_Enjoy,predict_Milano_Enjoy = model_fitting(df_Milano_Enjoy,
```





Step 6 Computing Error (MPE,MAPE,R2)

In [58]:
```
from sklearn.metrics import mean_absolute_percentage_error as mape
from sklearn.metrics import mean_absolute_error as mae
from sklearn.metrics import mean_squared_error as mse
```

In [42]:
```
def error_computing(data,predict):
    MAPE = mape(data['rentals'],predict) #mape=mae/real, smaller is better
    MAE = mae(data['rentals'],predict)
    MSE = mse(data['rentals'],predict)
    R2 = np.corrcoef(data['rentals'],predict)[0,1]**2  #close to 1, obove 0.

    return MAPE,MAE,MSE,R2
```

In [34]:
```
MAPE,MAE,MSE,R2=error_computing(TestData_Milano_Car2Go,predict_Milano_Car2Go
print('MAPE:'+ str(MAPE)+'\nMAE:'+ str(MAE) +'\nMSE:'+ str(MSE)+'\nR2:'+ str
```

```
MAPE:0.25488277904661844
MAE:39.15617863807434
MSE:2377.2502229272595
R2:0.8680594097683716
```

In [35]:
```
MAPE,MAE,MSE,R2=error_computing(TestData_Hamburg_Car2Go,predict_Hamburg_Car2
print('MAPE:'+ str(MAPE)+'\nMAE:'+ str(MAE) +'\nMSE:'+ str(MSE)+'\nR2:'+ str
```

```
MAPE:0.19879009856711902
MAE:31.130714697998602
MSE:1651.1979532134778
R2:0.9065991799896558
```

In [36]:
```
MAPE,MAE,MSE,R2=error_computing(TestData_Calgary_Car2Go,predict_Calgary_Car2
print('MAPE:'+ str(MAPE)+'\nMAE:'+ str(MAE) +'\nMSE:'+ str(MSE)+'\nR2:'+ str
```

```
MAPE:0.5283810091781636
MAE:22.71011022281535
MSE:1091.5799999324106
R2:0.8469844482654749
```

In [37]:
```
MAPE,MAE,MSE,R2=error_computing(TestData_Milano_Enjoy,predict_Milano_Enjoy)
print('MAPE:'+ str(MAPE)+'\nMAE:'+ str(MAE) +'\nMSE:'+ str(MSE)+'\nR2:'+ str
```

```
MAPE:0.22244994346079286
MAE:35.49783987756967
MSE:2009.1722756277866
R2:0.8303145428347956
```

Step 7a Keep N fixed, and do a grid search varying (p,d,q)

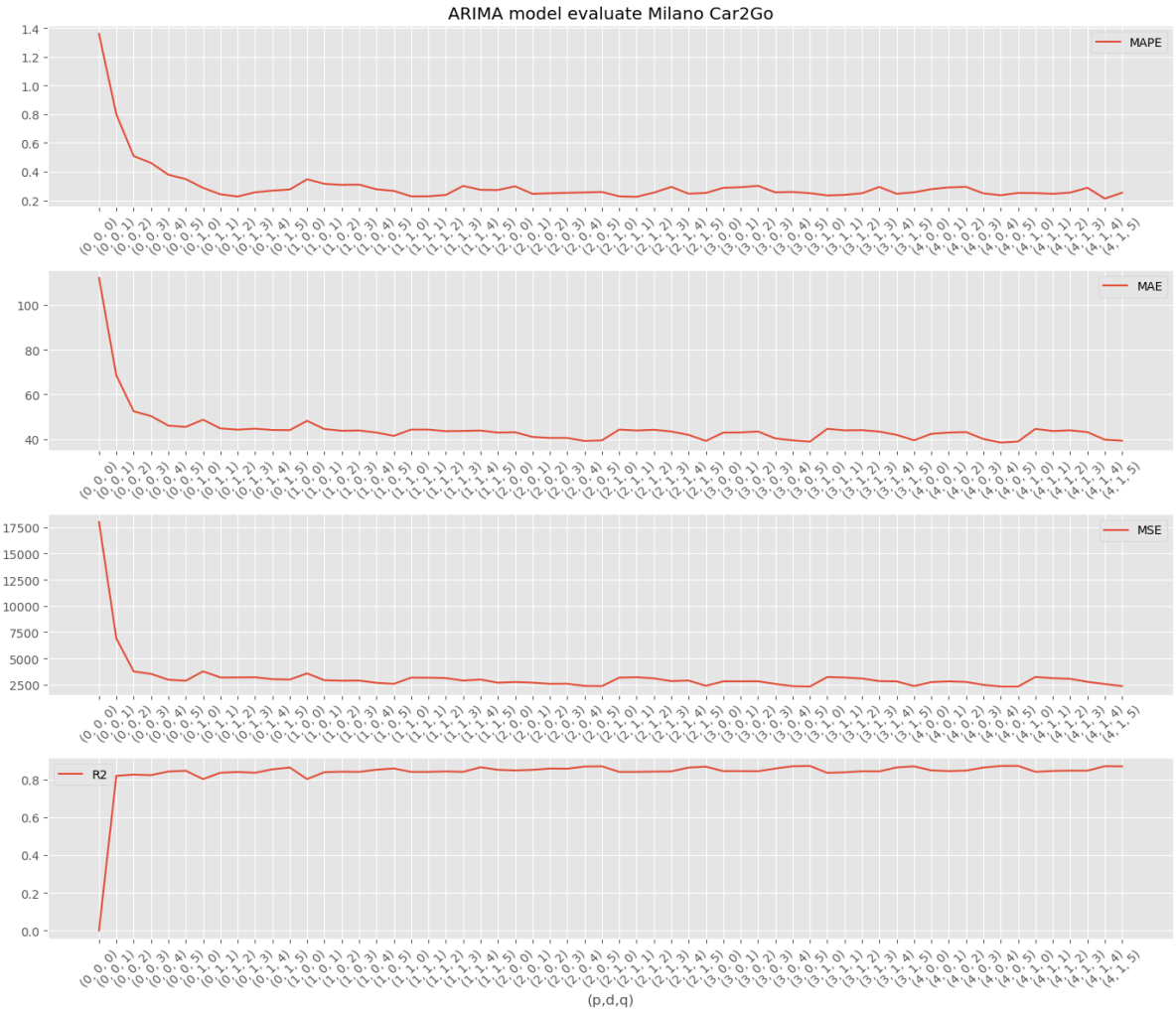
In [38]:
```
# evaluate an ARIMA model for a given order (p,d,q),N:days for training
def evaluate_arima_model(data, arima_order,N):
    # prepare training dataset
    n=N*24
    TrainData=data.iloc[:n]
    TestData=data.iloc[n:2*n]
    ARIMA_model = ARIMA(data['rentals'][:2*n], order=arima_order)
    ARIMA_result = ARIMA_model.fit()
    predict_df = ARIMA_result.get_prediction(n)
    result=predict_df.predicted_mean
    MAPE,MAE,MSE,R2 = error_computing(TestData,result)
    return MAPE,MAE,MSE,R2
```

In [39]:
```
# evaluate combinations of p, d and q values for an ARIMA model
def evaluate_models(dataset, p_values, d_values, q_values,city):
    #dataset = dataset.astype('float32')
    MAPE,MAE,MSE,R2,Order=[],[],[],[],[]
    best_mape, mape_cfg = float("inf"), None
    best_mae, mae_cfg = float("inf"), None
    best_mse, mse_cfg = float("inf"), None
    best_r2, r2_cfg = 0, None
    for p in p_values:
        for d in d_values:
            for q in q_values:
                order = (p,d,q)
                try:
                    mape,mae,mse,r2 = evaluate_arima_model(dataset, order,7)
                    Order.append(order)
                    MAPE.append(mape)
                    MAE.append(mae)
                    MSE.append(mse)
                    R2.append(r2)
                    if mape < best_mape:
                        best_mape, mape_cfg = mape, order
                    if mae < best_mae:
                        best_mae, mae_cfg = mae, order
                    if mse < best_mse:
                        best_mse, mse_cfg = mse, order
                    if r2 > best_r2:
                        best_r2, r2_cfg = r2, order
                except:
                    continue
    plot_evaluation(MAPE,MAE,MSE,R2,Order,city)
```

