Politecnico di Torino

ICT for Smart Mobility

Second Laboratory Report

Professor:

Marco Mellia

Group 9

Students:

Andres Giovanni Orellana Hidalgo s306610
Laura Maria Joaqui Muñoz s307972
Keyvan Abbasmajidi s300443

A.Y. 2023/2024

Contents

1 Step 3 - Prediction Using ARIMA Models:

The purpose of this laboratory was experimenting with predictions using ARIMA models, as well as analyse how error changes with hyper-parameters changes in ARIMA models with respect to a time series of data, it was considered rental data of each hour for one month in Washington DC(Car2Go), Torino(Car2Go) and Torino Enjoy. To properly use the ARIMA model it is important not to have missing values in the series. In order to find the best month to use with ARIMA model, we considered from July 2017 to January 2018.

1.1 Hourly Data Extraction:

As previously explained, we considered from July 2017 to January 2018 in order to see which one of these months is the best one to do a further analysis with the ARIMA model, for this we evaluated each month to see the number of missing values, if it satisfies stationary condition and if it has a periodic trend.

1.2 Handling Missing Samples

Evaluating each month we were able to see that October is the best month among the others, because it was the month with the less number of missing values, for the special months, December and January are not convenient due to stationary and periodicity analysis explained in 1.3, for the other months we were able to see that they had more missing values than October so the selected month for further analysis is October, although there were missing values to handle, it was the best month among the other. For handling missing data we adopted the strategy to replace the missing values with a week shifted values of the next week plus noise, if the missing data appears at the last week of the month the missing are replaced with a week shifted value at the previous week. This strategy takes into consideration on the weekly periodic behaviour in the data, results and missing values are shown in Table 1.

City	Missing Values
Washington DC	13
Torino	9
Torino Enjoy	9

Table 1: Missing Values for Each City

1.3 Stationary And Periodicity Analysis

When using ARIMA model it is necessary to satisfy the stationary conditions. For December and January, there is not periodic trend and the series do not satisfy the stationary condition, they are non-stationary due to the holy-days so these are not good months to benefit from using the ARIMA model. Using a window size of 168 hours the rolling statistics are shown in Figure 1 and we were able to see that the properties are constant over time. Also applying Augmented Dickey Fuller(ADF) test that is commonly used statistical test to test weather a time series is stationary or not. Results are shown in Table 2. Finally we obtained that October is a stationary month so differentiating is not necessary in this case, additionally we can set the parameter d equal to zero in ARIMA(p,d,q), so our model simplifies to a ARMA model.

City	$ADFtest < 0.005$
Washington DC	$1.93x10^{-}7$
Torino	0.00021
Torino Enjoy	0.00021

Table 2: Augmented Dickey Fuller Test

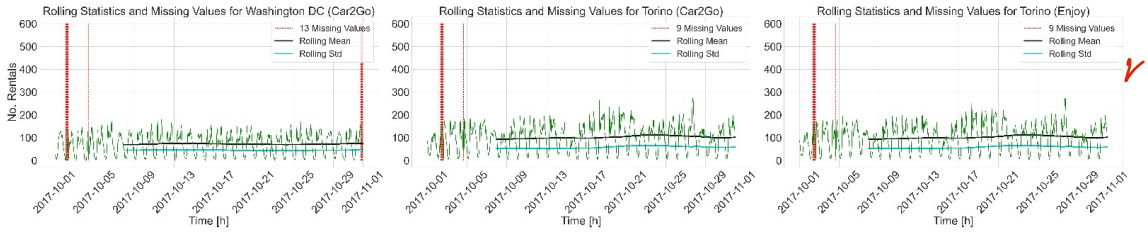

Figure 1: Statistics with the window size of 168 hours(one week)

1.4 Auto Correlation Function(ACF) And Partial Auto Correlation Function(PACF)

In order to use the ARMA model we need to know two sub-models Auto-Regressive(AR) model and Moving Average(MA) that require hyper-parameters that correspond to number of previous samples (p) and previous errors (q) that affect the current realization of the observed random variable, AR demonstrates that for $h > p$ the PACF is equal or nearly equal to 0 and MA demonstrates that the ACF is equal or nearly equal to 0 for $h > q$.

