Laboratory 2 – Prediction using ARIMA models

GROUP 16

Elsa Romeo, s317695
Umberto Brozzo Doda, s314562
Giovanni Dettori, s315017

January 6th, 2024

A.Y. 2023-2024

1 Time series extraction and filtering

✓ From laboratory 1, the time series of Munchen, Vancouver (Car2Go system), and Milano (Enjoy) have been selected, maintaining the filtering criteria unaltered. Indeed, only bookings where there has been a movement, lasting more than 5 minutes and less than 3 hours are considered rentals. The period selected is from 8 January 2017 to 8 February 2017 in Munchen and Vancouver, while in Milan during the same period in 2018. The choice has been performed to avoid a non-stationary trend, such as in the Christmas period, or wide periods with missing data. The difference in the year for the three cities is because a good overlapping period between Enjoy and Car2go has not been found. ✓

2 Missing samples

The ARIMA model requires the time series to have no missing values. In the dataset used for this activity, some values are missing. In order to avoid this problem, missing values have been replaced by linearly interpolating the extreme values of the time in which there are no values and adding a random component w. If there are h missing values consecutively, and the last value before the missing ones is v_o and the first after them is v_f, the $i-th$ missing value will be replaced with:

$$v_i = v_o + \frac{v_f - v_o}{h} \cdot i + w_i \qquad \text{where} \quad i \in [1, h-1] \qquad (1)$$ ✓

3 Checking for stationarity

ARIMA models are based on the assumption that the statistical properties of a time series are quite constant or, at least, biased by a trend. As a consequence, it is possible to forecast future values learning from a previous period. For this reason, it is necessary to check for stationarity and the presence of any trend to be removed. Having a look at the representation of the time series, there is no evidence of the presence of a trend, but a seasonality is present. Indeed, in all three cities, the days and the weeks are recognizable.
To confirm the previous assumptions a statistical analysis has been performed. In particular, by plotting the rolling statistics, it is possible to visually inspect if the statistical properties are constant during time.

over which ΔT ?

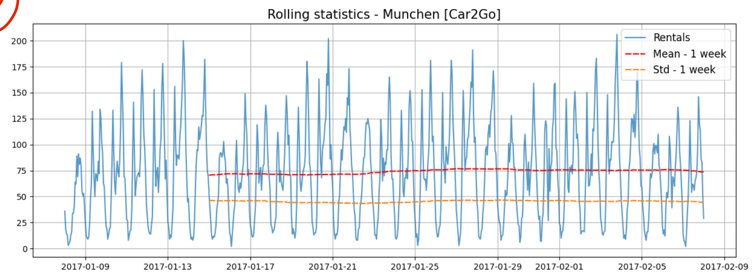
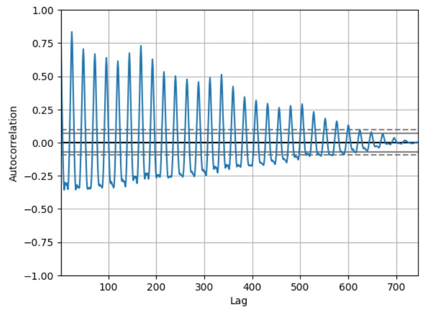


Figure 1: Rolling statistics in the city of Munchen.

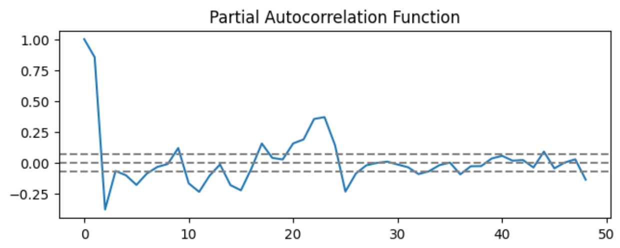
Figure 1 reports the rolling mean and standard deviation in the city of Munchen, considering a window of one week. As it is possible to notice both properties are quite constant and don't present relevant irregularities. For this reason, it is possible to conclude that the time series is stationary. Therefore, the parameter d has been chosen as equal to 0, obtaining an ARIMA model with parameters (p, 0, q), where p and q have not yet been chosen. ✓

4 ACF and PACF

To select the optimal values for p and q, ACF (Autocorrelation Function) and PACF (Partial Autocorrelation Function) of the time series were computed. PACF is useful for identifying Autoregressive (AR) models and stops after lag p for purely AR models, representing the direct correlation between observations separated by p time steps. ACF is suitable for Moving Average (MA) models, with the gradual decrease in correlation indicating the size of the moving average window (q). As shown in Figure 2a positive and negative values in ACF reveal patterns of the different hours of the day and differences between day and night. In this specific case, ACF enters the confidence interval after 600 lags, indicating a slow decay in autocorrelation, while PACF enters after 30 lags as highlighted in Figure 2b, suggesting a more immediate influence of past observations on the current one.



