Lab 2 Report

Florin Roberto Bratu, s312156
Fabio Callerio, s314926
Stefano Viccari, s317139

ICT for Smart Mobility, A.Y. 2023/24
Politecnico di Torino

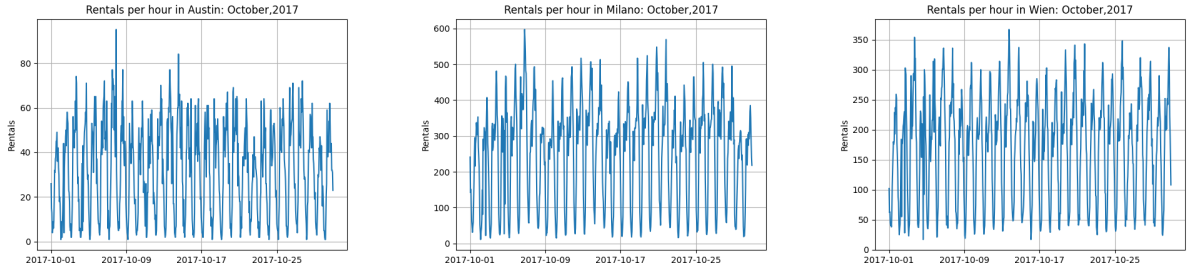
1 Data selection and Preprocessing

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

We selected as period the month of October 2017, since it doesn't contain any seasonal variation or holiday.

Outliers can be caused by canceled bookings, errors in the system, maintenance (a car moving to get maintenance is seen as a booking) and bookings with very unlikely big duration and so must be removed. To do so, as in first lab, we chose to consider only the bookings lasting more than 5 minutes, having different origin and destination addresses and, in order to remove outliers, we calculated mean and variance of the duration and excluded the bookings with duration less than the calculated value of mean plus two times the standard deviation.

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



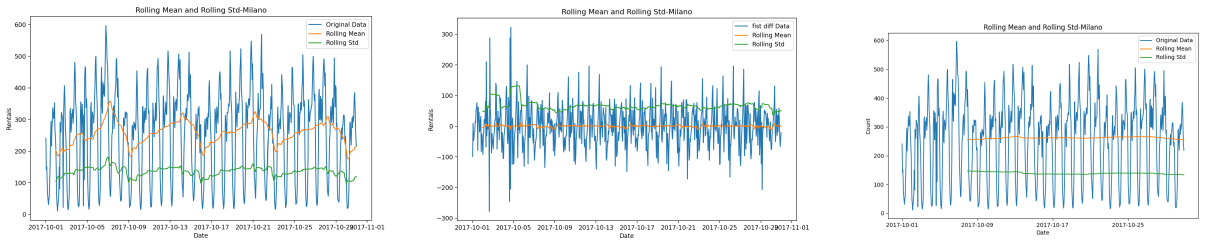
(a) Austin (b) Milano (c) Wien

Figure 1: Rentals per hour in the month of October,2017

Milano and Wien have 10 missing samples, while Austin 16. In order to fit them, we decided to replace data with the average value evaluated on all samples belonging to the same hour of each day of October. Doing so, we avoid focusing just on single weeks (i.e considering just the current week hours) and single day type (weekday or weekend day). In fact, "last value" solution assumes that the most recent observation is a good estimate for the missing value (but that may not always be the case) and "replacement with zeros" solution can significantly impact the mean and variance of a dataset, especially in a city like Milano with a lot of rentals.

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

For each of the 3 cities we plotted the differencing and we computed rolling statistics with sliding window both of 24 samples (24h) and $24 * 7$ samples (1 week). As we can see from Figures 2, 9, 10, for all the cases, first order differencing should be necessary since original dataset is apparently not stationary with a sliding window of 24h (Figures 2b, 3b, 4b), having both rolling mean and standard deviation variable. Instead, with the weekly solution we can notice that both mean and standard deviation are stationary for all three cities. So at the end we can consider the datasets as stationary, choosing as consequence $d = 0$.

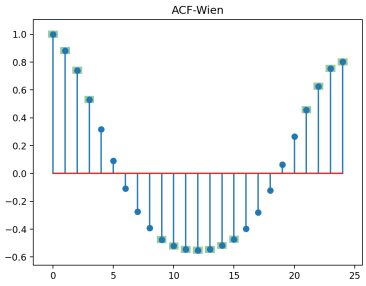


(a) rolling statistics on original dataset (b) rolling statistics on 1st ordered differenced data (c) rolling statistics on orginal dataset with weekly window

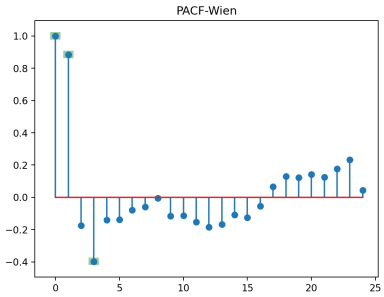
Figure 2: Differencing and rolling statistics on Milano's dataset

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

In Figure 3 and in Figures 11, 12 (can be find in section 'Appendix') , we plotted all the Auto-Correlation Functions (ACFs) and Partial Auto-Correlation Functions (PACFs), highlighting with a greed square the corresponding hours that have a ACF or PACF value above a threshold t, selected as: $t_{acf} = 0.4$; $t_{pacf} = 0.3$. Since ACFs don't really cut off or tail off and PACFs also don't show a clear pattern, the best solution is to consider our model as a potential ARMA, performing a grid search in order to find the optimal values of p and q.



(a) ACF - Wien



(b) PACF - Wien

Figure 3: ACF & PACF over hour of day for Wien's dataset

5. Decide the number of past samples N to use for training, and how many to use for testing.

We chose the period ranging from 1-10-2017 to 31-10-2017: we decided to take the last 7 days as testing set (168 samples), then the previous 14 days as training set (N = 336 samples). This decision was made by two criterias:

1. an empirical criteria: this division produced consistent results.
2. a convenience criteria: in case the length of the training needed to be tuned up, it was possible to do so without compromising the testing set, for example by further expanding the training backwards until reaching a maximum of three weeks.