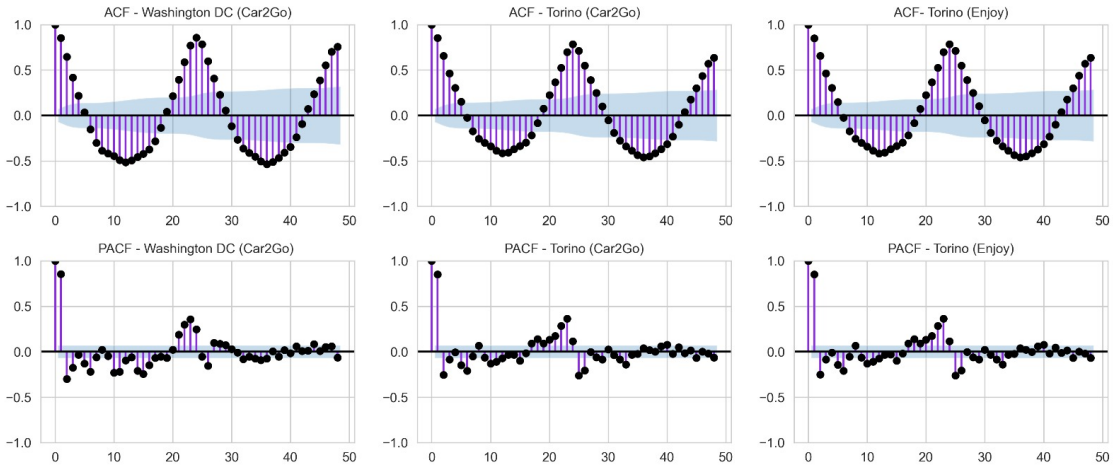

Figure 2: ACF and PACF for 48 hours

based on Figure 2, were established p=2 and q=4 for Washington DC(Car2Go), p=2 and q=4 for Torino(Car2Go) and p=2 and q=4 for Torino Enjoy as boundary parameters.

1.5 ARMA Model

For training of the ARMA model we used the parameters p and q selected in part 1.4. 3 weeks were used for training and 1 week for testing, for each sample a new model was trained with the expanding window learning strategy.

1.6 ARMA model Training and Testing

Our model was trained and tested with the samples chosen in section 1.5. To evaluate the model were used the Minimum Mean Squared Error (MSE), Mean Absolute error (MAE), Mean Absolute Percentage Error (MAPE) and R^2. In Figure 3 we can see the predictions, real rentals and element wise error. In Table 3 we can see the measurements for evaluating our model, finally model performance for Torino Enjoy for different values of p can be seen in Figure 4, as we increase the p value, the model becomes more complex and tends to fluctuate more, resulting in more errors.

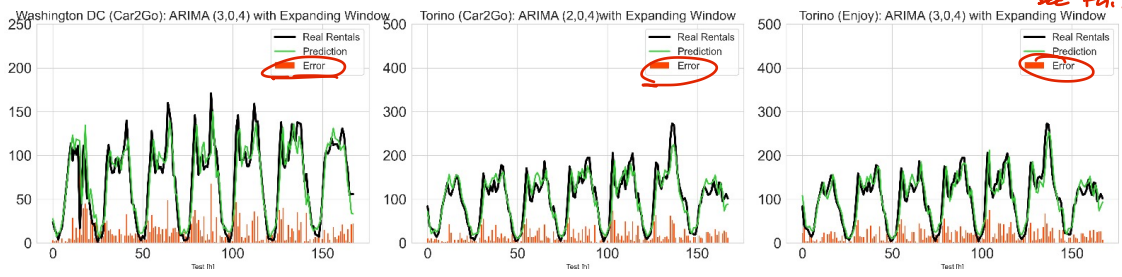

Figure 3: First ARMA predictions for each city

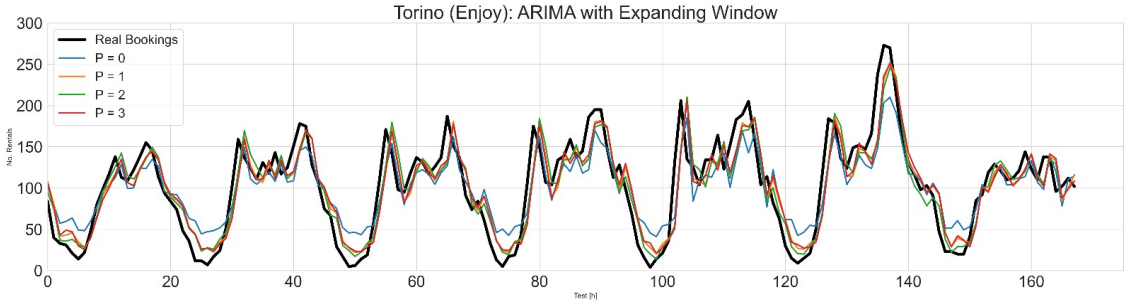

Figure 4: Predictions for Torino Enjoy for q=2 and different values of p

City	p	d	q	MSE	MAE	MAPE	R^2
Washington DC	2	0	4	383.37	13,94	0.58	0.81
Torino	2	0	4	574.01	17.82	0.27	0.82
Torino Enjoy	2	0	4	622.47	19.53	0.31	0.81

Table 3: Evaluation errors for the three cities

From Table 3 we can see that Washington DC has the minimum MSE AND MAE this was expected because MSE and MAE are proportional to the number of rentals and we already knew Washington DC had the less number of rentals. From MAPE(independent on the number of rentals) we can more accurately decide about the model performance.