```
    print('Best ARIMA%s MAPE=%.3f' % (mape_cfg, best_mape))
    print('Best ARIMA%s MAE=%.3f' % (mae_cfg, best_mae))
    print('Best ARIMA%s MSE=%.3f' % (mse_cfg, best_mse))
    print('Best ARIMA%s R2=%.3f' % (r2_cfg, best_r2))
```

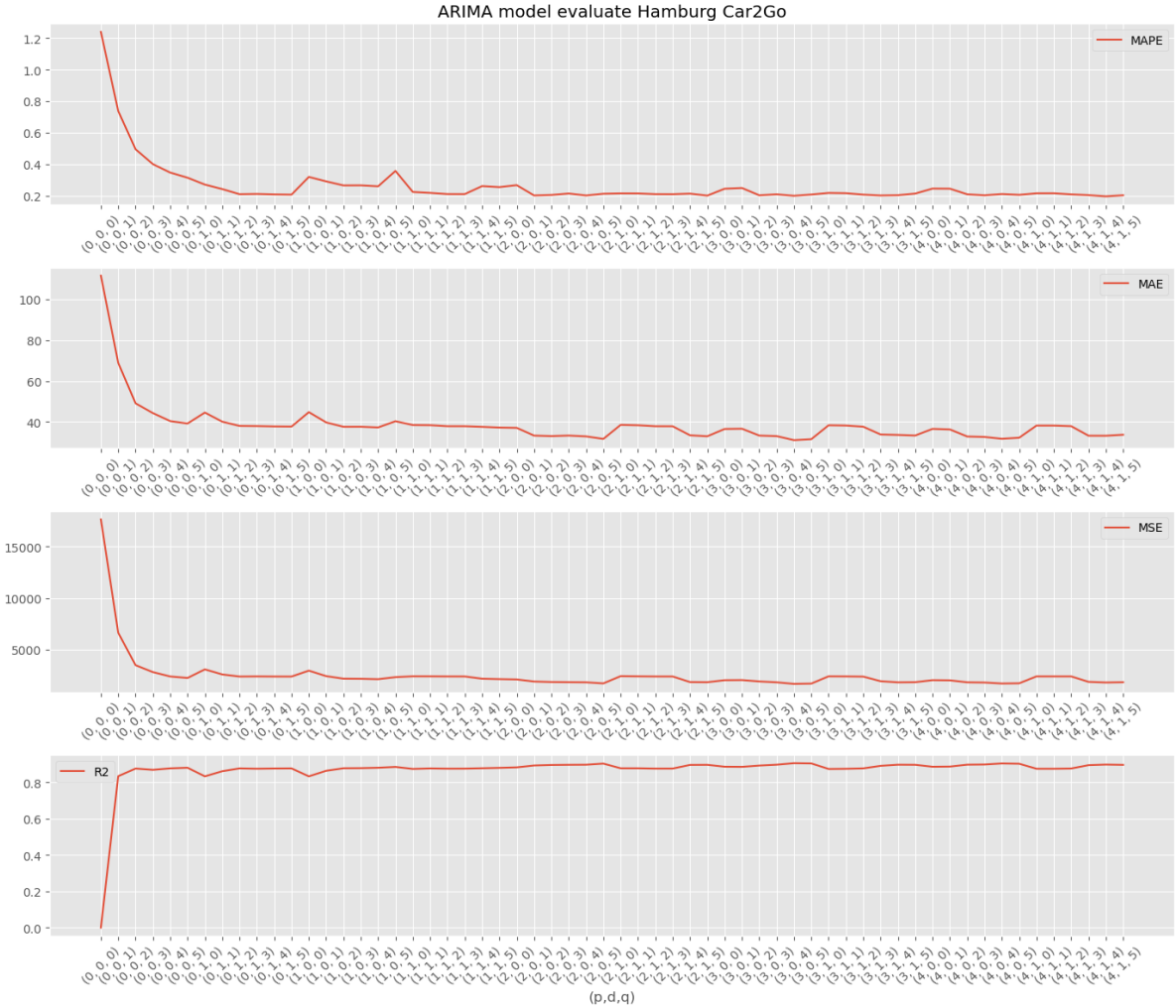
In [40]:
```
def plot_evaluation(MAPE,MAE,MSE,R2,Order,city):
    x = []
    temp_loc = 0
    for elem in Order:
        x.append(temp_loc)
        temp_loc += 1
    plt.figure(facecolor='white',figsize=(14,12))
    plt.subplot(411)
    plt.title('ARIMA model evaluate '+ city)
    plt.plot(MAPE,label='MAPE')
    plt.legend(loc='best')
    plt.xticks(x,Order,rotation=45)
    plt.subplot(412)
    plt.plot(MAE,label='MAE')
    plt.legend(loc='best')
    plt.xticks(x,Order,rotation=45)
    plt.subplot(413)
    plt.plot(MSE,label='MSE')
    plt.legend(loc='best')
    plt.xticks(x,Order,rotation=45)
    plt.subplot(414)
    plt.plot(R2,label='R2')
    plt.legend(loc='best')
    plt.xticks(x,Order,rotation=45)
    plt.xlabel('(p,d,q)')
    plt.tight_layout()
    plt.show()
```

In [41]:
```
# evaluate parameters
p_values = range(0, 5)
d_values = range(0, 2)
q_values = range(0, 6)
warnings.filterwarnings("ignore")
evaluate_models(df_Milano_Car2Go, p_values, d_values, q_values,'Milano Car2C
```



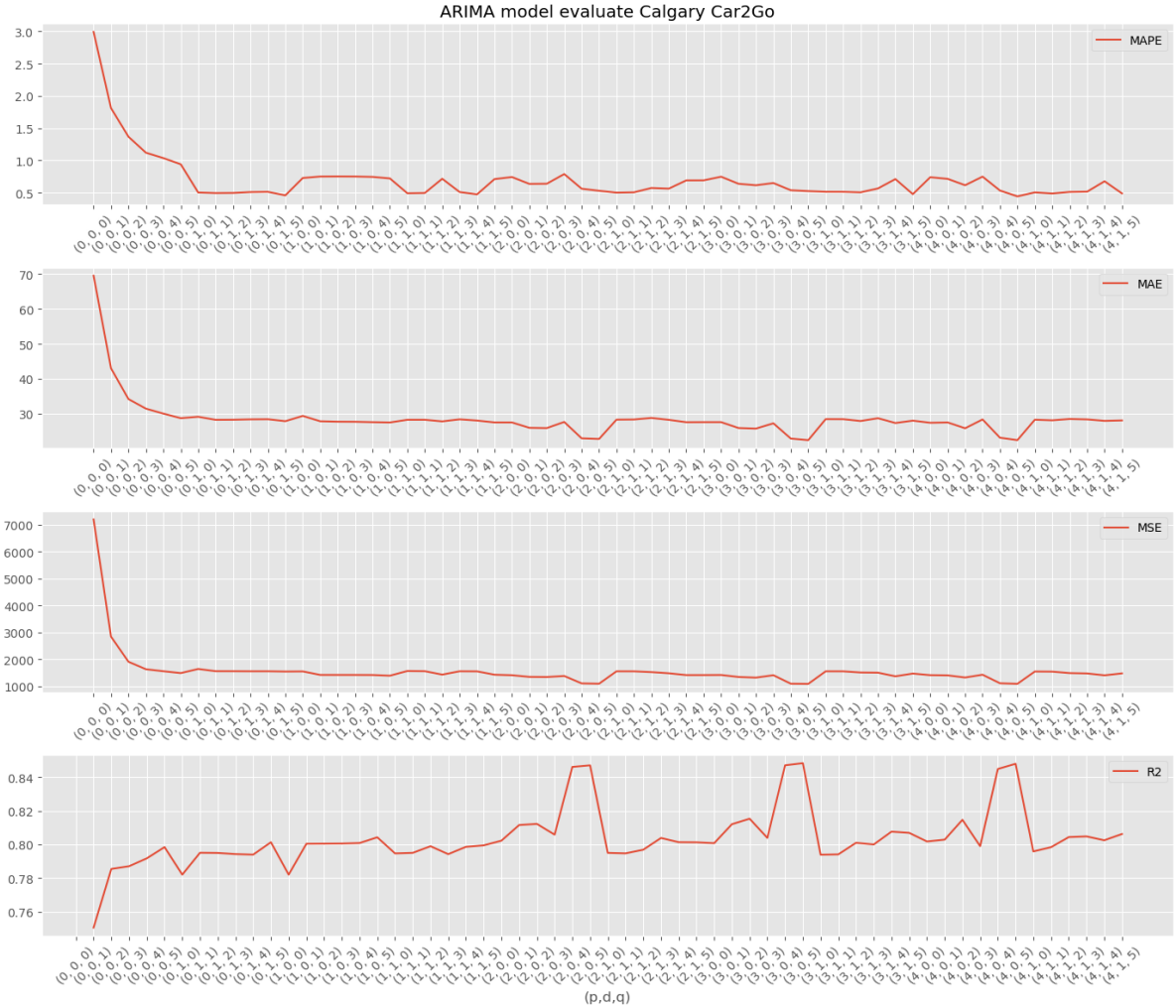