6. Given N, (p,d,q), train a model, compute the error. Consider the MPE and/or MAPE and/or other metrics – so that you can compare results for different cities

We decided to test the order (3,2,2) using both expanding window and sliding window (by keeping N fixed to two weeks). Even though we chose d=2, the prediction is still acceptable since the MAPE is not that high, especially for the city of Wien, so we expect even better results with d equal to 0. As we can see in both Table 1 and 2 the model works

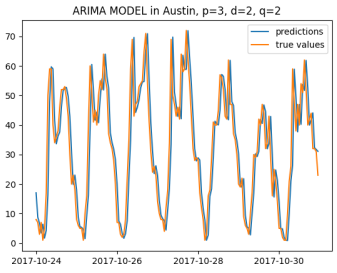
the worst in Austin, having the lowest MAE but the highest MAPE and lowest R2 score for both strategies. This happens because it's the city that after the filtering phase has the least number of bookings per hour on average, so even if the absolute error is small, the relative one is bigger than the other cities. We can also see that in every city the expanding window configuration works only slightly better for this order than the sliding window's one, but with a very small difference. However, Figure 4 shows that even in the worst case, the prediction is quite good, but not perfect, since the MAPE is high and probably the predictions are almost the values of some hours before, but shifted(Milano and Wien in appendix in Figure 13 and Figure 14).

city	MAE	MAPE	R2 score
Austin	7.44	22.84	0.74
Milano	39.28	18.08	0.75
Wien	25.20	14.70	0.868

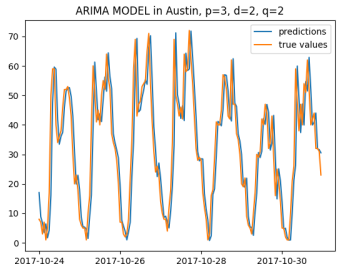
Table 1: Results of expanding window

city	MAE	MAPE	R2 score
Austin	7.47	22.93	0.73
Milano	40.99	18.87	0.74
Wien	25.19	14.69	0.87

Table 2: Results of sliding window



(a) expanding window Austin



(b) sliding window Austin

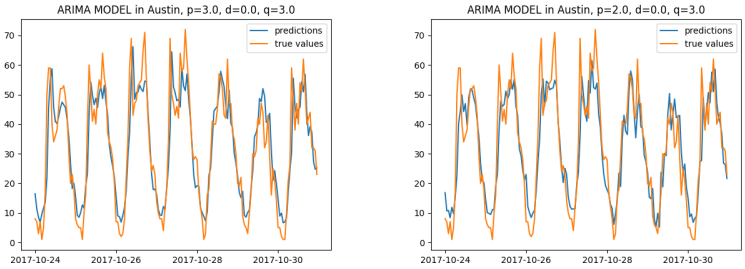
Figure 4: ARIMA order(3,2,2) prediction - Austin

2 Parameters Analysis

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

The grid we chose is composed as it follows: $p = [0, 1, 2, 3]; d = [0, 1, 2]; q = [0, 1, 2, 3]$. We did the grid search for both expanding and sliding window configuration. Figure 5 shows the prediction of last 7 days of October for the city of Austin. It's evident from Table 3 that the best parameters can change by the type of configuration we decide to use and based on the city; we can confirm the model is stationary since we obtain the best results

with d=0. Moreover, the ARIMA prediction doesn't work with some of the parameters(for example for p=3 and q=2 for the city of Austin), so we would need to consider not only the smallest MAPE, but also to try to keep the complexity of the model as low as possible, since the error does not change much by using bigger values of p and q.

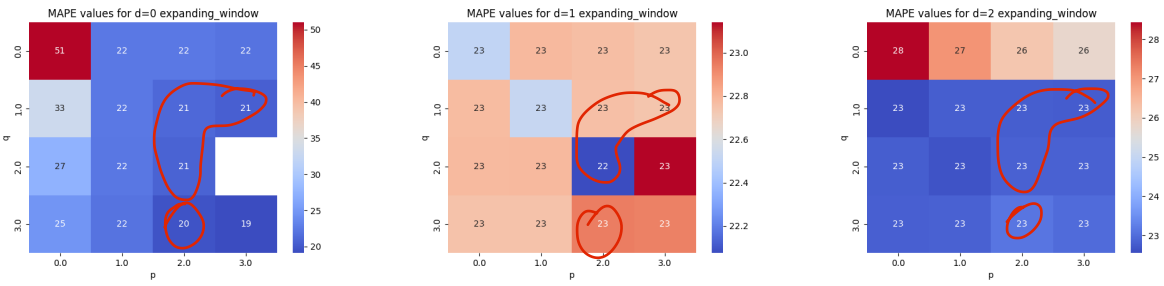


(a) expanding window (b) sliding window

Figure 5: Prediction of Austin last 7 days of October for expanding and sliding window after optimization with grid search

City	strategy	(p,d,q)	MAE	MAPE	R2 score
Milano	sliding window	(3,0,3)	39.83	15.64	0.85
	expanding window	(2,0,3)	40.23	15.8	0.85
Wien	sliding window	(3,0,1)	20.99	12.25	0.90
	expanding window	(2,0,1)	20.92	12.20	0.90
Austin	sliding window	(2,0,3)	6.35	19.48	0.80
	expanding window	(3,0,3)	6.25	19.18	0.81

Table 3: Results of the best configuration after tuning the order



(a) d = 0 (b) d = 1 (c) d = 2

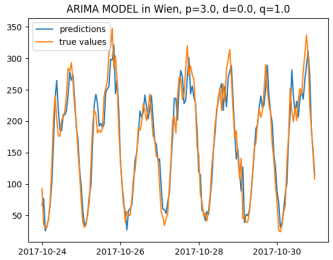
Figure 6: MAPE values for p,d,q with expanding window in Austin

7.b Given the best parameter configuration, change N and the learning strategy (expanding versus sliding window). Always keep the testing set on the

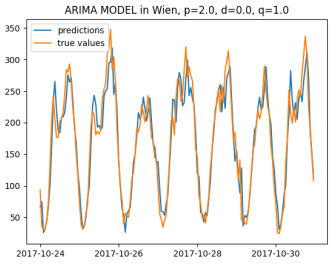
same portion of the data. For instance, if you use 1 week for testing, you can use from 1 day to 3 weeks for training.
The training set was tuned for each configuration in Table 3, choosing the parameters having the smallest MAPE, since in most cases we don't have many parameters and the model is not so complex. The procedure starts from a testing set that contains the samples of the last seven days of October, a training set that contains the samples of one day. We compute the model for each configuration, the errors and the R2 scores and for each strategy. After that we iterate again by adding the samples of the previous day to the training set, to see the difference by changing N, until there are no more days available in our dataset. We obtained, as a result, the optimal number of days(N) for the training set that minimizes the MAPE, for each configuration and for each city, all shown in Table 4. Comparing it with Table 3, we can see no particular improvement since the optimal values of N are quite closer to the value of N we already chose before (N=14, so two weeks is a good choice for the training set); this is shown also in Figure 7, where no clear difference can be seen since the parameters were closer to those chosen before.

