ICT for Smart Mobility

Lab 2

Group1

Kousha Nikkar | 309359

Amirhossein Shekari | 309094

Mohammadsajjad Madadian | 307789

January 2024

1.Introduction

The main objective of this laboratory is to create the Autoregressive Integrated Moving Average (ARIMA) model to predict time series values with an acceptable error.

The ARIMA model has 3 main parameters (p,d,q) which must be tuned and determined according to the data. Moreover, we must decide whether to use a sliding window method or an expanding window for training the model and the suitable size for that training window (N).

We again use the data of the MongoDB carsharing database, and the time series by definition is the number of the bookings in different hours, for a 30 days time period (to be determined) within a specific city. We develop the model for 3 different cities (Torino,Hamburg and Montreal) and the horizon for the prediction is 1 hour.

2.Methodology

2.1 Period selection for time series

One of the main assumptions of an ARIMA model is to have a continuous time series with no missing value in the training data and the missing values must be compensated by different methods like averaging. Moreover, this model has the best performance when the data is stationary (has no trend and d=0). To keep the model simple, with less parameters (d=0), it is preferred to choose a period of time when data has no significant trend.

Upon analyzing the data for various months in the database, it becomes clear that there are several periods with missing values within each month, some of which span long durations without any bookings. Any attempts to adjust for these long period missing values could potentially harm the future precision of the model.

Finally, despite the mild trend in January's data due to the holiday, we chose to focus on this period in 2018 as there were the least missing values (Almost no hours without a booking). Consequently, we created a time series of bookings for the 720 unique hours from 01/01/2018 to 31/01/2018. This evaluation procedure has the same result for all the 3 cities.

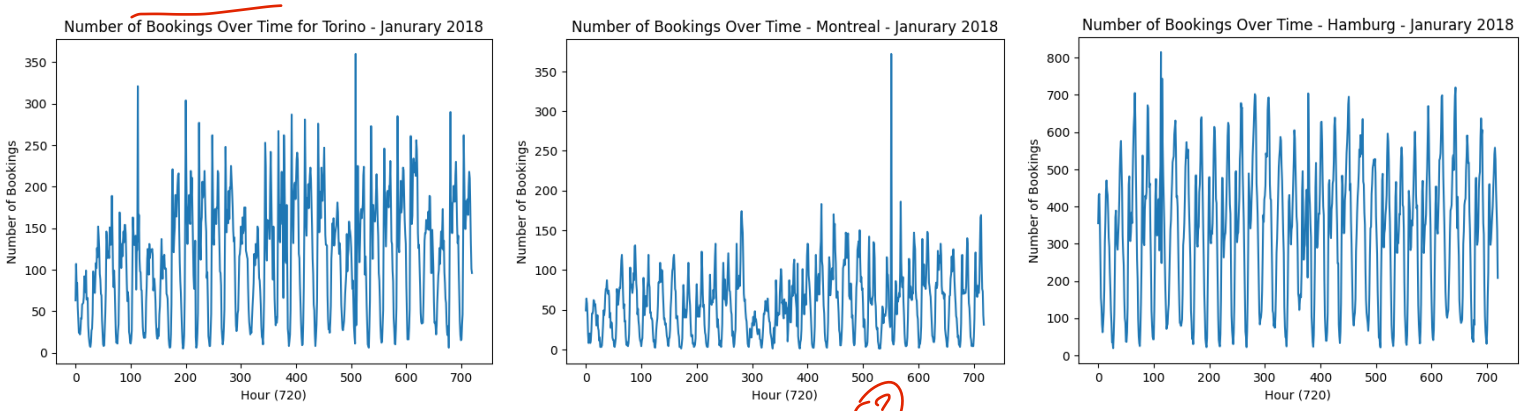


Figure1 - Number of Bookings (before removing outliers) Over different hours for Torino, Montreal and Hamburg - From 01-01-2018 00:00 to 31-01-2018 00:00

2.1 Preprocessing

Prior to creating the time series, to have reliable data, we removed unrealistic bookings with very long (longer than 10 hours) or very short (shorter than 3 minutes) durations. Then we created the time series by summing the number of bookings in each hour and plotted it in figure 1, but the time series data exhibits outliers, characterized by an unusually high or low number of bookings

you did this analysis already in Lab 1 - why redoing it again?

per hour .In our analysis, outliers are now defined as values that fall outside 1.5 times the interquartile range from the quartiles. These outliers are replaced with the mean of their adjacent samples, refining the data for further analysis..

2.2 Checking for stationariness (determining d value)

As mentioned before, the ARIMA model works with stationary data with no trend and if the data is not stationary, we must remove the trend using methods like differencing. In ARIMA, 'd' represents the order of differencing. The value of d for a stationary time series is equal to zero and the model will be simpler and easier to tune in this case. To check the stationariness, we use rolling mean and rolling standard deviation for 24 hours and 168 hours(1week) windows, which are suitable metrics for revealing a trend in any series of data and particularly in time series. We plotted the values for Torino as a sample of the procedure as Montreal and Hamburg showed a similar pattern. (You can find their plottings in the appendix 2.1)