1.7 Parameters Impact

1.7.1 Finding Best Parameters(p and q)

For obtaining better parameters we performed a grid search with fixed number of training and testing samples, same as the samples selected in section 1.5. The learning strategy was the expanding window and MAPE was used for evaluate its performance, for a variation of p=[0,3] and q=[0,4] for Washington DC(Car2Go) and Torino Enjoy and p=[0,2] and q=[0,4] for Torino(Car2Go). These ranges were chosen as considered in section 1.4, in Figure 4 are shown the MAPE errors for the three cities.

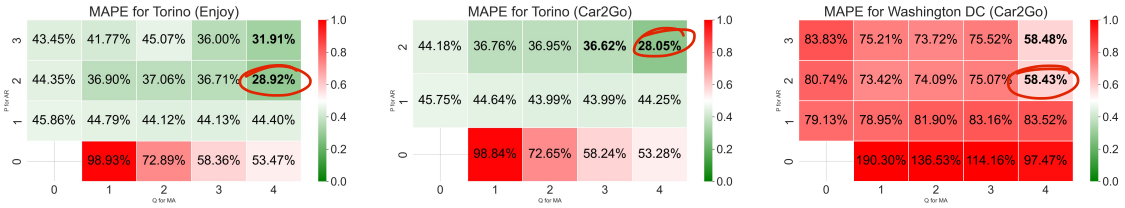

Figure 5: Grid search MAPE for each city

From Figure 4 we obtained that for all the three cities the best parameters are p=2 and q=4.

1.7.2 Change Learning Strategy and Training Sample Size

in order to evaluate the learning strategy and the training sample, another grid search was performed to find the optimal training sample(N) and the best learning strategy between Expanding Window and Sliding Window. The ARMA was trained with ranging size from one to three weeks with steps of two days with both strategies.

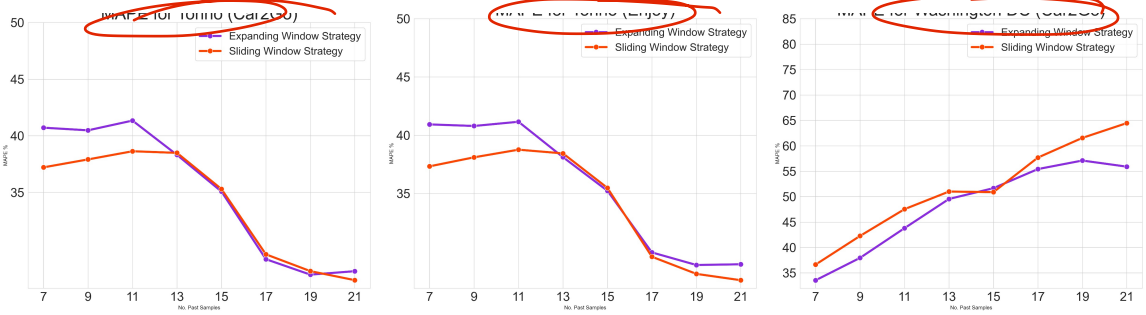

Figure 6: Expanding vs Sliding window learning strategies MAPE

In Figure 6 we see that expanding window is a better learning strategy since there is a little but remarkable difference between the two strategies MAPE, expanding window enables the prediction model not only to be trained with the initial training set but also the test values already predicted so the training window expands as predictions are made, also we can see that MAPE decreases as the number of training samples increase as the model generalizes the data predicting new data with less errors.

1.7.3 Optimal Hyper-parameters

With all the parameters already optimized we can compare them with results obtained in previous sections. We can conclude that Expanding Window outperforms Sliding Window at least in most cases, also keeping a smaller q and p is better since model is more robust, finally Figure 7 shows the results with the optimal parameters found in previous sections.