City	strategy	days of train	MAE	MAPE	R2 score
Milano	sliding window	15	39.06	15.34	0.85
	expanding window	16	39.93	15.68	0.85
Wien	sliding window	15	20.90	12.19	0.90
	expanding window	16	20.91	12.20	0.90
Austin	sliding window	16	6.20	19.03	0.80
	expanding window	14	6.25	19.18	0.81

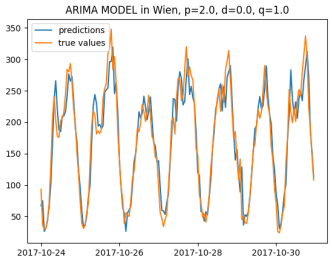
Table 4: Results of the absolute best configuration after tuning the training set



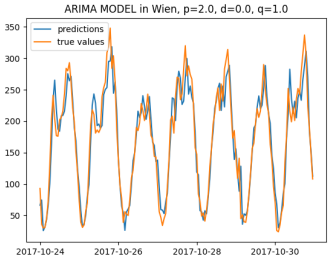
(a) Sliding window configuration with 14 days



(b) Sliding window configuration best number of days per training set, 15



(c) Expanding window configuration with 14 days



(d) Expanding window configuration best number of days per training set, 16

Figure 7: Comparison on Wien's last Week of October between before and after tuning the train

7.c Compare results for the different cities. How the relative error changes w.r.t. the absolute number of rentals?
Having a look at Figure 8, it is noticeable that Milano and Wien are the cities with the lowest MAPE values for each day of training. Instead Austin, has both the highest values and also a peak with N=11, while for N=17 the model could not make the predictions. Going back to Figure 1, we can conclude that cities with higher number of rentals (per hour or per day) tend to have lower MAPE and higher R2 score. However, even if Wien is not the city with most bookings, it has highest R2 score and MAPE. At the end, for cities like Austin, with really small rentals, it is clear that the ARIMA model has more difficulties in both fitting and making predictions.

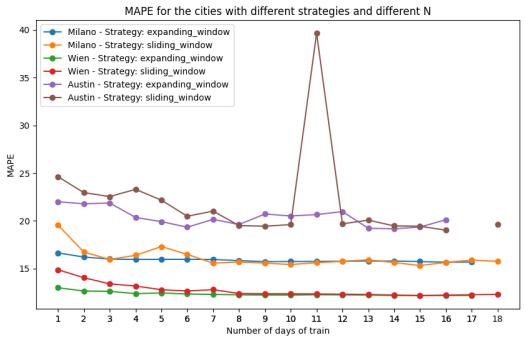
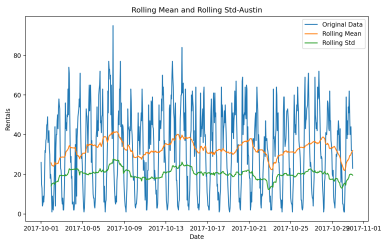


Figure 8: MAPE behavior for each couple city-configuration by changing the number of training days

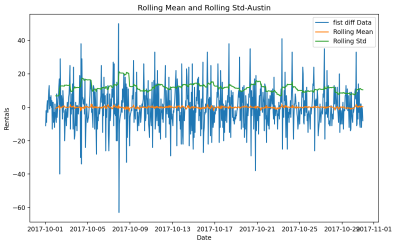
3 Appendix

- complete and well done
- missing comparison with baseline
- no time horizon study

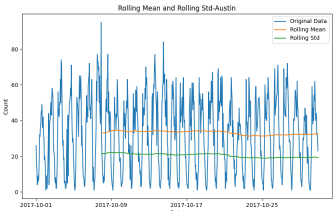
4.5/5



(a) rolling statistics on original dataset

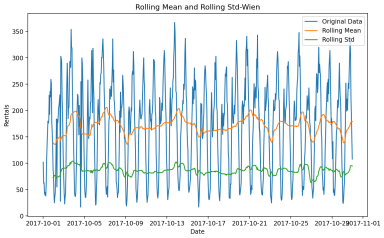


(b) rolling statistics on 1st ordered differenced data

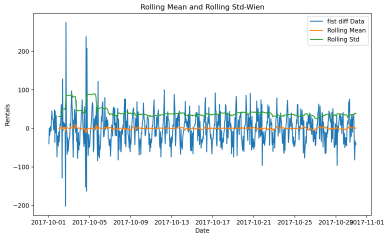


(c) rolling statistics on orginal dataset with weekly window

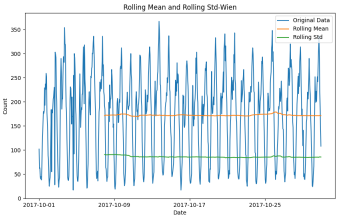
Figure 9: Differencing and rolling statistics on Austin's dataset