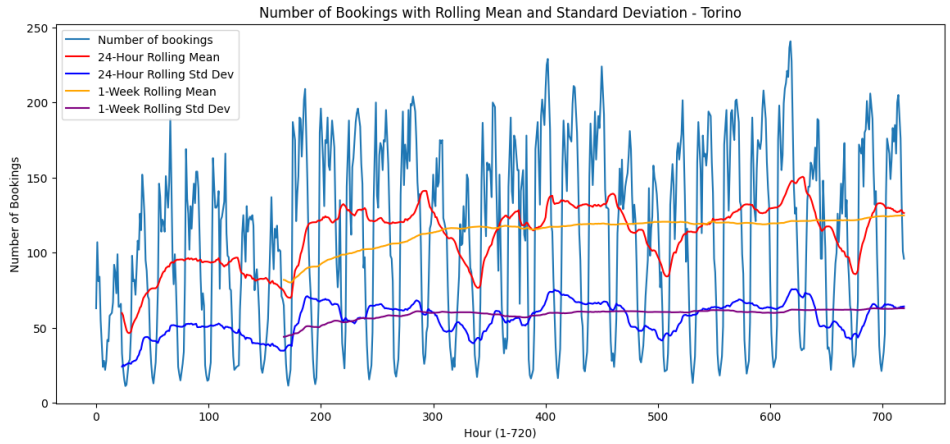


Figure 2 - The 24-hour and weekly rolling averages, along with rolling standard deviations, for the Torino booking numbers across various hours from 01-01-2018 00:00 to 31-01-2018 00:00.

Figure 2 indicates that after the first week of January, there isn't a significant trend in the daily and weekly rolling mean and standard deviation. Therefore, to keep the model simple, we will assume a value of 0 for 'd' (ARMA model) and exclude the first week in the model training phase.

do you discard the first week of Jan then?

2.2 Calculating ACF and PACF (determining an initial p and q values)

To have an accurate model we need to tune the values of p and q which are the order of AutoRegressive part and Moving Average part, respectively. We will do this by first calculating the ACF and PACF of the time series which can help us to understand the range of possible values for q and p respectively. (We will train the model for different values of p and q in the future to determine the best combination) We will calculate ACF and PACF only for the past 48 samples (2days), as the correlation of data to samples before that probably would converge to zero.

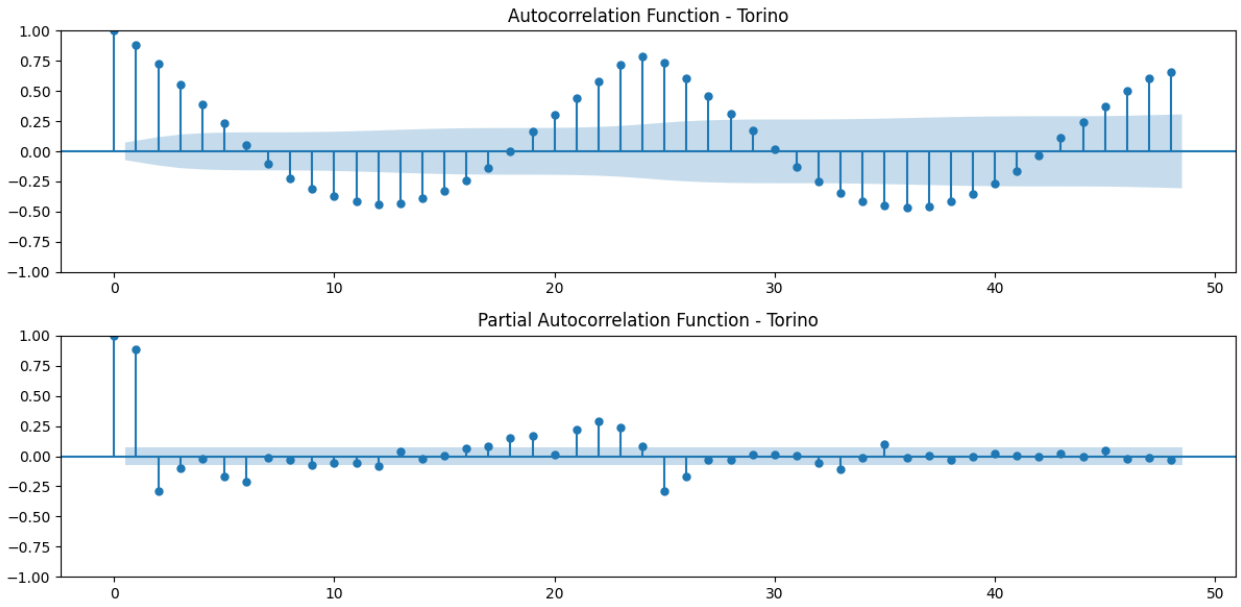


Figure 3 - Plotting the values of the autocorrelation function (ACF) and partial autocorrelation function(PACF) of booking number data for Torino's across 01-01-2018 00:00 to 31-01-2018 00:00. You can find the similar plot for Hamburg and Montreal in the appendix 2.2.

The possible values for p, are 0, 1 or 2 as the PACF for the future values goes to near zero values. A similar set of patterns of ACF and PACF derived with the same values of possible p value (2) for both Montreal and Hamburg. Moreover a cycling pattern of each 24 samples can be seen in ACF as the number of bookings in each day hour is similar to its past day value. In general, q is harder to determine as MA model relies on the past errors which are not observable. We decided the same range of possible values of p for q as ACF gives us no information about the best possible value for q. We try to keep the model as simple as possible. (lower values of p and q). For now, it is not possible to determine whether the model is purely AR or MA and therefore we have to tune the model with a grid search over the possible values for these parameters. p,q $\in$ [0,3]. (We will also consider p,q=3, hoping for a better accuracy)

2.3 ARIMA model training and calculation of residual for different number of N

To predict using the ARIMA model, we use a walk forward sliding window and implement it using an iterative approach. In each iteration we train a new model, predict the next point, calculate the error by comparing it to the test actual dataset and then we retrain the model and redo the process by including the most recent predicted point in the new training set.