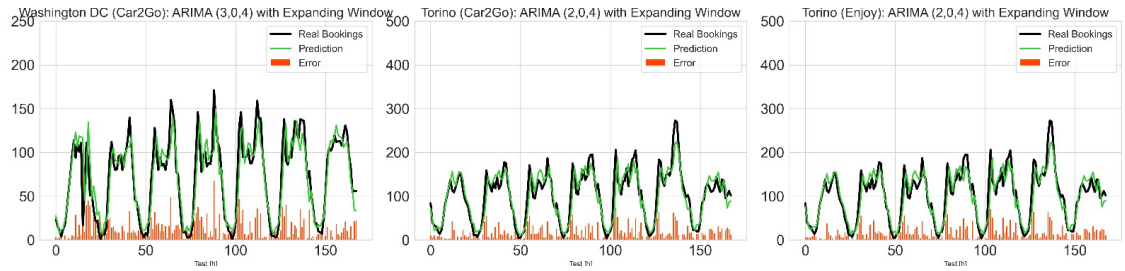

Figure 7: Prediction Results with obtained optimal parameters

2 Appendix

2.1 Hourly Data Extraction

```
# Connecting to the database through PyMongo
client = pm.MongoClient('bigdatadb.polito.it',
                        ssl=True,authSource = 'carsharing',
                        tlsAllowInvalidCertificates=True,
                        username='ictts',
                        password='Ict4SM22!')
db = client['carsharing']
# Get Permanent Bookings Collections
Car2goPermanentBook = db['PermanentBookings']
Car2goPermanentPark = db['PermanentParkings']
Car2goActiveBook = db['ActiveBookings']
Car2goActivePark = db['ActiveParkings']
enjoyPermanentBook = db['enjoy_PermanentBookings']
enjoyPermanentPark = db['enjoy_PermanentParkings']
enjoyActiveBook = db['enjoy_ActiveBookings']
enjoyActivePark = db['enjoy_ActiveBookings']
```

```
def get_cars_per_Hour_bookFilter(city, Unix_startTime, Unix_endTime):
    query = [{'$match': { 'city': city,'init_time': {'$gte': Unix_startTime, '$lte': Unix_endTime}}},{'$project' : { '_id':0,'city': 1,'hourOfDay': {'$dateFromParts': {'year':{'$year':'$init_dat

    return query
```

```
cities=['Washington DC', 'Torino Enjoy', 'Torino']
```

```
results = {}
for city in cities:
    city_dict = {'bookings':{}}
    startingDate = dt(2017, 10, 1, 0) # Init Time
    endingDate = dt(2017, 11, 1, 0) # End Time
    Unix_startTime = dt.timestamp(startingDate)
    Unix_endTime = dt.timestamp(endingDate)
# Make Ready the Query for the collection
    if 'Enjoy' in city:
        book_pipe = get_cars_per_Hour_bookFilter('Torino', Unix_startTime, Unix_endTime)
        i_booking = Car2goPermanentBook
    else:
        book_pipe = get_cars_per_Hour_bookFilter(city, Unix_startTime, Unix_endTime)
        i_booking = Car2goPermanentBook
    # Assign proper collection for the aggregation
    daily_bookings = i_booking.aggregate(book_pipe)
    for item in daily_bookings:
        city_dict['bookings'][item['_id']] = item['count']
# Dictionary to DataFrame Convertion
        city_df_from_dict = pd.DataFrame.from_dict(city_dict)
        city_df = city_df_from_dict.reindex(pd.date_range(start="2017-10-01",end="2017-11-01",freq='H'),fill_value=0)
    results[city] = city_df[:-1]
```

2.2 Handling Missing Samples

```
# Handling Missing Data
# If there were any missings: replace with last or next week of data
all_dates = []
missing_dates = {}
week_list = pd.date_range(start="2017-10-01",end="2017-11-01",freq='H').strftime("%Y-%m-%d %H:%M:%S").tolist()
for day in week_list: all_dates.append(dt.strptime(day, '%Y-%m-%d %H:%M:%S'))
for city in cities:
    missing_dates[city] = []
    for date in all_dates[:-1]:
        if results[city].loc[date]['bookings']==0:
            missing_dates[city].append(date)
            if date.day>=25 or date.month==11:
                results[city].loc[date]['bookings'] = results[city].loc[date.replace(day=date.day-7)]['bookings'] + random.randint(0,10)
            else:
                results[city].loc[date]['bookings'] = results[city].loc[date.replace(day=date.day+7)]['bookings'] + random.randint(0,10)
        if city=='Torino Enjoy':
            if results[city].loc[date]['bookings']>=500:
                results[city].loc[date]['bookings'] = (results[city].loc[date+timedelta(days=1)]['bookings']+results[city].loc[date-timedelta(days=-1)]['bookings'])/2
```