(a) rolling statistics on original dataset

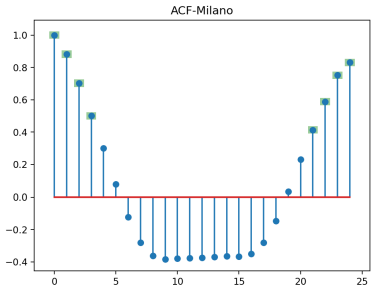


(b) rolling statistics on 1st ordered differenced data

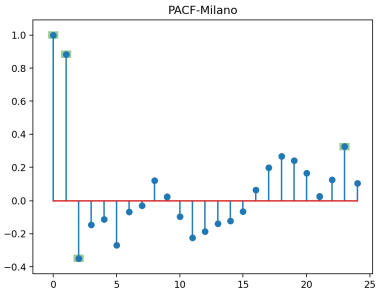


(c) rolling statistics on orginal dataset with weekly window

Figure 10: Differencing and rolling statistics on Wien's dataset

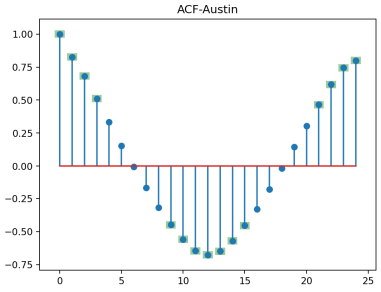


(a) ACF - Milano

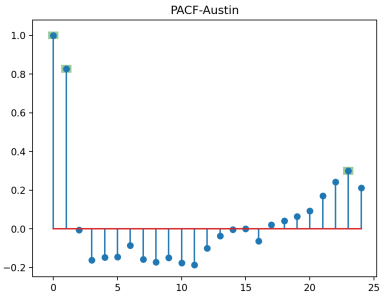


(b) PACF - Milano

Figure 11: ACF & PACF over hour of day for Milano's dataset

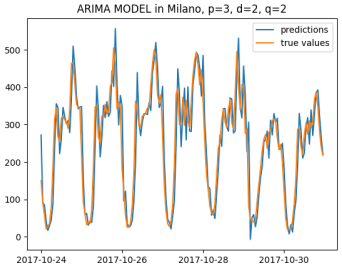


(a) ACF - Austin

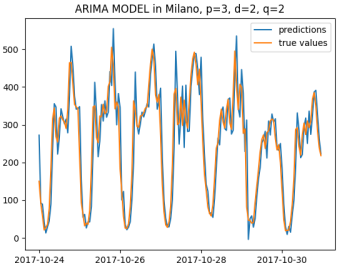


(b) PACF - Austin

Figure 12: ACF & PACF over hour of day for Austin's dataset

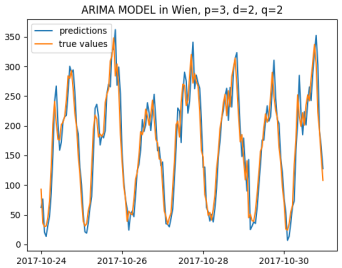


(a) expanding window Milano

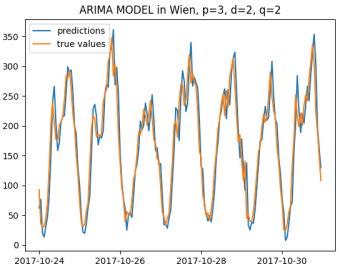


(b) sliding window Milano

Figure 13: ARIMA prediction - Milano

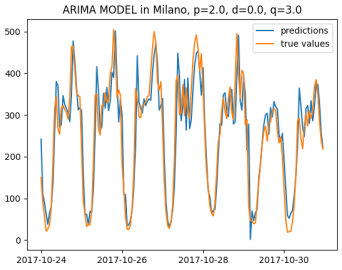


(a) expanding window Wien

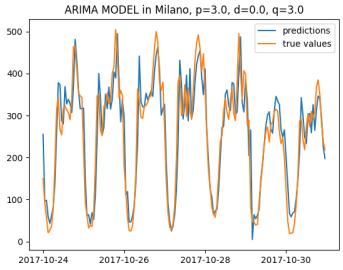


(b) sliding window Wien

Figure 14: point 6 - Wien

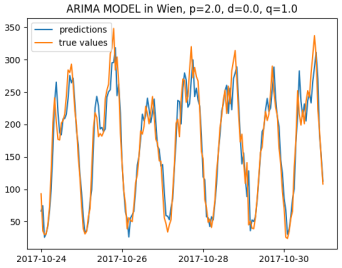


(a) expanding window

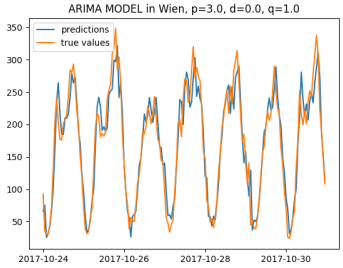


(b) sliding window

Figure 15: Prediction of Milano last 7 days of October for expanding and sliding window after optimization with grid search

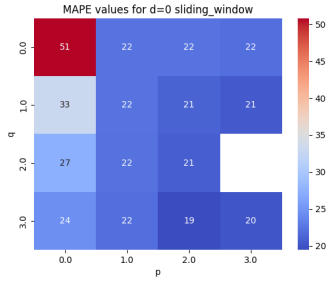


(a) expanding window

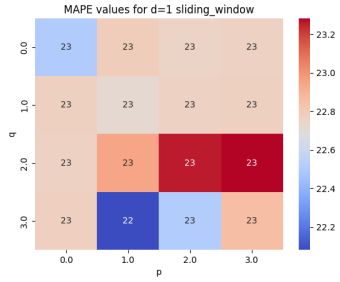


(b) sliding window

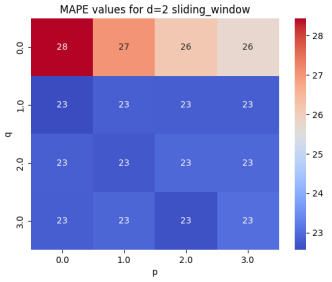
Figure 16: Prediction of Wien last 7 days of October for expanding and sliding window after optimization with grid search