To determine the suitable value of N, we train the model with different training windows over 3 different N_train values (48,168,336) starting from the second week of January to exclude the first week with a mild trend and then we test it for N_test values (24,84,168), half of the training window size, respectively. For this step, we fit the ARIMA model with parameters p=2, d=0, and q=2.(value of p is chosen based on PACF and the value of q is chosen arbitrarily). Then we compute the RMSE (Root Mean Square Error) and MAPE (Mean Absolute Percentage Error) for each of these models as RMSE emphasizes large errors and MAPE provides a relative error measure which is useful for between-city comparisons. To understand prediction precision, we calculate residuals as the difference between predicted and actual data, and plot their density (Figure 4) knowing that a good residual plot shows residuals normally distributed around zero.

is the test period the same (?)

We repeat this procedure for the 3 cities to detect the suitable value of N for each of them.The error calculations are presented in table 1.

	48 H train - 24 H test		168 H train - 84 H test		336 H train - 168 H test	
City/Index	**MAPE**	**RMSE**	**MAPE**	**RMSE**	**MAPE**	**RMSE**
Torino	35.405	35.405	28.683	29.283	21.733	25.616
Hamburg	26.175	59.466	28.568	92.734	23.035	56.192
Montreal	46.821	18.204	105.237	24.538	63.358	20.248

Table 1 - MPE and MAPE of ARIMA model residuals for varying number of N of the data related to Torino, Hamburg and Motreal with p=2,d=0,q=2

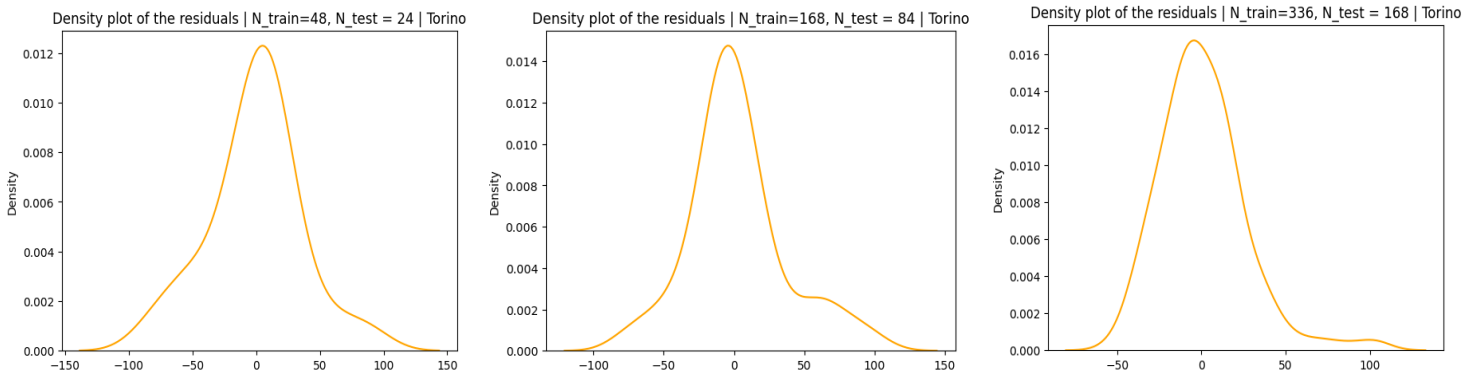


Figure 4 - Plotting the ARIMA model density of residuals for Torino's data across 01-01-2018 00:00 to 31-01-2018 00:00.

Table 1 and figure 4 show that for both Torino and Hamburg, N=336 shows the best accuracy with the least values of errors and a narrower density plot of residuals so we choose N= 336 (A two week training window) for these 2 cities and for Montreal N=48 is the best option.

2.5 Performing grid search

In order to determine the best parameters (p,q) for the ARIMA model, we execute a grid search across various combinations of p and q values. Although we derive p=2 in 2.2 for all the 3 cities, here we opted for p=3 and q=3, despite the increased computation time, hoping for a more accurate model.

This is done using a walk forward sliding window method (N_train,test =336,168 for Torino and Hamburg and N_train,test= 48,24 for Montreal based on table 1), excluding the initial week of January. For each combination, we compute the MAPE to assess the model's effectiveness. The optimal combination is the one with the smallest error. Figure 6 visually depicts these errors through a heatmap.