(a) Autocorrelation function



(b) Partial Autocorrelation Function

Figure 2: Munchen

5 Training and testing

For the correct development of an ARIMA model, it is necessary to split the available data into two subsets: the training and testing ones. Therefore, an important parameter to be tuned is the length of the training set, as it is explained in detail in section 7.2. However, as an initial approach, a window of three weeks for training and one week for testing was chosen. Specifically, the training set runs from 8/01 to 29/01, and the test set from 1/02 to 8/02. In addition, the sliding learning method has been selected.

6 Model evaluation

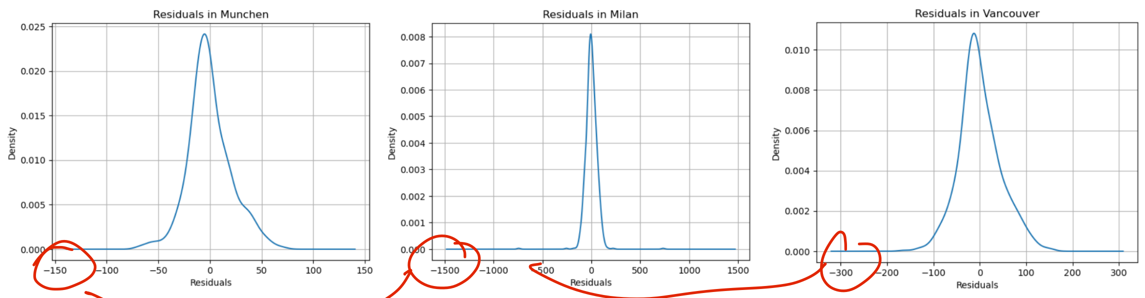
Considering the same scenario explained in section 5 and fixing p and q values equal to 2, further analyses have been performed. The objective was to evaluate the performance of a general model in each city and to compare it.

City	MAPE	R^2	MSE	MAE
Munchen	21.075	0.786	421.858	15.582
Vancouver	19.744	0.856	2038.060	35.974
Milan	22.955	0.847	2788.751	40.194

Table 1: Different metrics for the cities of Munchen, Vancouver, and Milano

Table 1 shows for Munchen, Vancouver, and Milan the Mean Absolute Percentage Error (MAPE), Coefficient of Determination (R^2), Mean Squared Error (MSE), and Mean Absolute Error (MAE). However, to have a meaningful comparison only the MAPE and R^2 have been taken into account, because the first is a percentage error while the latter is an index. The lowest values of MAPE are recorded for Munchen and Vancouver, suggesting both cities perform well across all metrics and indicating that the forecasting model is quite accurate. Finally, Milan has the highest values for MAPE, indicating that the forecasting model for Milan is relatively less accurate compared to the other cities. Overall, the three cities have a reasonably high R^2 value, indicating a good fit of the model to the data. ✓

Afterward, a visual inspection of the residual distribution was performed to recognize how the differences between the actual and predicted values are distributed. As it is possible to see in Figure 3 the general shape is similar to a Gaussian distribution with a mean close to zero. Therefore, the model performs quite well even if some non-idealities persist. For instance, especially in the distribution of Munchen, it is possible to notice some humbs in the bottom part. In addition, in the figure 3b the tails are rather long, which means that there are some points in which the model behaves very badly. ✓



(a) Munchen. (b) Milano. (c) Vancouver.

Figure 3: Evaluation of the residuals in the three cities.

different scales do not help in comparison

7 Impact of parameters

The objective of this section is to tune correctly the parameters finding the correct triple (p, d, q), the best length of the training set (N), and if it is better to use an expanding or sliding window.