(a) d = 0

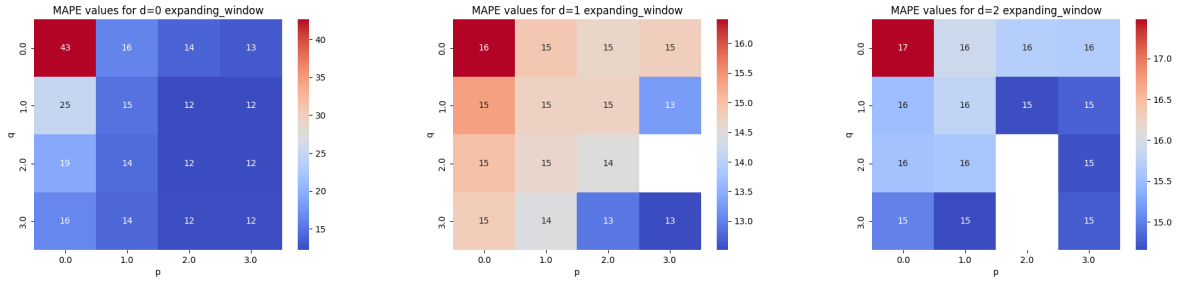


(b) d = 1



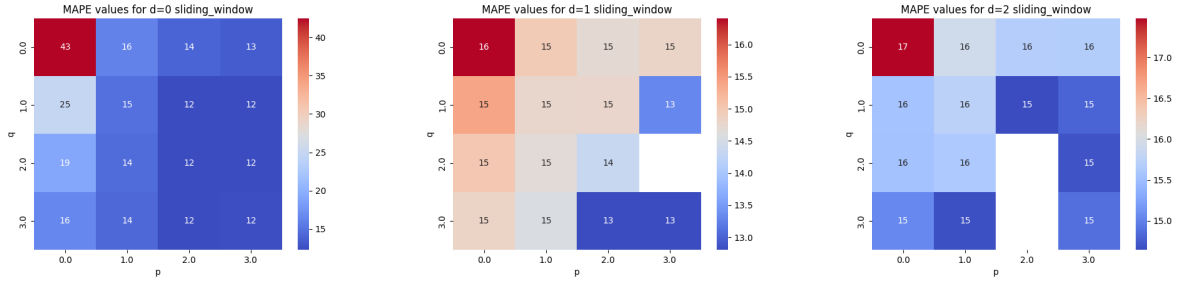
(c) d = 2

Figure 17: MAPE values for p,d,q with sliding in Austin



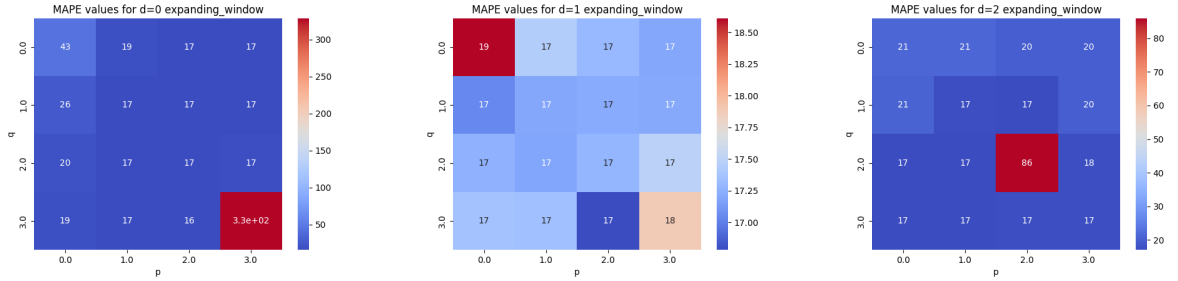
(a) d = 0 (b) d = 1 (c) d = 2

Figure 18: MAPE values for p,d,q with expanding window in Wien



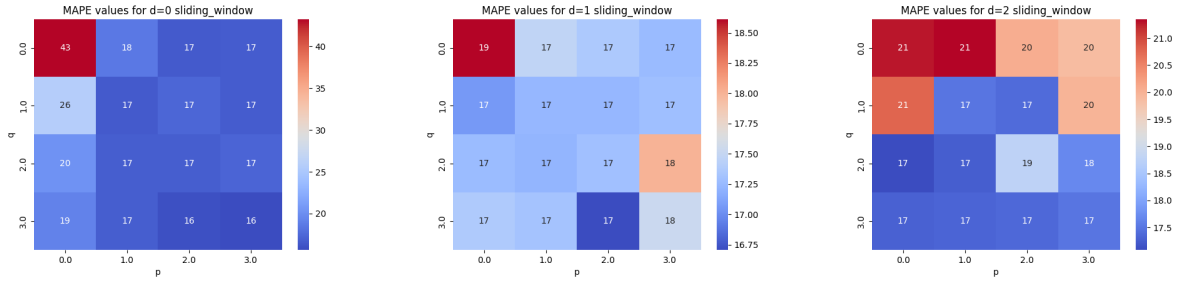
(a) d = 0 (b) d = 1 (c) d = 2

Figure 19: MAPE values for p,d,q with sliding window in Wien



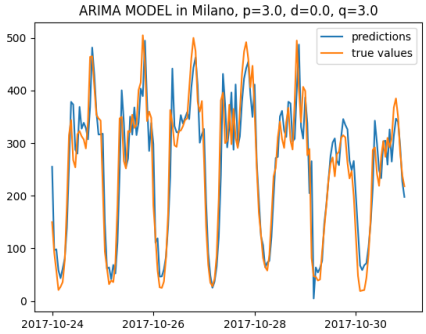
(a) d = 0 (b) d = 1 (c) d = 2

Figure 20: MAPE values for p,d,q with expanding window in Milano

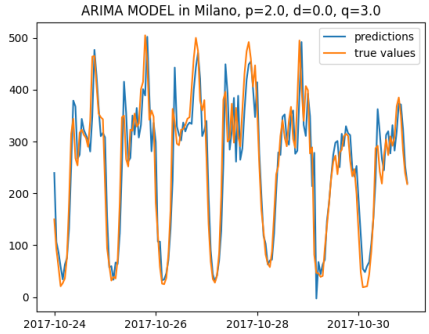


(a) d = 0 (b) d = 1 (c) d = 2

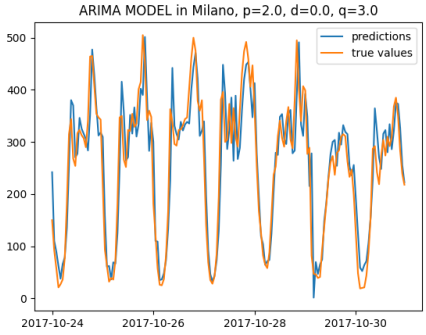
Figure 21: MAPE values for p,d,q with sliding window in Milano



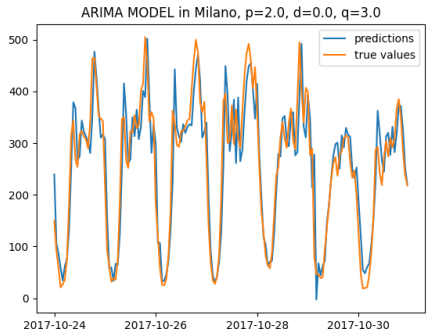
(a) Sliding window configuration with 14 days



(b) Sliding window configuration best number of days per training set, 15

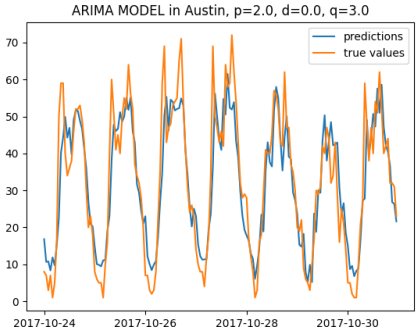


(c) expanding window configuration with 14 days

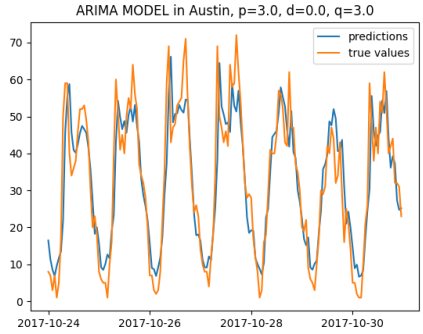


(d) Expanding window configuration best number of days per training set, 16

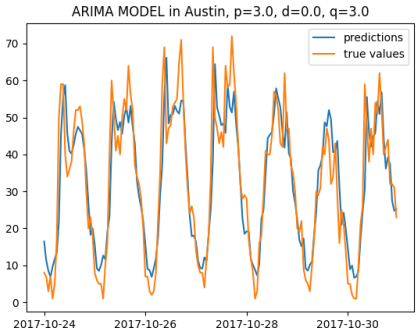
Figure 22: Comparison on Milano's last Week of October between before and after tuning the train



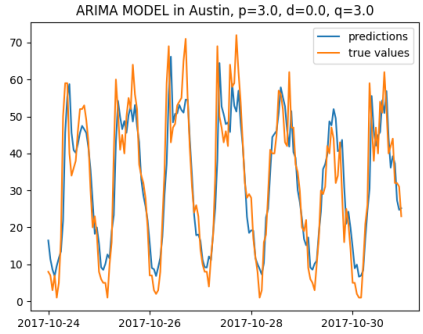
(a) Sliding window configuration with 14 days



(b) Sliding window configuration best number of days per training set, 16



(c) expanding window configuration with 14 days



(d) Expanding window configuration best number of days per training set, 14

Figure 23: Comparision on Austin's last Week of October between before and after tuning the train