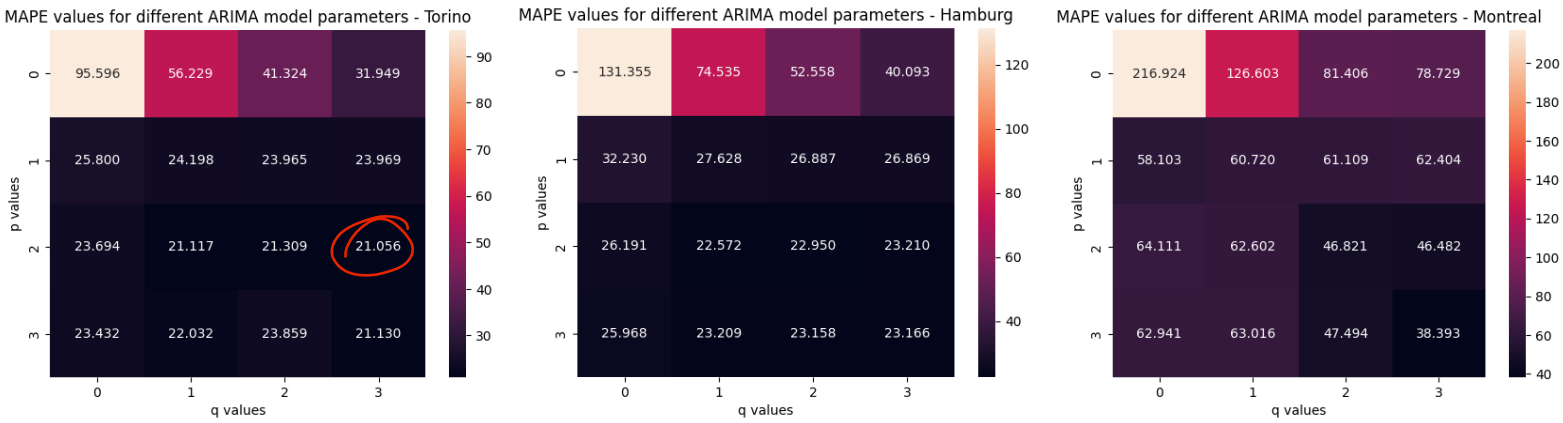


Figure 6 -MAPE heatmap for different combination of p and q values for the cities (sliding window walk forward)

As the figure suggests, Optimal model parameters for Torino is p=2,d=0,q=3, Hamburg is p =2, d=0 , q = 1 and Montreal is p = 3, d=0 , q=3)

2.4 ARIMA model training with a sliding and expanding window (walk forward)

For the sliding window method, the size of the training data remains fixed, as we remove the oldest data in the training set, and a window (of the determined size of 336) will slide over the training dataset, in the latter approach, the window (with an initial size of 336) grows as we do not remove the oldest point in each iteration. In both methods, we skip January's first week due to a non-zero trend, conflicting with our intended d=0 model parameter. The values of p and q for each city is determined based on figure 6 values derived from the grid search.The error calculations are presented in table 2.

	Sliding window		Expanding window	
City / Index	RMSE	MAPE	RMSE	MAPE
Torino	25.428	21.056	27.088	22.952
Hamburg	59.589	22.572	69.483	22.554
Montreal	20.278	37.798	21.148	38.393

Table 2 - Comparing MAPE and RMSE of the methods for Torino, Hamburg and Montreal

Table 2 does not show a significant difference between a sliding and expanding window approach. Therefore, we will make an arbitrary choice of sliding window approach for doing the grid search as it will be more memory saving and faster to train the model on fewer numbers of the samples.

	Number of standard rentals	MAPE
Torino	83246	21.056
Hamburg	241257	22.572
Montreal	40977	38.393

Table 3 - Comparing the number of rentals (after removing outliers with very short or long durations) with the least possible MAPE for the 3 cities

As we can observe from the table, when the dataset is limited, such as the total standard number of bookings in Montreal with approximately 40,000 bookings, the Mean Absolute Percentage Error (MAPE) approaches 38% , indicating that the estimates appear not to be very accurate.

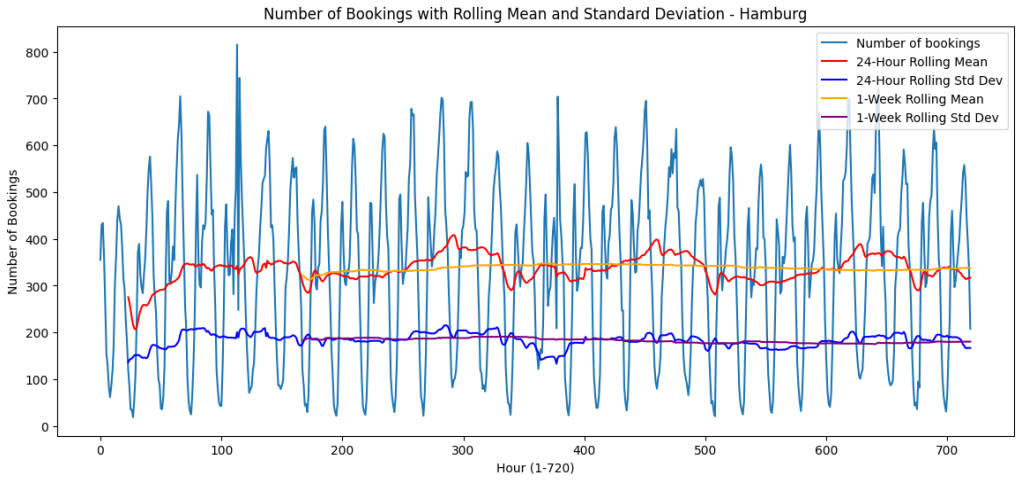
- OK
- some comments are not clear
- walk forward on different test sets
- Horizon = 1 only
- no comparison with baseline