7.1 Tuning parameters p and q

Optimizing the model performance requires an adjustment of the parameters p and q. While keeping N fixed, the hyperparameter p = 0, 1, 2, 3, 4 was tested together with q = 0, 1, 2, 3 and the calculated MAPE was used for comparing and selecting the optimal pair of values (p, q). The heatmap in Figure 4a shows that for the city of Munchen the best combination is using p = 2 and q = 1 as parameters for the ARIMA model. Although the MAPE is not the lowest for this pair of values, it is the most suitable when considering also the complexity and robustness of the model. The latter is indeed higher for smaller values of p and q. Further investigation has been carried out with higher values of p as shown in Figure 4b. By increasing the number of lag observations (p) while keeping the moving average window (q) fixed, the Mean Absolute Percentage Error values are significantly lower. Fitting a large number of parameters requires a large computation time and a high value of p may lead to overfitting. For these reasons, to perform further analyses on the model, it has been chosen to keep using the combination p = 2, d = 0, and q = 1 which still provides accurate predictions for the ARIMA model. ✓

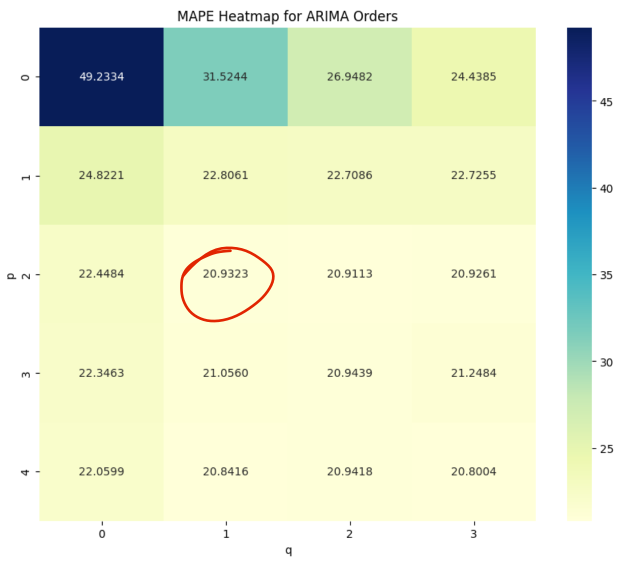
7.2 Tuning N

Selected the optimal values for the ARIMA model, different values of N were tested to observe the variation of the outcomes. By considering a window of 24 days, both MAPE using the sliding window strategy and

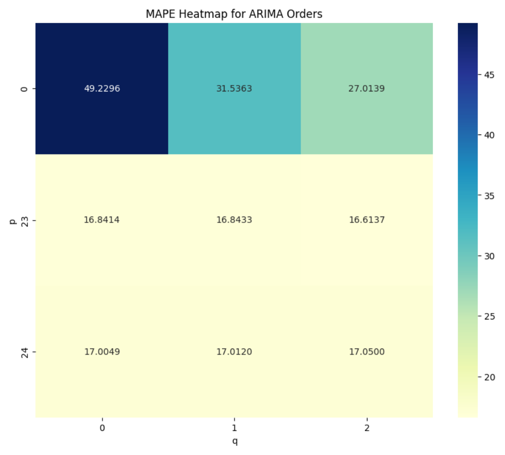
↳ for what?

is the testing set the same?

are all digits significant ?



(a) MAPE with low values of p.

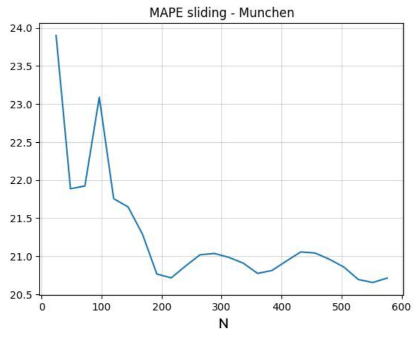


(b) MAPE with high values of p.

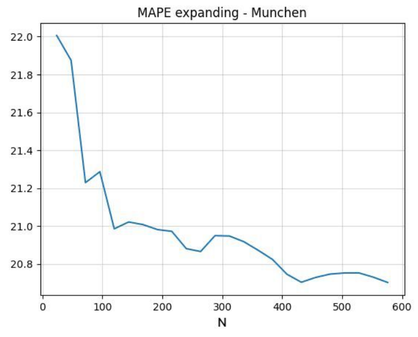
Figure 4: MAPE heatmaps for different combinations of parameters p and q

would it be better to consider N = 24.k ?

expanding window strategy were calculated and plotted to show the model behavior. As highlighted in Figure 8, both graphs show a downward trend, demonstrating a consistent decrease in the MAPE on the y-axis with increasing values of N. The peak at N = 100, is peculiar to this city´s dataset and although in Figure 8a is evident, the difference of the percentage error is still relatively small. The difference is indeed not noticeable anymore when the expanding window strategy is used as the impact of recent observations on the overall calculations diminishes.