```
Best ARIMA(4, 1, 4) MAPE=0.212
Best ARIMA(4, 0, 4) MAE=38.426
Best ARIMA(4, 0, 5) MSE=2317.610
Best ARIMA(4, 0, 5) R2=0.872
```

In [42]:
```
evaluate_models(df_Hamburg_Car2Go, p_values, d_values, q_values,'Hamburg Car
```



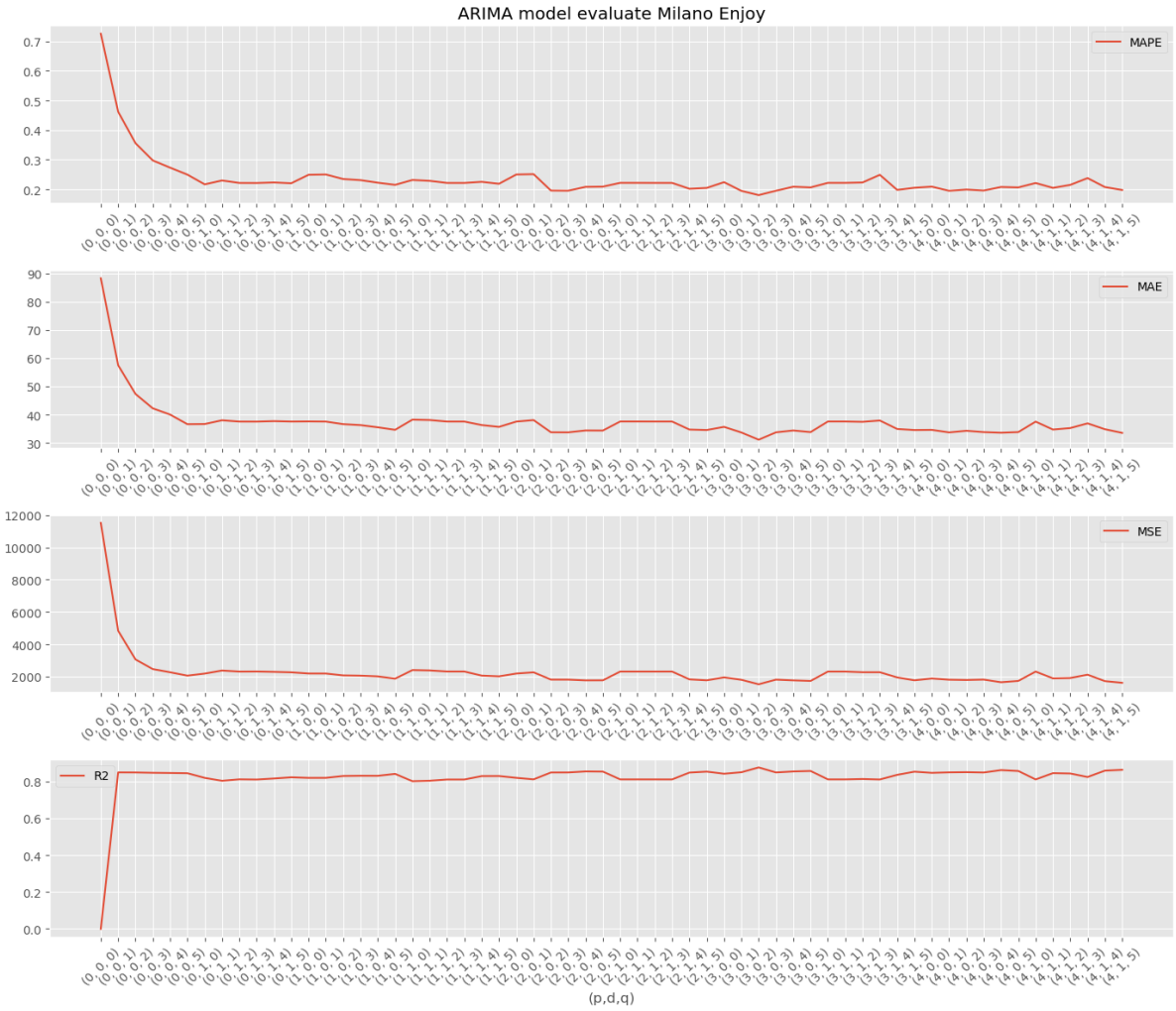