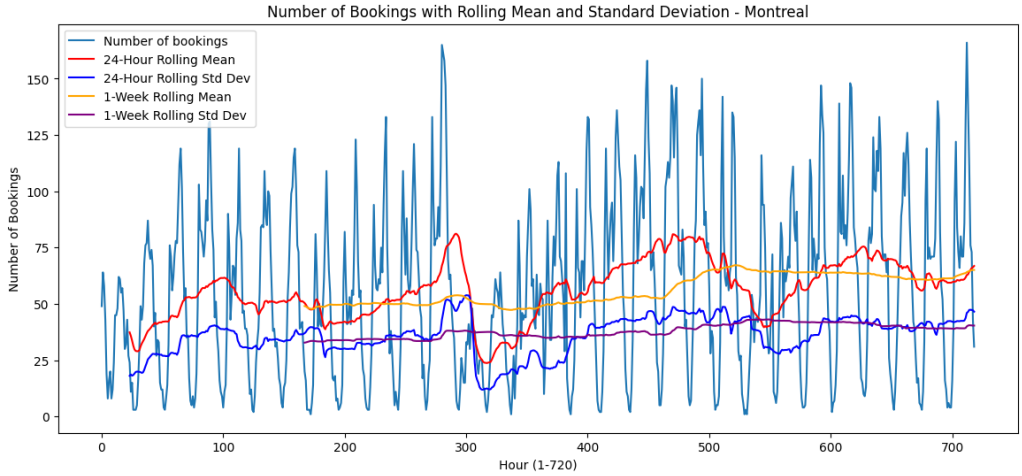
4-/5

Appendix

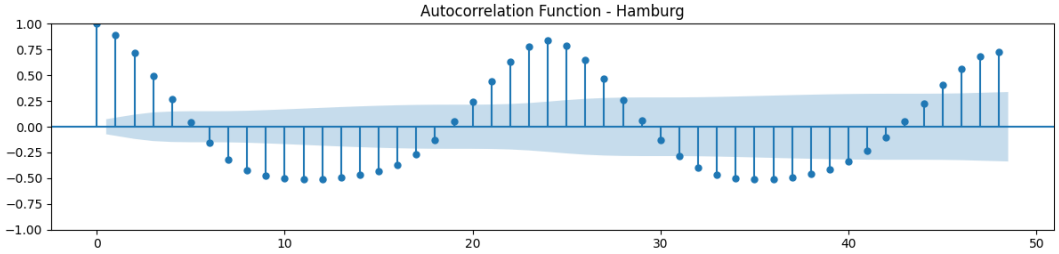
1-Figures and Tables:

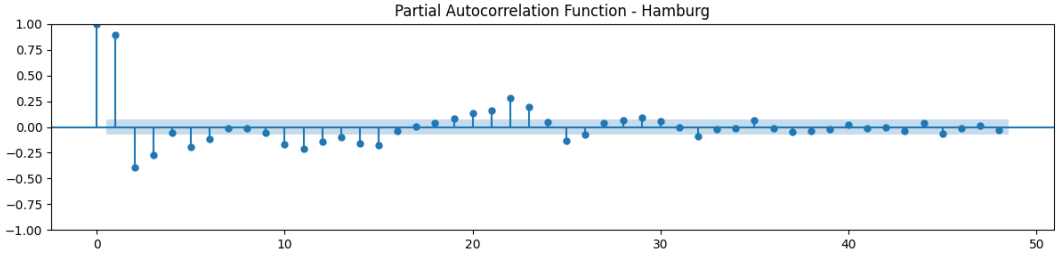
1.1

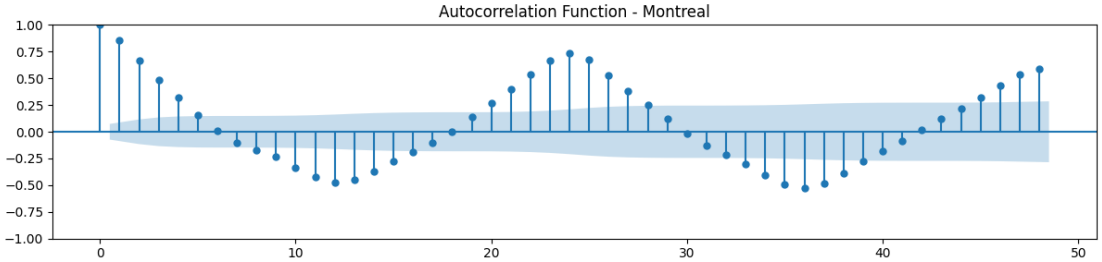


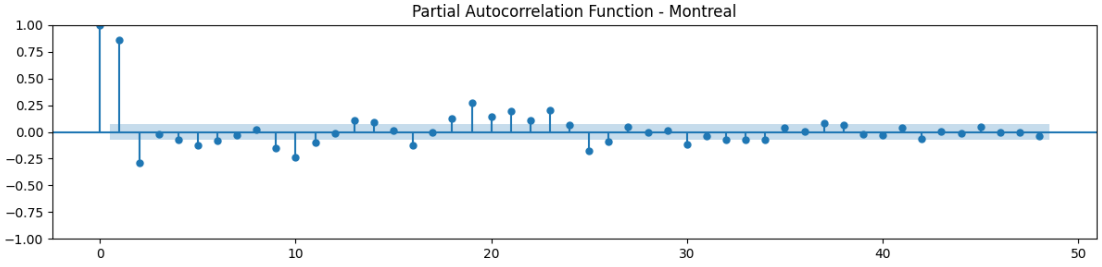

Number of Bookings with Rolling Mean and Standard Deviation - Montreal
Number of bookings
24-Hour Rolling Mean
24-Hour Rolling Std Dev
1-Week Rolling Mean
1-Week Rolling Std Dev
Number of Bookings
Hour (1-720)