2.3 Stationary And Periodicity Analysis

```
# Adjusting plots rcParams for the proper fontsizes
plt.rc('axes', labelsize=30)
plt.rc('ytick', labelsize=34)
plt.rc('xtick', labelsize=34)
plt.rc('legend', fontsize=28)
plt.rc('figure', titlesize=28)
fig, axes = plt.subplots(1, 3, figsize=(50, 10), tight_layout=True)
date_ticks = np.arange("2017-10-01","2017-11-01", dtype ='datetime64')
fig.subplots_adjust(wspace=0.08)
#Get X and Y Axis for Each City
x_washington, y_washington = zip(*sorted(results['Washington DC']['bookings'].items()))
x_torino, y_torino = zip(*sorted(results['Torino']['bookings'].items()))
x_torino_enj, y_torino_enj = zip(*sorted(results['Torino Enjoy']['bookings'].items()))
# Rolling Statistics for 168h (7d) for Seattle Car2Go
r_mean_washington, r_std_washington = results['Washington DC']['bookings'].rolling(168).mean(), results['Washington DC'].rolling(168).std()
axes[0].tick_params(axis='x', labelrotation = 45)
axes[0].set_title("Rolling Statistics and Missing Values for Washington DC (Car2Go)", fontsize=36)
axes[0].set_xlabel("Time [h]", fontsize=36)
axes[0].set_ylabel("No. Rentals", fontsize=36)
axes[0].vlines(missing_dates['Washington DC'], color="red", linestyle="--", alpha=1, ymin=0, ymax=600, label=f"{len(missing_dates['Washington DC'])} Missing Values")
axes[0].plot(r_mean_washington, label='Rolling Mean', linewidth=4, color='k')
axes[0].plot(r_std_washington, label='Rolling Std', linewidth=4, color='c')
sns.lineplot(data=results['Washington DC'], x=x_washington, y=y_washington, ax=axes[0], color='g')
# Rolling Statistics for 168h (7d) for Milano Car2Go
r_mean_torino, r_std_torino = results['Torino']['bookings'].rolling(168).mean(), results['Torino']['bookings'].rolling(168).std()
axes[1].tick_params(axis='x', labelrotation = 45)
axes[1].set_title("Rolling Statistics and Missing Values for Torino (Car2Go)", fontsize=36)
axes[1].set_xlabel("Time [h]", fontsize=36)
axes[1].vlines(missing_dates['Torino'], color="red", linestyle="--", alpha=1, ymin=0, ymax=600, label=f"{len(missing_dates['Torino'])} Missing Values")
axes[1].plot(r_mean_torino, label='Rolling Mean', linewidth=4, color='k')
axes[1].plot(r_std_torino, label='Rolling Std', linewidth=4, color='c')
sns.lineplot(data=results['Torino'], x=x_torino, y=y_torino, ax=axes[1], color='g')
# Rolling Statistics for 168h (7d) for Milano Enjoy
r_mean_torino_enj, r_std_torino_enj = results['Torino Enjoy']['bookings'].rolling(168).mean(), results['Torino Enjoy']['bookings'].rolling(168).std()
axes[2].tick_params(axis='x', labelrotation = 45)
axes[2].set_title("Rolling Statistics and Missing Values for Torino (Enjoy)", fontsize=36)
axes[2].set_xlabel("Time [h]", fontsize=36)
axes[2].vlines(missing_dates['Torino Enjoy'], color="red", linestyle="--", alpha=1, ymin=0, ymax=600, label=f"{len(missing_dates['Torino Enjoy'])} Missing Values")
axes[2].plot(r_mean_torino_enj, label='Rolling Mean', linewidth=4, color='k')
axes[2].plot(r_std_torino_enj, label='Rolling Std', linewidth=4, color='c')
sns.lineplot(data=results['Torino Enjoy'], x=x_torino_enj, y=y_torino_enj, ax=axes[2], color='g')
plt.savefig('S3_T3.png', dpi=300)
plt.show()
```

2.4 Auto Correlation Function(ACF) And Partial Auto Correlation Function(PACF)

```
# Adjusting plots rcParams for the proper fontsizes
plt.rc('axes', labelsize=2)
plt.rc('ytick', labelsize=10)
plt.rc('xtick', labelsize=10)
plt.rc('legend', fontsize=14)
plt.rc('figure', titlesize=22)
plt.rcParams.update({'axes.titlesize': 'medium'})
fig, axes = plt.subplots(2, 3, figsize=(12, 5), tight_layout=True)
fig.tight_layout()
# Statsmodel ACF and PACF plots
sm.graphics.tsa.plot_acf(results['Washington DC']['bookings'].values, lags=48, title='ACF - Washington DC (Car2Go)', ax=axes[0,0], color='k', vlines_kwargs={"colors": 'blueviolet'})
sm.graphics.tsa.plot_pacf(results['Washington DC']['bookings'].values, lags=48, title='PACF - Washington DC (Car2Go) ', ax=axes[1,0], color='k', vlines_kwargs={"colors": 'blueviolet'})
sm.graphics.tsa.plot_acf(results['Torino']['bookings'].values, lags=48, title='ACF - Torino (Car2Go)', ax=axes[0,1], color='k', vlines_kwargs={"colors": 'blueviolet'})
sm.graphics.tsa.plot_pacf(results['Torino']['bookings'].values, lags=48, title='PACF - Torino (Car2Go)', ax=axes[1,1], color='k', vlines_kwargs={"colors": 'blueviolet'})
sm.graphics.tsa.plot_acf(results['Torino Enjoy']['bookings'].values, lags=48, title='ACF- Torino (Enjoy)', ax=axes[0,2], color='k', vlines_kwargs={"colors": 'blueviolet'})
sm.graphics.tsa.plot_pacf(results['Torino Enjoy']['bookings'].values, lags=48, title='PACF - Torino (Enjoy)', ax=axes[1,2], color='k', vlines_kwargs={"colors": 'blueviolet'})
fig.subplots_adjust(wspace=0.2, hspace=0.3)
plt.savefig('S3_T4.png', dpi=300)
plt.show()
# Adfuller Stationarity check
print(f"Augmented Dickey Fuller (ADF) Test for Washington (Car2Go): {adfuller(results['Washington DC'].values)[1]} - < 0.005. Good! It's Stationary!")
print(f"Augmented Dickey Fuller (ADF) Test for Torino (Car2Go): {adfuller(results['Torino'].values)[1]} - < 0.005. Good! It's Stationary!")
print(f"Augmented Dickey Fuller (ADF) Test for Torino (Enjoy): {adfuller(results['Torino Enjoy'].values)[1]} - < 0.005. Good! It's Stationary!")
```