(a) MAPE using sliding window



(b) MAPE using expanding window

Figure 5: Varying N keeping p,d,q fixed at 2,0,1

7.3 Comments on the results in different cities

Comparing the results among Milan, Munich, and Vancouver reveals that the relative error decreases as N increases for all three cities. What varies is the rate at which it decreases, especially in the first 100 lags; a negative exponential trend can be recognized. As demonstrated in the Appendix, the graph depicting the trend in Vancouver exhibits a steeper decline in Mean Absolute Percentage Error (MAPE).

?

so which is the best policy at the end ?

8 [Optional] Impact of the time horizon on the prediction

In the previous parts, the ARIMA model was used to predict the next value in the time series, i.e. the value at $t + 1$. However, the ARIMA model can be used to predict also subsequent values after a certain horizon h, i.e. the value at $t + h$. Fig. 6 shows the result obtained using an ARIMA model (2, 0, 1) to predict values in the time series with different time horizon h, in particular, the values used for h were 1, 3, 18, and 24. As the time horizon increases, the predictions become less precise.

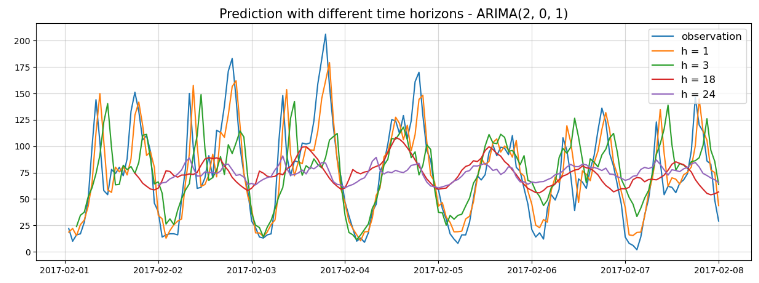
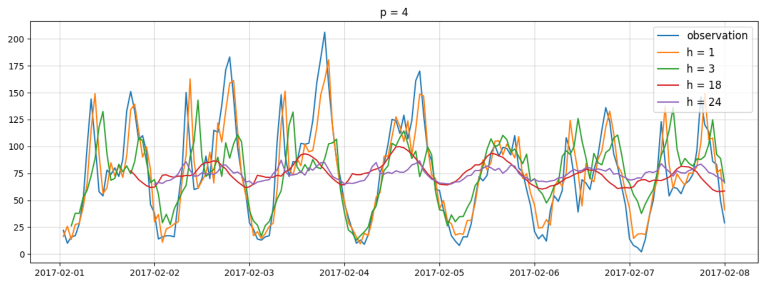


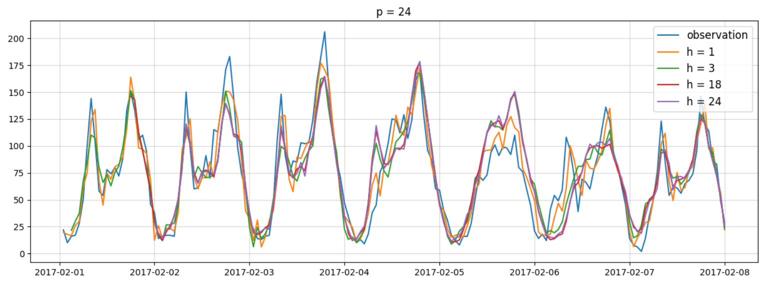
Figure 6: Prediction of ARIMA model (2,0,1) with different time horizon

Since parameter p is the one determining how many previous samples are considered for the prediction of the new one, it would be interesting to see how this parameter impacts the prediction with different time horizons. Fig. 7 shows the prediction created by three different ARIMA models, respectively with p equal to 4 and 24, with the same time horizons used before. Results show that by increasing the value of p, the prediction becomes more precise even when the time horizon is large. In particular, we obtain the best results when p is equal to 24: this is due to the fact that samples taken 24 hours apart are still correlated as it is possible to notice from the autocorrelation function of the time series shown in Fig. 2a.