1.2

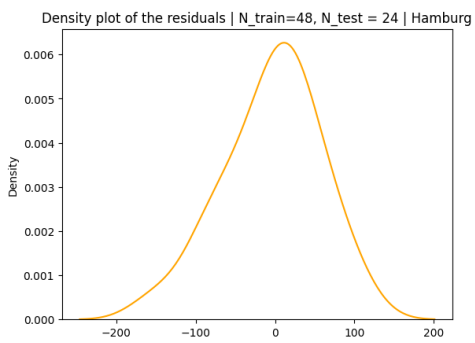

Autocorrelation Function - Hamburg

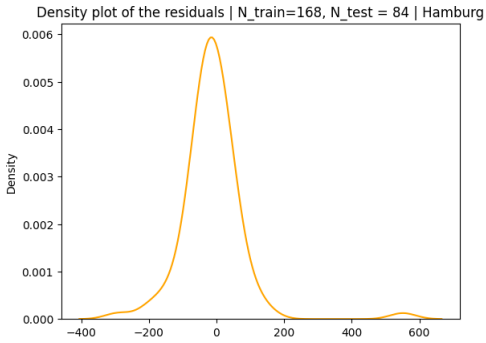

Partial Autocorrelation Function - Hamburg

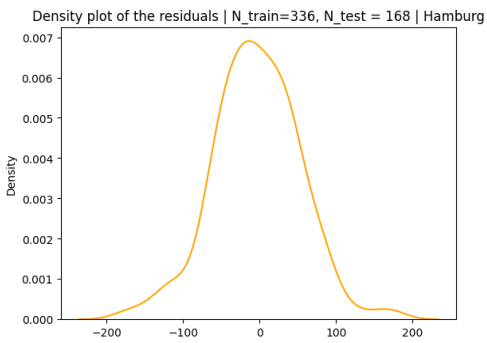

Autocorrelation Function - Montreal

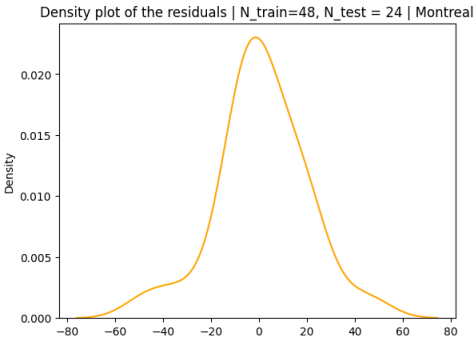

Partial Autocorrelation Function - Montreal

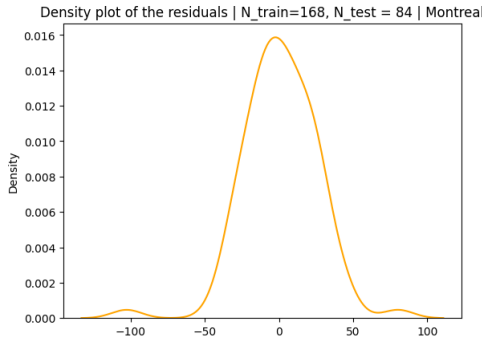
1.3

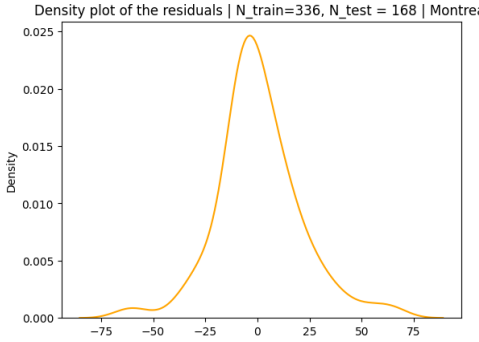












2- Codes:

```
pip install pymongo

#||||| Connection to the client and importing necessary libraries

from pymongo import MongoClient
import matplotlib.pyplot as plt
from datetime import datetime
import pandas as pd
from pandas import datetime
from pandas import DataFrame
from sklearn.metrics import mean_squared_error
from math import sqrt
import numpy as np

# Connection parameters
username = "ictts"
password = "Ict4SM22!"
database = "carsharing"
server = "bigdatadb.polito.it"
port = 27017

# Create a connection string
```

```
connection_string =
f"mongodb://{username}:{password}@{server}:{port}/{database}?authMechanism
=SCRAM-SHA-1&ssl=true&tlsAllowInvalidCertificates=true"

# Create a new MongoClient instance
client = MongoClient(connection_string)

# Access the database
db = client[database]

print(db.list_collection_names())


#||||| Get the data for the jan 2018 as most of the other monthes in the
collection have some days of missing values

# Define the date range
start_date = datetime(2018,1, 1)
end_date = datetime(2018, 1, 31)

# Query the PermanentBooking collection
query = {"city": "Hamburg", "init_date": {"$gte": start_date, "$lte":
end_date}}
documents = db.PermanentBookings.find(query)

# convert to dataframe
rawdata = pd.DataFrame(list(documents))
rawdata


#||||| Removing the data with very long or short durations (larger than 10
hours and less than 3 minutes) - outlier removal

# Calculate the duration in seconds
rawdata['duration'] = (rawdata['final_date'] -
rawdata['init_date']).dt.total_seconds()

# Filter out the records where the duration is larger than 10 hours or
less than 2 Minures
filtered_data = rawdata[(rawdata['duration'] <= 36000) &
(rawdata['duration'] >= 120)]
```

```
# Removing data with the same init and final address that have a less than
10 minute duration
filtered_data = filtered_data[~((filtered_data['init_address'] ==
filtered_data['final_address']) & (filtered_data['duration'] < 300))]


# Print the number of records before and after filtering
print(rawdata.shape[0])
print(filtered_data.shape[0])

#||||| Preserving only the needed columns

filtered_data = filtered_data[['city', 'init_date', 'final_date']]
filtered_data

#||||| Counting the number of booking per hour

# Extract the date and hour from init_date
filtered_data['date'] = filtered_data['init_date'].dt.date
filtered_data['hour'] = filtered_data['init_date'].dt.hour

# Group by city, date, and hour, and count the number of bookings
booking_counts = filtered_data.groupby(['city', 'date',
'hour']).size().reset_index(name='number_of_bookings')


print(booking_counts.shape[0])
booking_counts

import matplotlib.pyplot as plt

# Filter the data for the city "Torino"
city_data = booking_counts[booking_counts['city'] == 'Hamburg']

# Create a figure and axes
fig, ax = plt.subplots()

# Plot the data
ax.plot(city_data['number_of_bookings'])

# Set the title and labels
ax.set_title('Number of Bookings Over Time - Hamburg - Janurary 2018')
ax.set_xlabel('Hour (720)')
```

```
ax.set_ylabel('Number of Bookings')

# Show the plot
plt.show()

#|||||| Replacing outliers (values that fall outside 1.5 times the
interquartile range from the quartiles.)

# Define a function to calculate the lower and upper bound
def calculate_bounds(data):
    Q1 = data.quantile(0.25)
    Q3 = data.quantile(0.75)
    IQR = Q3 - Q1
    lower_bound = Q1 - 1.5 * IQR
    upper_bound = Q3 + 1.5 * IQR
    return lower_bound, upper_bound

# Calculate the lower and upper bound
lower, upper = calculate_bounds(booking_counts['number_of_bookings'])

# Identify the indices of the rows with 'number_of_bookings' outside the
bounds
outlier_indices = booking_counts[(booking_counts['number_of_bookings'] <
lower) | (booking_counts['number_of_bookings'] > upper)].index

# For each outlier, compute the mean of the previous and next value, and
replace the outlier with this mean
for idx in outlier_indices:
    if idx > 0 and idx < len(booking_counts) - 1:
        mean_value = np.mean([booking_counts.loc[idx - 1,
'number_of_bookings'], booking_counts.loc[idx + 1, 'number_of_bookings']])
        booking_counts.loc[idx, 'number_of_bookings'] = mean_value


#|||||| Checking if the time series is stationary or not to take the
required actions


# Plot the time series
plt.figure(figsize=(14, 6))
plt.plot(booking_counts['number_of_bookings'], label='Number of bookings')
```

```
# Calculate and plot the rolling mean and standard deviation with a window
of 24 hours
rolling_mean_24 =
booking_counts['number_of_bookings'].rolling(window=24).mean()
rolling_std_24 =
booking_counts['number_of_bookings'].rolling(window=24).std()
plt.plot(rolling_mean_24, color='red', label='24-Hour Rolling Mean')
plt.plot(rolling_std_24, color='blue', label='24-Hour Rolling Std Dev')

# Calculate and plot the rolling mean and standard deviation with a window
of 1 week (168 hours)
rolling_mean_168 =
booking_counts['number_of_bookings'].rolling(window=168).mean()
rolling_std_168 =
booking_counts['number_of_bookings'].rolling(window=168).std()
plt.plot(rolling_mean_168, color='orange', label='1-Week Rolling Mean')
plt.plot(rolling_std_168, color='purple', label='1-Week Rolling Std Dev')

# Add labels and title
plt.xlabel('Hour (1-720)')
plt.ylabel('Number of Bookings')
plt.title('Number of Bookings with Rolling Mean and Standard Deviation -
Hamburg')

# Add a legend
plt.legend(loc='best')

# Show the plot
plt.show()

#||||||| Determining p and q (AR and MA hyperparameters)

from statsmodels.graphics.tsaplots import plot_acf, plot_pacf
from statsmodels.graphics.tsaplots import plot_acf, plot_pacf

# Create a series with 'number_of_bookings' values indexed by date and
hour
time_series = booking_counts.set_index(['date',
'hour'])['number_of_bookings']

fig, ax = plt.subplots(2, figsize=(12,6))

# Plot the autocorrelation function
```

```
plot_acf(time_series, ax=ax[0], lags=48)
ax[0].set_title('Autocorrelation Function - Hamburg')

# Plot the partial autocorrelation function
plot_pacf(time_series, ax=ax[1], lags=48)
ax[1].set_title('Partial Autocorrelation Function - Hamburg')

plt.tight_layout()
plt.show()

#|||||| ARIMA model training

import seaborn as sns
from statsmodels.tsa.arima.model import ARIMA
# evaluate an ARIMA model using a walk-forward validation

# load dataset (assuming you have already loaded it as 'booking_counts')
X = booking_counts['number_of_bookings']

# Training over a 48 samples period and testing over 168 samples
train_size = 48
test_size = 24
train, test = X[168:168+train_size],
X[168+train_size:168+train_size+test_size]
history = [x for x in train]
predictions = list()

for t in range(len(test)):
    model = ARIMA(history, order=(2,0,2))
    model_fit = model.fit()
    output = model_fit.forecast()
    yhat = output[0]
    predictions.append(yhat)
    obs = test.iloc[t]
    history.append(obs)
    # SLIDING WINDOW
    history.pop(0)
    #print('predicted=%f, expected=%f' % (yhat, obs))

# evaluate forecasts by RMSE
rmse = sqrt(mean_squared_error(test, predictions))
print('Test RMSE: %.3f' % rmse)
```

```
# calculate MAPE
mape = np.mean(np.abs((np.array(test) - np.array(predictions)) /
np.array(test))) * 100
print('Test MAPE: %.3f' % mape)

# calculate residuals
residuals = [test.iloc[i]-predictions[i] for i in range(len(test))]

# plot density of residuals
plt.figure()
sns.kdeplot(residuals, fill=False , color='orange')
plt.title("Density plot of the residuals | N_train=48, N_test = 24 |
Hamburg")
plt.show()

#||||| Performing grid search

import numpy as np
import seaborn as sns
from statsmodels.tsa.arima.model import ARIMA
from sklearn.metrics import mean_squared_error
from math import sqrt

# load dataset (assuming you have already loaded it as 'booking_counts')
X = booking_counts['number_of_bookings']
week = 168
#Training over a one week period
train, test = X[week:3*week], X[3*week:len(X)] # Skipping the first week
for training as we seen some trend

# define p, d and q values to take on any value between 0 and 2
p_values = range(0, 4)
q_values = range(0, 4)
d_values = [0]  # assuming d is fixed at 0

# create dictionaries to store the RMSE and MAPE values for each (p,q)
combination
rmse_values = {}
mape_values = {}

for p in p_values:
    for q in q_values:
        history = [x for x in train]
```

```
        predictions = list()