Listing 1: Ex. 2.1, 2.2

```
start = datetime(2017, 10, 1, 0, 0, 0)
end = datetime(2017, 10, 31, 0, 0, 0)
cities = ['Milano', 'Wien', 'Austin']
for city in cities:
    data = Es.es_1_and_2(db = db, collection = 'PermanentBookings', city = city,
        start = start, end = end)

    def to_datetime(row):
        return datetime(row['year'], row['month'], row['day'], row['hour'])


    def query_date_city(city, init_time, final_time):
        return {"city": city, "init_time": {"$gte": init_time, "$lt": final_time}}
```

```
def mean_variance(db, collection, city, init_time, final_time):
    city_filter = Exercises.query_date_city(city, init_time, final_time)
    projection = {
               "duration": {"$subtract": ["$final_time", "$init_time"]}
    }
    second_filter = {"duration": {"$gte": 180}}
    pipeline_statistics = [
        {"$match": city_filter},
        {"$project": projection},
        {"$match": second_filter},
    ]

    result = pd.DataFrame(list(db[collection].aggregate(pipeline_statistics)))
    mean = int(result['duration'].mean())
    stand_dev = int(result['duration'].std())
    return mean, stand_dev


def hourly_mean(df, df_missing, column_date, column_values):
    copy_df = df_missing.copy()

    hour_df = np.array(df.groupby(df[column_date].dt.hour)[column_values].mean
        ())
    hourly_mean = np.floor(hour_df).astype(int)

    copy_df[column_values] = np.where(copy_df[column_values].isna(),
                                      hourly_mean[copy_df[column_date].dt.hour],
                                      copy_df[column_values])
    if len(copy_df.dropna(subset=[column_values])) == len(df):
        raise ArithmeticError
    return copy_df

def es_1_and_2(db, collection, city, start, end):
    init_time = int(start.replace(tzinfo = timezone.utc).timestamp())
    final_time = int(end.replace(tzinfo = timezone.utc).timestamp())
    mean, stand_dev = Exercises.mean_variance(db, collection, city, init_time,
         final_time)
    outliers_removal = mean + 2 * stand_dev

    projection = {"duration": {"$subtract": ["$final_time", "$init_time"]},
                  "init_date": 1,
                  "init_address": 1,
                  "final_address": 1}
```

```
filter_outliers = {"duration": {"$lte": int(outliers_removal)},
                   "$expr": {"$ne": ["$init_address", "$final_address"]}
                   }
project_hourly = {
    "_id": 0,
    "hour": {"$hour": "$init_date"},
    "day": {"$dayOfMonth": "$init_date"},
    "month": {"$month": "$init_date"},
    "year": {"$year": "$init_date"}, }
group_hourly = {
    "_id": {"hour": "$hour", "day": "$day", "month": "$month", "year": "
       $year"},
    "count": {"$sum": 1}
}

pipeline_hours = [
    {"$match": Exercises.query_date_city(city, init_time, final_time)},
    {"$project": projection}, {"$match": filter_outliers},
    {"$project": project_hourly},
    {"$group": group_hourly},
]

df_hourly = pd.DataFrame(list(db[collection].aggregate(pipeline_hours)))
df_hourly['_id'] = df_hourly['_id'].apply(Exercises.to_datetime)
date_array = pd.date_range(start, end - timedelta(hours = 1), freq = 'H')
date_df = pd.DataFrame({'_id': date_array})
result_df = pd.merge(date_df, df_hourly, on = '_id', how = 'left')
df_def = Exercises.hourly_mean(df = df_hourly, df_missing = result_df,
    column_date = '_id',
                               column_values = 'count')
df_def.set_index('_id', inplace = True)
df_def.sort_index(inplace = True)
plt.plot(df_def['count'].values)
l1, l2 = Exercises.dates_4_ticking(dates = list(df_def.index), n = 48*4)
plt.xticks(l1, l2)
plt.grid()
plt.savefig('data_'+city+'.png')
plt.close()
df_def.index = pd.to_datetime(df_def.index)
df_def.index.freq = 'H'
df_def.to_csv('data_'+city+'.csv')
return df_def
```

Listing 2: Ex. 2.3

```
def dates_4_ticking(dates: list, n: int):
    l1 = []
    l2 = []
    for i in range(0, len(dates), n):
        l1.append(i)
    for i in l1:
        formatted_string = dates[i].strftime("%Y-%m-%d")
        l2.append(formatted_string)
    return l1, l2

def rolling_comp(data, corr_col, city):
    rolling_mean = data[corr_col].rolling(window=24*7).mean() # one week
    rolling_std = data[corr_col].rolling(window=24*7).std()
    plt.figure(figsize=(10, 6))
    if corr_col == 'count':
        plt.plot(data[corr_col].values, label='Original Data')
    elif corr_col == 'first_diff':
        plt.plot(data[corr_col].values, label='fist diff Data')
    elif corr_col == 'second_diff':
        plt.plot(data[corr_col].values, label='second diff Data')
    plt.plot(rolling_mean.values, label='Rolling Mean')
    plt.plot(rolling_std.values, label='Rolling Std')
    plt.title('Rolling Mean and Rolling Std')
    l1, l2 = Exercises.dates_4_ticking(dates = list(data.index), n = 48*4)
    plt.xticks(l1, l2)
    plt.xlabel('Date')
    plt.ylabel('Count')
    plt.legend()
    plt.savefig('rolling_comp_'+city+'.png')
    plt.close()
    return


def es_3(data, column_values, city):
    copy_data = data.copy()

    Exercises.rolling_comp(copy_data, 'count', city = city)
    copy_data['first_diff'] = copy_data[column_values].diff()
    Exercises.rolling_comp(copy_data, 'first_diff', city = city)
    copy_data['second_diff'] = copy_data['first_diff'].diff()
    Exercises.rolling_comp(copy_data, 'second_diff', city = city)
    plt.plot(copy_data['second_diff'].values, label = 'second difference')
    plt.plot(copy_data['first_diff'].values, label = 'first difference')
    l1, l2 = Exercises.dates_4_ticking(dates = list(copy_data.index), n =
```

```
            48*4)
        plt.legend()
        plt.xticks(l1, l2)
        plt.grid()
        plt.savefig('stationary_data_'+city+'.png')
        plt.close()
        return copy_data
```

Listing 3: Ex. 2.4

```
acf, p_acf = Es.es_4(data = data, column_values = 'count', thresh_a = 0.4,
    thresh_p = 0.3, city = city)

    def filter_array(a, thresh):
        list_of_pos = []
        for i in range(len(a)):
            if abs(a[i]) > thresh:
                list_of_pos.append(i)

        return np.array(list_of_pos)


    def es_4(data, column_values, city, n_lags=24, thresh_a=0.4, thresh_p=0.3):

        p_acf = sm.tsa.pacf(data[column_values], nlags = n_lags)
        acf = sm.tsa.acf(data[column_values], nlags = n_lags)
        acf_thresh = Exercises.filter_array(acf, thresh = thresh_a)
        p_acf_thresh = Exercises.filter_array(p_acf, thresh = thresh_p)

        acf_sorted = acf_thresh[np.argsort(acf_thresh)]
        p_acf_sorted = p_acf_thresh[np.argsort(p_acf_thresh)]
        x_range_a = len(acf) - 1
        y_range_a = max(acf) - min(acf)
        ar_a = y_range_a / x_range_a
        rect_width = 0.8
        rec_height_a = ar_a * rect_width
        plt.stem(np.arange(len(acf)), acf)
        for val in acf_sorted:
            rect = Rectangle((val - rect_width/2, acf[val] - rec_height_a/2),
                             rect_width, rec_height_a, linewidth = 0, facecolor = '
                                green', alpha = 0.4)
            plt.gca().add_patch(rect)

        plt.savefig('autocorrelation_'+city+'.png')
        plt.close()
```

```
        x_range_p = len(p_acf) - 1
        y_range_p = max(p_acf) - min(p_acf)
        ar_p = y_range_p / x_range_p
        rec_height_p = ar_p * rect_width
        plt.stem(np.arange(len(p_acf)), p_acf)
        for val in p_acf_sorted:
            rect = Rectangle((val - rect_width/2, p_acf[val] - rec_height_p/2),
                rect_width, rec_height_p,
                            linewidth = 0, facecolor = 'green', alpha = 0.4)
            plt.gca().add_patch(rect)
        plt.savefig('p_autocorrelation_'+city+'.png')
        plt.close()
        return acf, p_acf
```