✓



(a) p=4



(b) p=24

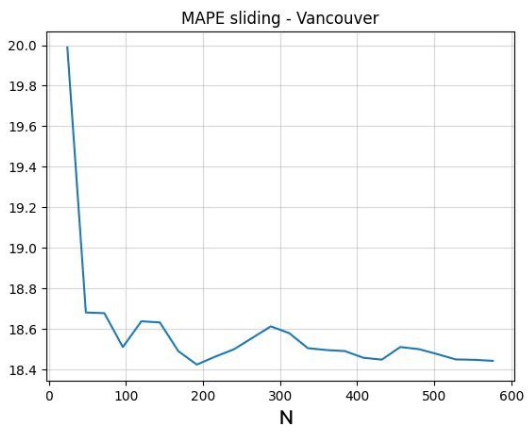
Figure 7: Results obtained applying three different ARIMA models respectively with p equal to 4 and 24, with different time horizons

- well done and complete
- missing baseline comparison

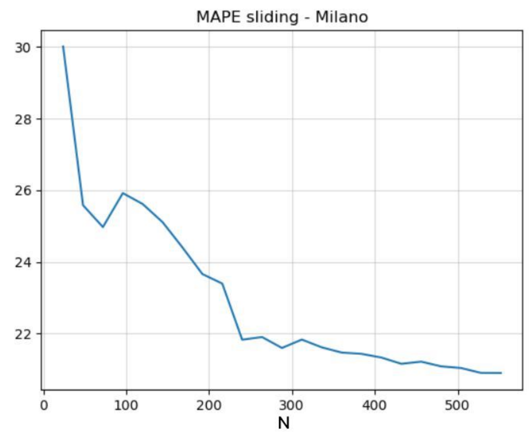
5/5

9 Appendix

9.0



(a) MAPE using sliding window



(b) MAPE using expanding window

Figure 8: Varying N keeping p,d,q fixed at 2,0,1

9.1

```
import pandas as pd
import matplotlib.pylab as plt
import numpy as np
from sklearn.metrics import mean_squared_error
from sklearn.metrics import mean_absolute_error
from sklearn.metrics import r2_score
from statsmodels.tsa.arima.model import ARIMA
from statsmodels.tsa.stattools import acf,pacf
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.metrics import mean_absolute_error
import warnings
warnings.filterwarnings('ignore')

def dateparse(date_string):
    return pd.to_datetime(date_string)

CITY = "Munchen"

if CITY == "Munchen":
    df = pd.read_csv('data/file_Munchen.csv', sep=',', parse_dates=['date'], date_parser=
        dateparse, index_col=1, dtype={'count': np.float64})
elif CITY == "Milano":
    df = pd.read_csv('data/file_Milano.csv', sep=',', parse_dates=['date'], date_parser=dateparse
        , index_col=1, dtype={'count': np.float64})
else:
    df = pd.read_csv('data/file_Vancouver.csv', sep=',', parse_dates=['date'], date_parser=
        dateparse, index_col=1, dtype={'count': np.float64})

df=df.rename(columns={'count':'rental','date':'Time'})
df.head()

df.rental+=np.random.random_sample(len(df.rental))/10.0
df.plot(label='Time',figsize=(15,5))

t_start=pd.to_datetime("2017-01-08 00:00:00")
t_end=pd.to_datetime("2017-02-08 00:00:00")
df=df.loc[(df.index>=t_start)&(df.index<=t_end)]
df.plot(figsize=(15,5))
```

9.2

```
#verify if there are missing samples
df2=pd.DataFrame(columns=['Time','rental'])
for i in range(1,len(df),1):
    if(str(df.index[i]-df.index[i-1])!='0 days 01:00:00'):
        steps=pd.date_range(df.index[i-1],df.index[i],freq='1H')
        v_init=df.rental[i-1]
        v_final=df.rental[i]
        delta=(v_final-v_init)/(len(steps)-2)
        for j in range (1,len(steps)-1):
            df2 = pd.concat([df2, pd.DataFrame([{'Time':steps[j],'rental':v_init+delta*j+np.
                random.random_sample()/10.0}])], ignore_index=True)

print(df2)
df2=df2.set_index('Time')
df=pd.concat([df,df2],sort=True)
df=df.sort_index()
df.plot(figsize=(15,5))
```