        # walk-forward validation (sliding window by appending and popping
the observed value)
        for t in range(len(test)):
            model = ARIMA(history, order=(p,0,q))
            model_fit = model.fit()
            output = model_fit.forecast()
            yhat = output[0]
            predictions.append(yhat)
            obs = test.iloc[t]
            history.append(obs)
            # Sliding Window
            history.pop(0)
        # evaluate forecasts by RMSE
        rmse = sqrt(mean_squared_error(test, predictions))
        rmse_values[(p, q)] = rmse
        # calculate MAPE
        mape = np.mean(np.abs((np.array(test) - np.array(predictions)) /
np.array(test))) * 100
        mape_values[(p, q)] = mape

# convert the dictionaries to matrices for heatmap plotting
rmse_matrix = np.array([[rmse_values[(p, q)] for q in q_values] for p in
p_values])
mape_matrix = np.array([[mape_values[(p, q)] for q in q_values] for p in
p_values])

# plot heatmaps
sns.heatmap(rmse_matrix, annot=True, fmt=".3f", xticklabels=q_values,
yticklabels=p_values)
plt.title("RMSE values for different ARIMA models parameters - Hamburg")
plt.xlabel("q values")
plt.ylabel("p values")
plt.show()

sns.heatmap(mape_matrix, annot=True, fmt=".3f", xticklabels=q_values,
yticklabels=p_values)
plt.title("MAPE values for different ARIMA model parameters - Hamburg")
plt.xlabel("q values")
plt.ylabel("p values")
plt.show()

#|||||||||| Detemining whether sliding window is better or Expanding
```

```
#||||| Sliding

# evaluate an ARIMA model using a walk-forward validation
from statsmodels.tsa.arima.model import ARIMA
# load dataset (assuming we have already loaded it as 'booking_counts')
X = booking_counts['number_of_bookings']
week = 168

#Training over a one week period
train, test = X[week:3*week], X[3*week:4*week] # Skipping the first week
for training as we seen some trend

history = [x for x in train]
predictions = list()

# walk-forward validation (sliding window by appending and popping the
observed value)
for t in range(len(test)):
    model = ARIMA(history, order=(2,0,1))
    model_fit = model.fit()
    output = model_fit.forecast()
    yhat = output[0]
    predictions.append(yhat)
    obs = test.iloc[t]
    history.append(obs)
    # SLIDING WINDOW
    history.pop(0)
    print('predicted=%f, expected=%f' % (yhat, obs))

# evaluate forecasts by RMSE
rmse = sqrt(mean_squared_error(test, predictions))
print('Test RMSE: %.3f' % rmse)

# calculate MAPE
mape = np.mean(np.abs((np.array(test) - np.array(predictions)) /
np.array(test))) * 100
print('Test MAPE: %.3f' % mape)

# convert predictions to a pandas series
predictions_series = pd.Series(predictions, index=test.index)

# plot forecasts against actual outcomes
```

```
plt.plot(test, label='Actual')
plt.plot(predictions_series, color='red', label='Predicted- sliding
window')
plt.title("Real values vs ARIMA predicted values using (sliding window) -
Torino")
plt.legend()
plt.show()

#||||| Expanding

# evaluate an ARIMA model using a walk-forward validation

# load dataset (assuming you have already loaded it as 'booking_counts')
X = booking_counts['number_of_bookings']
week = 168
#Training over a one week period
train, test = X[week:2*week], X[2*week:len(X)] # Skipping the first week
for training as we seen some trend
history = [x for x in train]
predictions = list()

# walk-forward validation (sliding window by appending and popping the
observed value)
for t in range(len(test)):
    model = ARIMA(history, order=(2,0,1))
    model_fit = model.fit()
    output = model_fit.forecast()
    yhat = output[0]
    predictions.append(yhat)
    obs = test.iloc[t]
    # Expanding WINDOW
    history.append(obs)
    print('predicted=%f, expected=%f' % (yhat, obs))

# evaluate forecasts by RMSE
rmse = sqrt(mean_squared_error(test, predictions))
print('Test RMSE: %.3f' % rmse)

# calculate MAPE
mape = np.mean(np.abs((np.array(test) - np.array(predictions)) /
np.array(test))) * 100
print('Test MAPE: %.3f' % mape)
```

```
# convert predictions to a pandas series
predictions_series = pd.Series(predictions, index=test.index)

# plot forecasts against actual outcomes
plt.plot(test, label='Actual')
plt.plot(predictions_series, color='orange', label='Predicted-Expanding
window')
plt.title("Real values vs ARIMA predicted values using (Expanding window)
- Torino")
plt.legend()
plt.show()
```