```
Best ARIMA(4, 1, 4) MAPE=0.195
Best ARIMA(3, 0, 4) MAE=31.131
Best ARIMA(3, 0, 4) MSE=1651.198
Best ARIMA(3, 0, 4) R2=0.907
```

In [44]:
```
evaluate_models(df_Calgary_Car2Go, p_values, d_values, q_values,'Calgary Car
```




```
Best ARIMA(4, 0, 5) MAPE=0.442
Best ARIMA(3, 0, 5) MAE=22.371
Best ARIMA(3, 0, 5) MSE=1085.553
Best ARIMA(3, 0, 5) R2=0.848
```

In [45]:
```
evaluate_models(df_Milano_Enjoy, p_values, d_values, q_values,'Milano Enjoy'
```




```
Best ARIMA(3, 0, 2) MAPE=0.181
Best ARIMA(3, 0, 2) MAE=31.114
Best ARIMA(3, 0, 2) MSE=1520.139
Best ARIMA(3, 0, 2) R2=0.876
```

Step 7b Given the best parameter configuration (p,d,q)=(2,0,4), change N training samples, 2 learning strategy (expanding versus sliding window). 1 week for testing

In [86]:

```
def Expanding(inputdata,p,d,q):
    MAE,MAPE,MSE,R2=[],[],[],[]
    test_len = 24*7
    Nlist = [1,3,5,7,9,11,13,17,19,21] #train list
    for i in Nlist:
        train_len = 24*i
        X = inputdata.rentals.values.astype(float)
        train = X[0:train_len]
        test = X[train_len:(train_len+test_len)]
        Test = [x for x in test]
        history = [x for x in train]
        prediction = []
        for t in range (0, test_len):
            model = ARIMA(history, order=(p,d,q))
            model_fit = model.fit()
            result = model_fit.forecast()
            prediction.append(result[0])
            history.append(Test[t])
```