9.3

```
#check stationarity
N=7
mean_list=[]
std_list = []
for i in range(0,len(df)-N*24):
    df_window=df.iloc[i:N*24+i]
    mean_list.append(df_window['rental'].mean())
    std_list.append(df_window['rental'].std())
plt.figure(figsize=(15,5))
plt.title(f"Rolling statistics - {CITY}")
plt.plot(df.index, df["rental"], alpha=0.8, linewidth = 1.5, label = "Rentals", )
plt.plot(list(df.index)[N*24:len(df["rental"])], mean_list,'r', label = "Mean - 1 week",
    linestyle=(0, (5, 1)))
plt.plot(list(df.index)[N*24:len(df["rental"])], std_list, label = "Std - 1 week", linestyle =
    (0, (5, 1)))
plt.legend()
plt.grid()
plt.show()

df=df.rental
df.head()
```

9.4

```
#autocorrelation pt 4
pd.plotting.autocorrelation_plot(df)

lag_acf=acf(df,nlags=48)
lag_pacf=pacf(df,nlags=48)
```

```
#plot acf
plt.subplot(211)
plt.plot(lag_acf)
plt.axis([0,48,-.5,1])
plt.axhline(y=0,linestyle='--',color='gray')
plt.axhline(y=-1.96/np.sqrt(len(df)),linestyle='--',color='gray')
plt.axhline(y=1.96/np.sqrt(len(df)),linestyle='--',color='gray')
plt.title('Autocorrelation Function')
plt.tight_layout()

#plot pacf
plt.subplot(212)
plt.plot(lag_pacf)
plt.axhline(y=0,linestyle='--',color='gray')
plt.axhline(y=-1.96/np.sqrt(len(df)),linestyle='--',color='gray')
plt.axhline(y=1.96/np.sqrt(len(df)),linestyle='--',color='gray')
plt.title('Partial Autocorrelation Function')
plt.tight_layout()
```

9.5

```
N_week_train = 3
train_len = 24 * 7 * N_week_train
N_week_test = 1
test_len=24 * 7 * N_week_test
```

9.6

```
X=df.values.astype(float)
predictions=np.zeros(test_len)
d=0; q=2; p=2
train, test=X[-(test_len+train_len):-test_len], X[-test_len:]
history=[x for x in train]
for t in range (0,test_len):
    model=ARIMA(history, order=(p,d,q))
    model_fit=model.fit()
    output=model_fit.forecast()
    yhat=output[0]
    predictions[t]=yhat
    obs=test[t]
    history.append(obs)
    history=history[1:]
```

9.7

```
orders=(0,1)
d=0;
X=df.values.astype(float)
N_train=3
train_len=24*7*N_train
print("training on ",train_len)
N_test=1
test_len=24*7*N_test
predictions_complete=[]
d=0
lag_orders=(0,1,2,3,4)
MA_orders=(0,1,2,3)
for p in lag_orders:
    predictions=np.zeros((len(MA_orders),test_len))
    for q in MA_orders:
        print('Testing ARIMA order (%i,%i,%i)' %(p,d,q))
        train, test=X[-(test_len+train_len):-test_len], X[-test_len:]
        history=[x for x in train]
        #history = [x for x in np.diff(train)]

        for t in range (0,test_len):
            #model=ARIMA(history, order=(p,d,q))
            model = ARIMA(history, order=(p, d, q), enforce_stationarity=False,
                enforce_invertibility=False)

            model_fit=model.fit(method='statespace')
            output=model_fit.forecast()
            yhat=output[0]
            predictions[MA_orders.index(q)][t]=yhat
            obs=test[t]
            history.append(obs)
            history=history[1:]

    predictions_complete.append(predictions)

#print(predictions_complete)

for p in range(len(lag_orders)):
    for q in range(len(MA_orders)):
        print('(%i,%i,%i)  MAPE: %.3f' % (p,d,q,mean_absolute_error(test, predictions_complete[p
            ][MA_orders.index(q)]) / test.mean()*100) )

predictions_complete_array = np.array(predictions_complete)
mape_values = []
mse_values = []
r2_values = []
for p in range(len(lag_orders)):
    for q in range(len(MA_orders)):
        mape_values.append(mean_absolute_error(test, predictions_complete_array[p, MA_orders.
            index(q)]) / test.mean() * 100)
        mse_values.append(mean_squared_error(test, predictions_complete_array[p, MA_orders.index(
            q)]))
        r2_values.append(r2_score(test, predictions_complete_array[p, MA_orders.index(q)]))

shape = (len(lag_orders), len(MA_orders))

# Reshape the MAPE values to match the shape of orders
mape_matrix = np.array(mape_values).reshape(shape)

# Create a heatmap
plt.figure(figsize=(10, 8))
sns.heatmap(mape_matrix, annot=True, fmt=".4f", cmap="YlGnBu",
            xticklabels=MA_orders, yticklabels=lag_orders)
plt.xlabel('q')
plt.ylabel('p')
plt.title('MAPE Heatmap for ARIMA Orders')
plt.show()

MA_orders=(0,1,2,3)
plt.plot(test,color='black',label='Orig')

p=2; d=0; q=1
X=df.values.astype(float)
N_test=1
test_len=24*7*N_test
predictions=np.zeros(test_len)
mape_list = []
r2_list = []
n = []
```

```
for train_len in range(24, len(df)-test_len, 24):
    #print('Testing ARIMA order (%i,%i,%i)' %(p,d,q))
    train, test=X[-(test_len+train_len):-test_len], X[-test_len:]
    history=[x for x in train]
    print(f"testing with N = {train_len}")
    for t in range (0,test_len):
       model = ARIMA(history, order=(p, d, q), enforce_stationarity=False,enforce_invertibility=
           False)
        model_fit=model.fit(method='statespace')
        output=model_fit.forecast()
        yhat=output[0]
        predictions[t]=yhat
        obs=test[t]
        history.append(obs)
        history=history[1:]
    mape_list.append(mean_absolute_error(test, predictions) / test.mean()*100)
    r2_list.append(r2_score(test, predictions))
    n.append(train_len)

plt.plot(n,mape_list)
plt.xlabel('lags')
plt.title('MAPE sliding - Munchen')
plt.grid(alpha=0.5)
plt.show()
```