2.5 ARMA Model

```
def ARIMA_fit(data, city, p, q, train_size, test_size, mode="expanding"):
# Train, test split
    counter = 0
    train_list = []
    test_list = []
    for index in data.index.values:
        if counter <= train_size:
            train_list.append(data.loc[index]['bookings'])
            counter += 1
        elif counter > train_size and counter <= test_size+train_size:
            test_list.append(data.loc[index]['bookings'])
            counter += 1
        else:
            break
# Stepwise fit and prediction for test data
    predictions = []
    for row in range(test_size):
        model = ARIMA(train_list, order=(p, 0, q))
        model = model.fit(method='statespace')
        pred = model.forecast()
        predictions.append(pred[0])
        observation = test_list[row]
        if mode=="expanding":
            train_list.append(observation)
        elif mode=="sliding":
            train_list.append(observation)
            train_list.pop(0) # Drop the first value for sliding
    return test_list, predictions
```

```
# ARMA max hyperparameters and train test split ratio
days_to_train = 21
days_to_test = 7
p_values = {'Washington DC':3, 'Torino':2, 'Torino Enjoy':3}
q_values = {'Washington DC':4, 'Torino':4, 'Torino Enjoy':4}
```

```
# Loop over cities for the maximum possible value for p and q in expanding mode
ARIMA_results = {}
tic = time.perf_counter()
for city in cities:
    print(city)
    ARIMA_results[city] = ARIMA_fit(results[city], city, p=p_values[city], q=q_values[city],train_size=days_to_train*24, test_size=days_to_test*24,mode="expanding")
toc = time.perf_counter()
print(f"Job Finished in {toc-tic} seconds.")
```