```
        MAE.append(mae(test,prediction))
        MAPE.append(mape(test,prediction))
        MSE.append(mse(test,prediction))
        R2.append(np.corrcoef(test,prediction)[0,1]**2)
    datalist = np.zeros((len(Nlist),6))
    for i in range(0,len(Nlist)):
        datalist[i][0] = Nlist[i] * 24
        datalist[i][1] = Nlist[i]
        datalist[i][2] = MAE[i]
        datalist[i][3] = MAPE[i]
        datalist[i][4] = MSE[i]
        datalist[i][5] = R2[i]
    return datalist
```

In [87]:
```
def SlidingWindows(inputdata,p,d,q):
    MAE,MAPE,MSE,R2=[],[],[],[]
    test_len = 24*7
    Nlist = [1,3,5,7,9,11,13,17,19,21] #train list
    for i in Nlist:
        train_len = 24*i
        X = inputdata.rentals.values.astype(float)
        train = X[:train_len]
        test = X[train_len:(train_len+test_len)]
        Test = [x for x in test]
        history = [x for x in train]
        prediction = []
        for t in range (0, test_len):
            model = ARIMA(history[t:], order=(p,d,q))
            model_fit = model.fit()
            result = model_fit.forecast()
            prediction.append(result[0])
            history.append(Test[t])
        MAE.append(mae(test,prediction))
        MAPE.append(mape(test,prediction))
        MSE.append(mse(test,prediction))
        R2.append(np.corrcoef(test,prediction)[0,1]**2)
    datalist = np.zeros((len(Nlist),6))
    for i in range(0,len(Nlist)):
        datalist[i][0] = Nlist[i] * 24
        datalist[i][1] = Nlist[i]
        datalist[i][2] = MAE[i]
        datalist[i][3] = MAPE[i]
        datalist[i][4] = MSE[i]
        datalist[i][5] = R2[i]
    return datalist
```

In [92]:
```
Milano_Car2Go_exp = pd.DataFrame(Expanding(df_Milano_Car2Go,2,0,4))
Milano_Car2Go_exp.rename(columns={0:'N_hours',1:'N_days',2:'MAE_exp',3:'MAPE
Milano_Car2Go_exp
```

Out[92]:

	N_hours	N_days	MAE_exp	MAPE_exp	MSE_exp	R2_exp
0	24.0	1.0	48.369327	0.364578	4362.951610	0.785093
1	72.0	3.0	40.389984	0.282432	2571.797884	0.875137
2	120.0	5.0	39.600538	0.269164	2413.088304	0.868347
3	168.0	7.0	40.352075	0.267769	2548.675979	0.858535
4	216.0	9.0	39.645854	0.253922	2379.505585	0.866637
5	264.0	11.0	40.158528	0.257540	2366.802519	0.869956
6	312.0	13.0	40.168626	0.257317	2419.671424	0.872033
7	408.0	17.0	41.664935	0.266920	2664.110543	0.858577
8	456.0	19.0	40.659231	0.267763	2575.148922	0.853627
9	504.0	21.0	38.297598	0.266959	2336.788145	0.866054

In [89]:
```
Milano_Car2Go_sw = pd.DataFrame(SlidingWindows(df_Milano_Car2Go,2,0,4))
Milano_Car2Go_sw.rename(columns={0:'N_hours',1:'N_days',2:'MAE_sw',3:'MAPE_s
Milano_Car2Go_sw
```

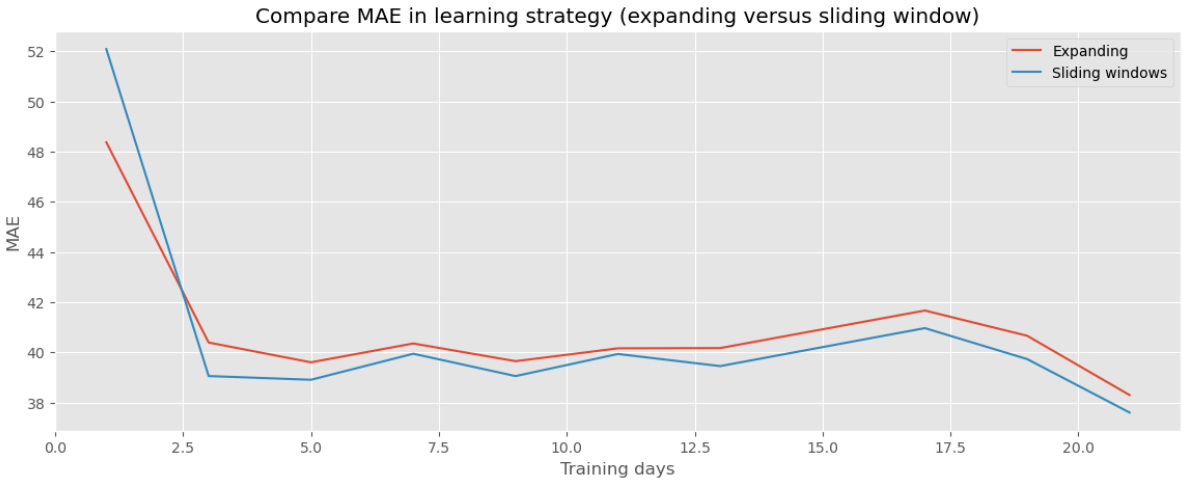
Out[89]:

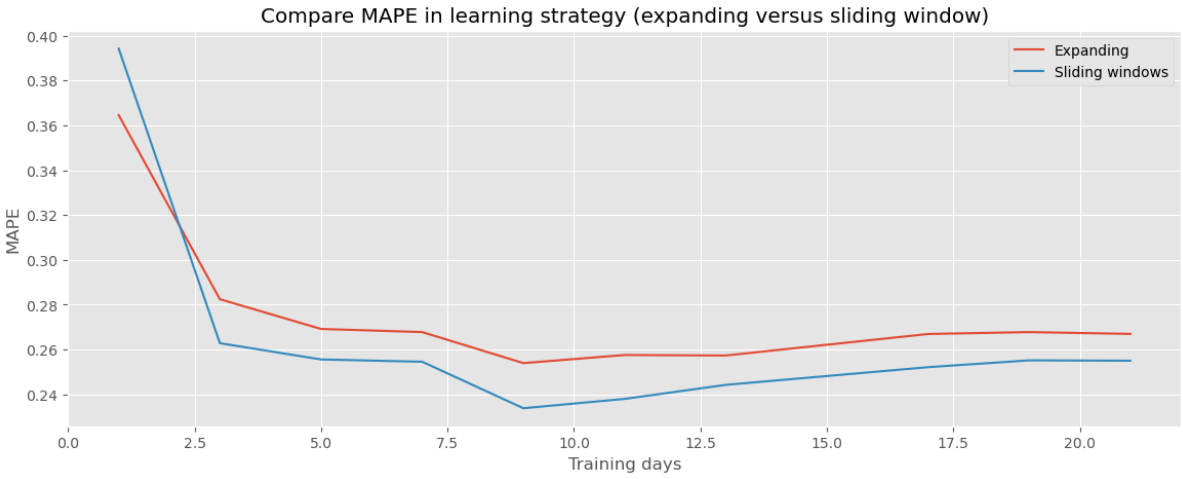
	N_hours	N_days	MAE_sw	MAPE_sw	MSE_sw	R2_sw
0	24.0	1.0	52.083533	0.394245	5234.054877	0.747561
1	72.0	3.0	39.053323	0.262809	2499.916988	0.877044
2	120.0	5.0	38.905174	0.255561	2264.185947	0.876207
3	168.0	7.0	39.942885	0.254532	2521.605366	0.860330
4	216.0	9.0	39.052320	0.233786	2332.827268	0.869196
5	264.0	11.0	39.933015	0.237957	2413.342999	0.867299
6	312.0	13.0	39.450141	0.244215	2471.994508	0.869176
7	408.0	17.0	40.963789	0.252110	2640.222727	0.859923
8	456.0	19.0	39.729306	0.255165	2474.011458	0.859430
9	504.0	21.0	37.601383	0.255000	2262.880964	0.870283

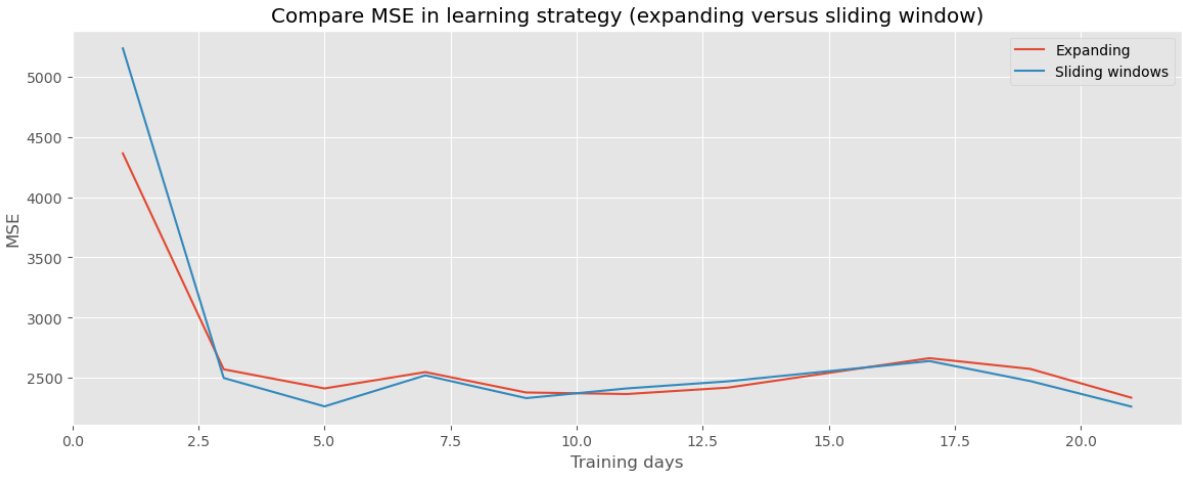
In [93]:
```
def plot_exp_sw(data_exp,data_sw):
    fig=plt.figure(figsize=(14,5))
    plt.plot(data_exp['N_days'],data_exp['MAE_exp'],label='Expanding')
    plt.plot(data_sw['N_days'],data_sw['MAE_sw'],label='Sliding windows')
    plt.ylabel('MAE')
    plt.xlabel('Training days')
    plt.title('Compare MAE in learning strategy (expanding versus sliding wi
    plt.legend()