9.8

```
q=1
d=0;
p=2
print("training on  ", train_len)
predictions=[]
h=24

train, test=X[-(test_len+train_len):-test_len], X[-test_len:]
history=[x for x in train]
for t in range (0,test_len):

    model=ARIMA(history, order=(p,d,q))
    model_fit=model.fit()
    output=model_fit.forecast(steps=h)
    yhat=output
    predictions.append(yhat)
    obs=test[t]
    history.append(obs)
    history=history[1:]
pred = []
h = [1, 3, 18, 24]
for i in h:
    j = i-1
    current = []
    for t in range(0,j):
        current.append(np.nan)
    for t in range(0,len(predictions)):
        current.append(predictions[t][j])
    pred.append(current)
t = list(df.index)[-len(test):]

plt.figure(figsize=(15,5))
plt.title(f"Prediction with different time horizons - ARIMA({p}, {d}, {q})", fontsize = 15)
plt.plot(list(df.index)[-len(test):], test,
        list(df.index)[-len(pred[0]):], pred[0],
        list(df.index)[-len(pred[1]):], pred[1],
        list(df.index)[-len(pred[2]):], pred[2],
        list(df.index)[-len(pred[3]):], pred[3])
plt.grid(alpha = 0.5)

plt.legend(labels =["observation", "h = 1", "h = 3", "h = 18", "h = 24"], fontsize = 12)

def predict_ARIMA_h24(X, p):
    q=1
    d=0;
    print("training on  ", train_len)
    predictions=[]
    h=24

    train, test=X[-(test_len+train_len):-test_len], X[-test_len:]
    history=[x for x in train]
    for t in range (0,test_len):

        model=ARIMA(history, order=(p,d,q))
        model_fit=model.fit()
        output=model_fit.forecast(steps=h)
        yhat=output
        predictions.append(yhat)
        obs=test[t]
        history.append(obs)
        history=history[1:]
    pred = []
    h = [1, 3, 18, 24]
    for i in h:
        j = i-1
        current = []
        for t in range(0,j):
            current.append(np.nan)
        for t in range(0,len(predictions)):
            current.append(predictions[t][j])
        pred.append(current)
    t = list(df.index)[-len(test):]
    plt.figure(figsize=(15,5))
    plt.title(f"p = {p}")
    plt.grid(alpha = 0.5)
    plt.plot(t, test,
        t, pred[0][:len(t)],
        t, pred[1][:len(t)],
        t, pred[2][:len(t)],
        t, pred[3][:len(t)])
    plt.legend(labels =["observation", "h = 1", "h = 3", "h = 18", "h = 24"], fontsize = 12,loc='
        upper right'),
    plt.show()

for p in [4, 24]:
    predict_ARIMA_h24(X, p)
```