2.6 ARMA model Training and Testing

```
def get_errors(original, predicitions):
    MSE = mean_squared_error(original, predicitions)
    MAE = mean_absolute_error(original, predicitions)
    MAPE = mean_absolute_percentage_error(original, predicitions)
    R2 = r2_score(original, predicitions)
    return [MSE, MAE, MAPE, R2]
```

```
plt.rc('axes', labelsize=20)
plt.rc('ytick', labelsize=20)
plt.rc('xtick', labelsize=20)
plt.rc('legend', fontsize=14)
plt.rc('figure', titlesize=18)
plt.rcParams.update({'axes.titlesize': 'small'})
fig, axes = plt.subplots(1, 3, figsize=(22, 5), tight_layout=True)
fig.tight_layout()
# Get element-wise error for better inspection
# Plot real, predicted, and error
error = np.abs(np.subtract(ARIMA_results['Washington DC'][0], ARIMA_results['Washington DC'][1]))
axes[0].plot(ARIMA_results['Washington DC'][0], color='black',label='Real Rentals', linewidth=3)
axes[0].plot(ARIMA_results['Washington DC'][1], color='limegreen', label='Prediction', linewidth=2)
axes[0].bar(np.arange(0,len(ARIMA_results['Washington DC'][0]),1), error, color='orangered', label='Error', width=1)
axes[0].set_title(f"Washington DC (Car2Go): ARIMA ({p_values['Washington DC']},0,{q_values['Washington DC']}) with Expanding Window", fontsize=18)
axes[0].legend(loc='best')
axes[0].set_xlabel('Test [h]')
axes[0].set_ylabel('No. Rentals')
axes[0].set_ylim(0, 250)
error = np.abs(np.subtract(ARIMA_results['Torino'][0], ARIMA_results['Torino'][1]))
axes[1].plot(ARIMA_results['Torino'][0], color='black', label='Real Rentals', linewidth=3)
axes[1].plot(ARIMA_results['Torino'][1], color='limegreen', label='Prediction', linewidth=2)
axes[1].bar(np.arange(0,len(ARIMA_results['Torino'][0]),1), error, color='orangered', label='Error', width=1)
axes[1].set_title(f"Torino (Car2Go): ARIMA ({p_values['Torino']},0,{q_values['Torino']})with Expanding Window", fontsize=18)
axes[1].legend(loc='best')
axes[1].set_xlabel('Test [h]')
axes[1].set_ylim(0, 500)
error = np.abs(np.subtract(ARIMA_results['Torino Enjoy'][0], ARIMA_results['Torino Enjoy'][1]))
axes[2].plot(ARIMA_results['Torino Enjoy'][0], color='black', label='Real Rentals', linewidth=3)
axes[2].plot(ARIMA_results['Torino Enjoy'][1], color='limegreen', label='Prediction', linewidth=2)
axes[2].bar(np.arange(0,len(ARIMA_results['Torino Enjoy'][0]),1), error, color='orangered', label='Error', width=1)
axes[2].set_title(f"Torino (Enjoy): ARIMA ({p_values['Torino Enjoy']},0,{q_values['Torino Enjoy']}) with Expanding Window", fontsize=18)
axes[2].legend(loc='best')
axes[2].set_xlabel('Test [h]')
axes[2].set_ylim(0, 500)
plt.savefig('S3_T6.png', dpi=300)
plt.show()
# Table for total errors
error_results = [get_errors(ARIMA_results['Washington DC'][0], ARIMA_results['Washington DC'][1]),get_errors(ARIMA_results['Torino'][0], ARIMA_results['Torino'][1]),get_errors(ARIMA_results['Tori
print(pd.DataFrame(error_results, index=["Washington DC (Car2Go)", "Torino (Car2Go)", "Torino(Enjoy)"],columns=["MSE", "MAE", "MAPE", "R2"]))
```

```
input_list = []
for city in cities:
    for p in range(0, p_values[city]+1):
        for q in range(0, q_values[city]+1):
            if p==0 and q==0:
                continue
            input_list.append([results[city], city, p, q])
```

```
def ARIMA_Thread(input_list):
    data, city, p, q = input_list
    days_to_train = 21
    days_to_test = 7
    original, predictions = ARIMA_fit(data, city, p=p, q=q, train_size=days_to_train*24,test_size=days_to_test*24, mode="expanding")
    MSE, MAE, MAPE, R2 = get_errors(original, predictions)
    return [original, predictions, city, p, q, MSE, MAE, MAPE, R2]
```

```
counter = 0
ARIMA_GridSearch = []
tic1 = time.perf_counter()
for inp in input_list:
    counter += 1
    tic2 = time.perf_counter()
    print(f"Input {counter}: City={inp[1]}-P={inp[2]}-Q={inp[3]}")
    ARIMA_GridSearch.append(ARIMA_Thread(inp))
    toc2 = time.perf_counter()
    print(f"Iteration Finished in {toc2-tic2} seconds.")
toc1 = time.perf_counter()
print(f"Job Finished in {toc1-tic1} seconds.")
headers = ['City', 'p', 'q', 'MSE', 'MAE', 'MAPE', 'R2', 'Original', 'Prediction']
ARIMA_GridSearch_Results = pd.DataFrame(columns=headers)
# Organize results in DataFrame
for elem in ARIMA_GridSearch:
    ARIMA_GridSearch_Results.loc[len(ARIMA_GridSearch_Results)] = [elem[2], elem[3], elem[4], elem[5], elem[6], elem[7], elem[8], elem[0], elem[1]]
```

```
ARIMA_GridSearch_Results.to_csv('GridSearchV2.csv')
ARIMA_GS = pd.read_csv('GridSearchV2.csv')
ARIMA_GS = ARIMA_GridSearch_Results.copy()
```

```
ARIMA_GS_ALLP = ARIMA_GS
ARIMA_ALL_P = ARIMA_GS_ALLP[(ARIMA_GS_ALLP['q']==2)].drop(columns=['q', 'MSE', 'MAE', 'MAPE', 'R2'])
plt.rc('axes', labelsize=24)
plt.rc('ytick', labelsize=22)
plt.rc('xtick', labelsize=22)
plt.rc('legend', fontsize=18)
plt.rc('figure', titlesize=18)
plt.rcParams.update({'axes.titlesize': 'small'})
plt.figure(figsize=(22, 6), tight_layout=True)
axes = plt.plot(ARIMA_results['Torino Enjoy'][0], color='black', label='Real Bookings',linewidth=4)
for p in range(0, p_values['Torino Enjoy']+1):
    axes = plt.plot((ARIMA_ALL_P[(ARIMA_ALL_P['City']=='Torino Enjoy')&(ARIMA_ALL_P['p']==p)]['Prediction'].iloc[0]), label=f'P = {p}', linewidth=2)
plt.title(f"Torino (Enjoy): ARIMA with Expanding Window", fontsize=26)
plt.legend(loc='upper left')
plt.xlabel('Test [h]')
plt.ylabel('No. Rentals')
plt.xlim(0, 175)
plt.ylim(0, 300)
plt.savefig('S3_T6_TorinoEnjoy_AllP.png', dpi=300)
plt.show()
```