    fig=plt.figure(figsize=(14,5))
    plt.plot(data_exp['N_days'],data_exp['MAPE_exp'],label='Expanding')
    plt.plot(data_sw['N_days'],data_sw['MAPE_sw'],label='Sliding windows')
    plt.ylabel('MAPE')
    plt.xlabel('Training days')
    plt.title('Compare MAPE in learning strategy (expanding versus sliding w
    plt.legend()

    fig=plt.figure(figsize=(14,5))
    plt.plot(data_exp['N_days'],data_exp['MSE_exp'],label='Expanding')
```

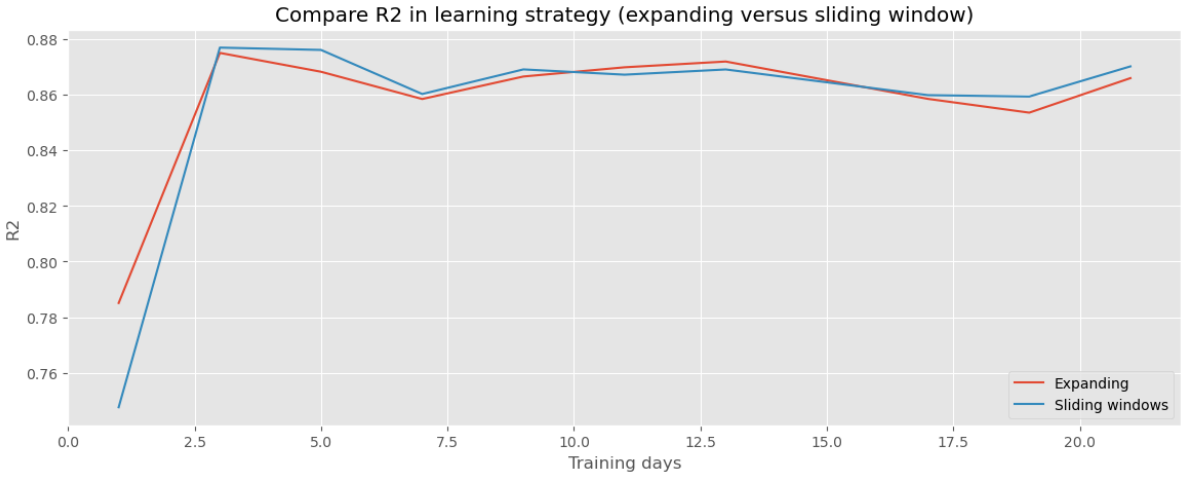
```
    plt.plot(data_sw['N_days'],data_sw['MSE_sw'],label='Sliding windows')
    plt.ylabel('MSE')
    plt.xlabel('Training days')
    plt.title('Compare MSE in learning strategy (expanding versus sliding wi
    plt.legend()