Listing 4: Ex. 2.5

```
samples_train = 24 * 14 #2 weeks
samples_test = 24 * 7 #1 week
train, test = Es.es_5(
            data,
            # init_date = start,
            column_values = 'count',
            samples_train = samples_train,
            samples_test = samples_test,
            city = city)

    def es_5(data, column_values, samples_train, samples_test, city: str):
        total_rows = len(data[column_values])
        index_test = total_rows-samples_test
        index_train = index_test - samples_train

        train = data.loc[data.index[index_train]:data.index[index_test-1]][
            column_values]
        test = data.loc[data.index[index_test]:][column_values]
        train.to_csv('train_' + city + '.csv', index = True)
        test.to_csv('test_' + city + '.csv', index = True)
        return train, test
```

Listing 5: Ex. 2.6

```
Es.es_6(data = data, p = 3, d = 2, q = 2, city = city, days_train = 14, days_test
    = 7, train = train, test = test)
```

```
def sliding_expanding(df = None, p_best = 24, d_best = 2, q_best = 24, city =
    'Milano', strategy = 'sliding_window',
                    days_train = 14, days_test = 7,
                    graph = True, train = None, test = None, out_name = None):
    # df = df.set_index('date')

    # first_day = datetime(2017, 10, 1)

    if train is None or test is None:
        if df is None:
            raise FileNotFoundError
        samples_train = days_train * 24 # 3 weeks
        samples_test = days_test * 7 # 1 week
        train, test = Exercises.es_5(df, 'count', samples_train, samples_test,
             city)

    tr_values = train.values.astype(float)
    # test = df.loc[train_end:test_end, ['count']]
    samples_test = len(test.values)
    history = [x for x in tr_values]
    predictions = []

    for t in range(samples_test):
        model = ARIMA(history, order=(p_best, d_best, q_best))
        model_fit = model.fit()
        forecast = model_fit.forecast()

        prediction = forecast[0]
        # only the prediction for the next hour
        predictions.append(prediction)
        obs = test.values[t]
        history.append(obs)
        if strategy == "sliding_window":
            history = history[1:]

    if graph:
        plt.plot(predictions, label='predictions')
        plt.title(f"ARIMA MODEL in {city}, p={p_best}, d={d_best}, q={q_best}"
            )
        plt.plot(np.array(test), label='true values')
        plt.legend()
        l1, l2 = Exercises.dates_4_ticking(dates = list(test.index), n = 48)
        plt.xticks(l1, l2)
        if out_name is not None:
            plt.savefig(out_name)
```

```
            plt.close()
        else:
            plt.show()
    m_ae = mae(predictions, test.values)
    mape = m_ae / np.mean(test.values) * 100
    r2 = r2_score(test.values, predictions)
    return m_ae, mape, r2


@staticmethod
def es_6(data, p, d, q, city, days_train = 14, days_test = 7, train = None,
    test = None):
    if test is not None and train is not None:
        mae_sw, mape_sw, r2_sw = Exercises.sliding_expanding(df = data,
                                                             p_best = p,
                                                             d_best = d,
                                                             q_best = q,
                                                             city = city,
                                                             strategy = '
                                                                sliding_window',
                                                             train = train, test =
                                                                test,
                                                             out_name = 'es_6_sw_'+
                                                                city+'.png'
                                                             )
        mae_ew, mape_ew, r2_ew = Exercises.sliding_expanding(df = data,
                                                             p_best = p,
                                                             d_best = d,
                                                             q_best = q,
                                                             city = city,
                                                             strategy = '
                                                                expanding_window',
                                                             train = train, test =
                                                                test,
                                                             out_name = 'es_6_ew_' +
                                                                 city + '.png'
                                                             )

    else:
        train, test = Exercises.es_5(data, column_values = 'count',
                                     samples_train = days_train*24, samples_test
                                        = days_test*24, city = city)
        mae_sw, mape_sw, r2_sw = Exercises.sliding_expanding(df = data,
                                                             p_best = p,
                                                             d_best = d,
```

```
                                                    q_best = q,
                                                    city = city,
                                                    strategy = '
                                                        sliding_window',
                                                    train = train,
                                                    test = test,
                                                    out_name = 'es_6_sw_'+
                                                        city+'.png')
            mae_ew, mape_ew, r2_ew = Exercises.sliding_expanding(df = data,
                                                    p_best = p,
                                                    d_best = d,
                                                    q_best = q,
                                                    city = city,
                                                    strategy = '
                                                        expanding_window',
                                                    train = train,
                                                    test = test,
                                                    out_name = 'es_6_ew_'+
                                                        city+'.png')

        print(f"sliding window: \nMAE->{mae_sw}\nMAPE->{mape_sw}\nR2->{r2_sw}")
        print(f"expanding window: \nMAE->{mae_ew}\nMAPE->{mape_ew}\nR2->{r2_ew}")
```

Listing 6: Ex. 7a

```
p, q, d = Es.es_7_a(df = df, column = 'MAPE', train = train, test = test, city =
    city)

    def es_7_a(df, column, train, test, city):
        copy_df = df.copy()
        copy_df[column].fillna(df[column], inplace=True)
        to_order = copy_df[column].values
        index_min = np.argsort(to_order)
        result = df.iloc[index_min[0]]
        p = result['p']
        q = result['q']
        d = result['d']
        print(city+' best combination: (d: ', d, 'q: ', q, 'p: ', p, ')')
        # dopo crea il modello e plotta (come nell' es_6)
        strategies = ['sliding_window', 'expanding_window']
        for strategy in strategies:
            df_s = df[df['strategy'] == strategy]
            index_min = np.argsort(df_s[column].values)
            result = df_s.iloc[index_min[0]]
            Exercises.sliding_expanding(df = df,
```

```
                                    train = train, test = test,
                                    city = city,
                                    p_best = result['p'], q_best = result['q'],
                                    d_best = result['d'], strategy = strategy,
                                    out_name = 'es_7a_'+strategy+'_'+city+'.png')
            d_unique = df_s['d'].unique()
            for val in d_unique:
                df_d = df_s[df_s['d'] == val]
                df_pivot = df_d.pivot(index = 'q', columns = 'p', values = column)
                ax = sns.heatmap(df_pivot, cmap = 'coolwarm', annot = True)
                ax.set_title("MAPE values for d=" + str(int(val)) + ' ' + strategy)
                if strategy == 'sliding_window':
                    plt.savefig('MAPE_sw_d'+str(int(val))+'_'+city+'.png')
                    plt.close()
                else:
                    plt.savefig('MAPE_ew_d' + str(int(val)) + '_' + city + '.png')
                    plt.close()
        return p, q, d
```

Listing 7: Ex. 7b

```
Es.es_7_b(data = data, column_value = 'count', p = p, q = q, d = d, days_train =
    14, days_test = 7, city = city)

    def best_n(data, column_value, days_test, p, d, q, city):
        tot_rows = len(data[column_value])
        tot_days = int(tot_rows / 24)
        max_train_days = tot_days - days_test
        end_train = tot_rows-(days_test*24)
        test = data.iloc[end_train:][column_value]
        df = pd.DataFrame({'days_train': [], 'MAE': [], 'MAPE': [], 'R2_score':
            [], 'strategy': []})
        new_row_1 = pd.DataFrame({'days_train': [], 'MAE': [], 'MAPE': [], '
            R2_score': [], 'strategy': []})
        new_row_2 = pd.DataFrame({'days_train': [], 'MAE': [], 'MAPE': [], '
            R2_score': [], 'strategy': []})

        for i in range(1, max_train_days + 1):
            print(i)
            start_train = end_train - i * 24
            train = data.iloc[start_train:end_train][column_value]
            try:
                mae_sw, mape_sw, r2_sw = Exercises.sliding_expanding(df = data,
                    p_best = p, d_best = d, q_best = q,
                                                                    city = city,
```

```
                                                                strategy = '
                                                                sliding_window',
                                                            train = train, test
                                                                 = test, graph =
                                                                 False)
            mae_ew, mape_ew, r2_ew = Exercises.sliding_expanding(df = data,
                p_best = p, d_best = d, q_best = q,
                                                            city = city,
                                                                strategy = '
                                                                expanding_window
                                                                ',
                                                            train = train, test
                                                                 = test, graph =
                                                                 False)
            new_row_1 = pd.DataFrame({'days_train': [i], 'MAE': [mae_sw], 'MAPE
                ': [mape_sw], 'R2_score': [r2_sw],
                                      'strategy': ["sliding_window"]})
            new_row_2 = pd.DataFrame({'days_train': [i], 'MAE': [mae_ew], 'MAPE
                ': [mape_ew], 'R2_score': [r2_ew],
                                      'strategy': ["expanding_window"]})
        except np.linalg.LinAlgError:

            new_row_1 = pd.DataFrame({'days_train': [i],
                                      'MAE': [None], 'MAPE': [None], 'R2_score':
                                          [None],
                                      'strategy': ["sliding_window"]})
            new_row_2 = pd.DataFrame({'days_train': [i],
                                      'MAE': [None], 'MAPE': [None], 'R2_score':
                                          [None],
                                      'strategy': ["expanding_window"]})
        finally:
            new_row = pd.concat([new_row_1, new_row_2], ignore_index = True)
            df = pd.concat([df, new_row], ignore_index = True)
    df.to_csv(f"days_search_{city}.csv", index = False)

def es_7_b(data, column_value, p, q, d, days_train, days_test, city):

    Exercises.sliding_expanding(df = data, p_best = p, q_best = q, d_best = d,
        city = city,
                                strategy = "expanding_windows",
                                days_train = days_train, days_test = days_test)
    Exercises.sliding_expanding(df = data, p_best = p, q_best = q, d_best = d,
        city = city,
                                strategy = "sliding_window",
                                days_train = days_train, days_test = days_test)
```

```
response = input('Hai gi fatto runnare best_n? y/n')
        if response.startswith('y') is not True:
            Exercises.best_n(data = data, column_value = column_value, days_test =
                days_test,
                           p = p, d = d, q = q, city = city)
        df = pd.read_csv(f"days_search_{city}.csv")
        strategies = ['sliding_window', 'expanding_window']

        for strategy in strategies:
            df_s = df[df['strategy'] == strategy]
            df_bd = df_s.sort_values(by='MAPE')
            days_train_best = df_bd.iloc[0]['days_train']
            m_ae, mape, r2 = Exercises.sliding_expanding(df = data, p_best = p,
                q_best = q, d_best = d,
                                                         city = city, strategy =
                                                            strategy,
                                                         days_train = days_train_best,
                                                            days_test = days_test)
            print(f"for {strategy}:\nMAE: {m_ae}\nMAPE: {mape}\nR2: {r2}\nBest num
                day-s training: {days_train_best}")
            plt.plot(df_s['days_train'], df_s['MAPE'])
            plt.title('MAPE behaviour changing training days in ' + strategy)
            plt.grid()
            plt.savefig('es_7b_MAPE_'+city+'.png')
            plt.close()
```

Listing 8: Ex. 7c

```
    def es_7_c():
        df_Milano = pd.read_csv('days_search_Milano.csv')
        df_Wien = pd.read_csv('days_search_Wien.csv')
        df_Austin = pd.read_csv('days_search_Austin.csv')
        df_Austin = df_Austin.loc[:df_Austin.index[34]]
        df_Milano = df_Milano.loc[:df_Milano.index[34]]
        df_Wien = df_Wien.loc[:df_Wien.index[34]]

        plt.figure(figsize=(10,6))
        plt.plot(df_Milano[df_Milano['strategy'] == 'expanding_window']['
            days_train'],
                df_Milano[df_Milano['strategy'] == 'expanding_window']['MAPE'],
                marker='o',linestyle = ' ', label='Milano - Strategy:
                    expanding_window')


        plt.plot(df_Milano[df_Milano['strategy'] == 'sliding_window']['days_train'
```

```
    ],
         df_Milano[df_Milano['strategy'] == 'sliding_window']['MAPE'],
         marker='o', linestyle = ' ',label='Milano - Strategy:
            sliding_window')

plt.plot(df_Wien[df_Wien['strategy'] == 'expanding_window']['days_train'],
         df_Wien[df_Wien['strategy'] == 'expanding_window']['MAPE'],
         marker='o',linestyle = ' ', label='Wien - Strategy:
            expanding_window')
plt.plot(df_Wien[df_Wien['strategy'] == 'sliding_window']['days_train'],
         df_Wien[df_Wien['strategy'] == 'sliding_window']['MAPE'],
         marker='o', linestyle = ' ', label='Wien - Strategy:
            sliding_window')

plt.plot(df_Austin[df_Austin['strategy'] == 'expanding_window']['
    days_train'],
         df_Austin[df_Austin['strategy'] == 'expanding_window']['MAPE'],
         marker='o',linestyle = ' ', label='Austin - Strategy:
            expanding_window')
plt.plot(df_Austin[df_Austin['strategy'] == 'sliding_window']['days_train'
    ],
         df_Austin[df_Austin['strategy'] == 'sliding_window']['MAPE'],
         marker='o', linestyle = ' ',label='Austin - Strategy:
            sliding_window')
plt.xticks(df_Milano['days_train'])
plt.xlabel('Number of days of train')
plt.ylabel('MAPE')
plt.title('MAPE for the cities with different strategies and different N')
plt.legend()
plt.show()
plt.savefig('es_7c.png')
```