    fig=plt.figure(figsize=(14,5))
    plt.plot(data_exp['N_days'],data_exp['R2_exp'],label='Expanding')
    plt.plot(data_sw['N_days'],data_sw['R2_sw'],label='Sliding windows')
    plt.ylabel('R2')
    plt.xlabel('Training days')
    plt.title('Compare R2 in learning strategy (expanding versus sliding wir
    plt.legend()
```

In [94]:
```
plot_exp_sw(Milano_Car2Go_exp,Milano_Car2Go_sw)
```







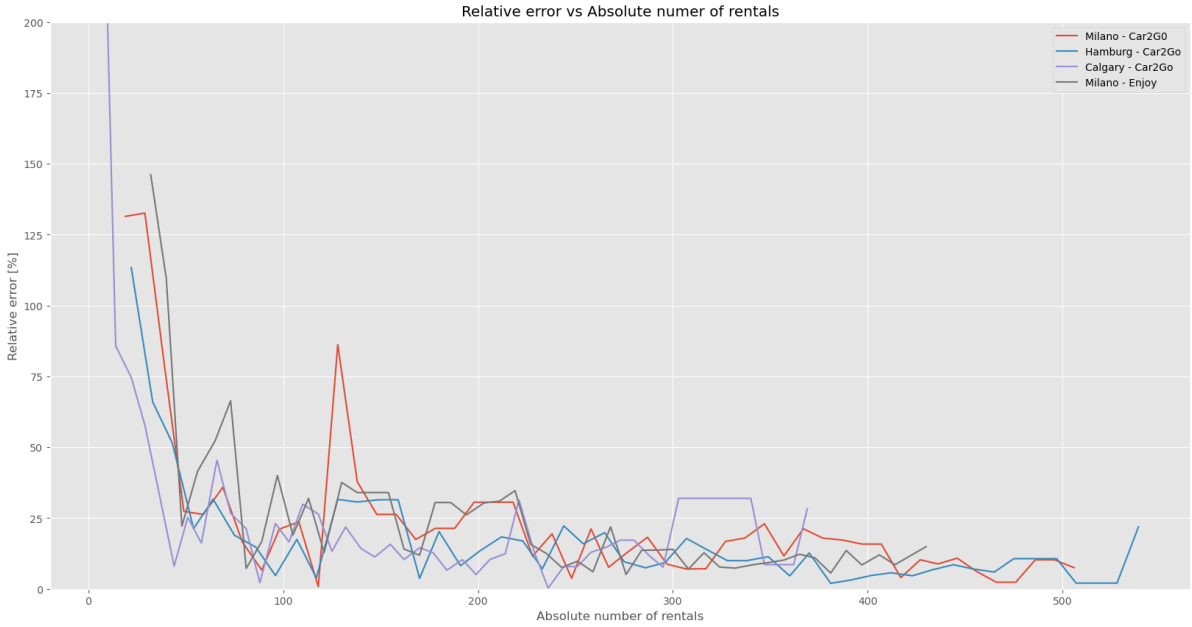


7c

In [ ]:
```
errors = {}
for name, test, predict in [
    ("Milano - Car2G0", TestData_Milano_Car2Go, predict_Milano_Car2Go),
    ("Hamburg - Car2Go", TestData_Hamburg_Car2Go, predict_Hamburg_Car2Go),
    ("Calgary - Car2Go", TestData_Calgary_Car2Go, predict_Calgary_Car2Go),
    ("Milano - Enjoy", TestData_Milano_Enjoy, predict_Milano_Enjoy),
]:
    rentals = test["rentals"]
    df = DataFrame({
        "real": list(rentals),
        "error": list(abs(predict - rentals) / rentals * 100),
    }, index=range(len(test)))
    errors[name] = df
```

In [163…
```
for city, df in errors.items():
    df_mean = DataFrame()
    df_mean["bin"] = pd.cut(df["real"], 50).apply(lambda x: round(x.mid))
    df_mean["error"] = df["error"]
    df_mean = df_mean.groupby("bin").mean()
    plt.plot(df_mean["error"].ffill(), label=city,)
plt.legend()
plt.xlabel("Absolute number of rentals")
plt.ylabel("Relative error [%]")
plt.title("Relative error vs Absolute numer of rentals")
plt.ylim(0, 200)
```

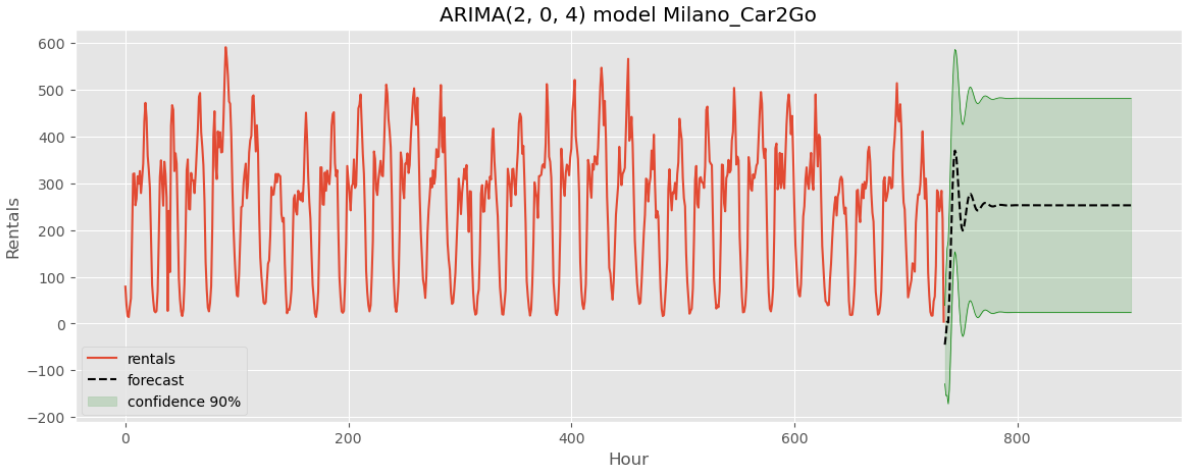
Out[163]:
```
(0.0, 200.0)
```



Compare results for the different cities.

Try to see how the time horizon h of the prediction impacts the performance. Instead of predicting the number of rentals at t+1, use the model to predict the future rentals at t+h, h in [1:24] or more. What happens if you change the parameter p? Which value could allow a better long-term prediction?

In [62]:
```
# Arima N:1 day
def model_fitting_forecast(data,city,N,order):
    n=N*24
    ARIMA_model = ARIMA(data['rentals'], order=order)  #order中分别为AR\差分和
    ARIMA_result = ARIMA_model.fit()
    predict_df = ARIMA_result.get_forecast(n) #predict(start=n,end=2*n,dynam
    confident = predict_df.conf_int(alpha=0.1)
    result=predict_df.predicted_mean #返回预测结果
    fig=plt.figure(figsize=(14,5))
    plt.plot(data['rentals'].index,data['rentals'],label='rentals')
    plt.plot(result.index, result,'--k',label='forecast')
    plt.plot(confident.index, confident['lower rentals'], 'g',marker='.',mar
    plt.plot(confident.index, confident['upper rentals'], 'g',marker='.',mar
    plt.fill_between(confident.index,confident['lower rentals'],confident['u
    plt.ylabel('Rentals')
    plt.xlabel('Hour')
    plt.title('ARIMA'+str(order)+' model '+city)
    plt.legend()

model_fitting_forecast(df_Milano_Car2Go, 'Milano_Car2Go',7,(2,0,4))
```



In [64]:
```
Data = df_Milano_Car2Go.rentals.values.astype(float)
start = 24*7
train_len = 24*7
test_len = 24*7
test = Data[start+train_len:start+train_len+test_len]
predictions_data = np.zeros((24,test_len))
for i in range(1,25):
    train = Data[start-i+1:start+train_len-i+1]
    history = [x for x in train]
    for t in range(0, test_len):
        model = ARIMA(history , order=(2,0,4))
        model_fit = model.fit()
        result = model_fit.forecast(steps=i) # forecast next i hours
        predictions_data[i-1][t]=result[-1]
        history.append(Data[start+train_len+t-i])
row,col = predictions_data.shape
xlist = np.arange(0, test_len, 1)
plt.figure(figsize=(20,10))
for i in range(24):
    plt.plot(xlist ,predictions_data[i], label='h='+str(i+1))
plt.plot(xlist, test , label='Orignal')
plt.ylabel('Rentals')
plt.xlabel('Test samples(7 days)')
plt.title('Predictions with h varying')
plt.legend(loc='best')
plt.show()
```