2.7 Parameters Impact

2.7.1 Finding Best Parameters(p and q)

```
GS_dict = {}
for city in cities:
    GS_MAPE = pd.DataFrame(index=[x for x in range(0, p_values[city]+1)], columns=[x for x in range(0, q_values[city]+1)])
    for p in range(0, p_values[city]+1):
        for q in range(0, q_values[city]+1):
            temp = ARIMA_GS[ARIMA_GS['City']==city].drop(columns=['City', 'MSE', 'MAE','R2', 'Original', 'Prediction'])
            if p==0 and q==0:
                GS_MAPE.iloc[p,q] = np.nan # P=0, Q=0? No way!
            else:
                GS_MAPE.iloc[p,q] = round(list(temp.loc[(temp['p']==int(p)) & (temp['q']==int(q)), 'MAPE'])[0],4)
    GS_dict[city] = pd.DataFrame(GS_MAPE.values.tolist(),columns = GS_MAPE.columns.tolist(), index =GS_MAPE.index.tolist())
```

```
plt.rc('axes', labelsize=22)
plt.rc('ytick', labelsize=20)
plt.rc('xtick', labelsize=20)
plt.rc('legend', fontsize=10)
plt.rc('figure', titlesize=18)
plt.rcParams.update({'axes.titlesize': 'xx-large'})
plt.figure(tight_layout=True)
cmap = LinearSegmentedColormap.from_list(
    name='ARIMA_GS',
    colors=['green','white','red']
)
for city in cities:
    plt.figure(figsize=(9, 5))
    data_matrix = GS_dict[city].to_numpy()
    min_in_each_column = np.sort(data_matrix.flatten())[2:3]
    axes = sns.heatmap(data_matrix,linewidth=0.8,annot=True,cmap=cmap,fmt=".2%",vmin=0,vmax=1,annot_kws={"size": 22, 'color':'k'})
# Mask for Bolding the 2 minimum values
    axes = sns.heatmap(data_matrix,mask=data_matrix >= min_in_each_column,annot_kws={"weight": "bold", "size": 22, 'color':'k'},linewidth=0.8,annot=True,cbar=False,cmap=cmap,fmt=".2%",vmin=0,vma
    axes.set(xlabel='Q for MA', ylabel='P for AR')
    if 'Enjoy' in city:
        title = f'MAPE for Torino (Enjoy)'
    else:
        title = f'MAPE for {city} (Car2Go)'
    axes.set_title(title, fontsize=24)
    axes.invert_yaxis()
    plt.tight_layout()
    plt.savefig(f"S3_T7_A_{city}.png", dpi=300)
    plt.show()
```

2.7.2 Change Learning Strategy and Training Sample Size

```
days_to_train = np.arange(7,22,2)
days_to_test = 7
modes = ["expanding", "sliding"]
counter = 0
ARIMA_NSearch = []
tic1 = time.perf_counter()
for city in cities:
    for mode in modes:
        for period in days_to_train:
            counter += 1
            tic2 = time.perf_counter()
            print(f"Input {counter}: Period={period}-City={city}-P={p_values_best[city]}-Q={q_values_best[city]}-Mode={mode}")
            original, predictions = ARIMA_fit(results[city], city, p=p_values_best[city], q=q_values_best[city],train_size=period*24, test_size=days_to_test*24,mode=mode)
            [MSE, MAE, MAPE, R2] = get_errors(original, predictions)
            ARIMA_NSearch.append([city, period, mode, MSE, MAE, MAPE, R2, original, predictions])
            toc2 = time.perf_counter()
            print(f"Iteration Finished in {toc2-tic2} seconds.")
toc1 = time.perf_counter()
print(f"Job Finished in {toc1-tic1} seconds.")
```

```
headers = ['City', 'N', 'Mode', 'MSE', 'MAE', 'MAPE', 'R2', 'Original', 'Prediction']
ARIMA_NSearch_Results = pd.DataFrame(columns=headers)
for elem in ARIMA_NSearch:
    ARIMA_NSearch_Results.loc[len(ARIMA_NSearch_Results)] = elem
ARIMA_NSearch_Results.to_csv('NSearchV2.csv')
```

```
ARIMA_GNS = ARIMA_NSearch_Results.drop(columns=['MSE', 'MAE', 'R2', 'Original', 'Prediction'])
```

```
GNS_dict = {}
for city in cities:
    GNS_dict[city] = ARIMA_GNS[ARIMA_GNS['City']==city]
```

```
plt.rc('axes', labelsize=28)
plt.rc('ytick', labelsize=24)
plt.rc('xtick', labelsize=24)
plt.rc('legend', fontsize=20)
plt.rc('figure', titlesize=26)
plt.rcParams.update({'axes.titlesize': 'small'})
plt.figure(figsize=(10, 8), tight_layout=True)
temp_city = GNS_dict['Washington DC']
#Expanding Line Plot
N_plot_seattle = sns.lineplot(x=temp_city[temp_city['Mode']=='expanding']['N'],y=temp_city[temp_city['Mode']=='expanding']['MAPE']*100,data=temp_city, marker="o", label='Expanding Window Strategy
#Sliding Line Plot
N_plot_seattle = sns.lineplot(x=temp_city[temp_city['Mode']=='sliding']['N'],y=temp_city[temp_city['Mode']=='sliding']['MAPE']*100,data=temp_city, marker="o", label='Sliding Window Strategy',line
N_plot_seattle.set_xlabel('No. Past Samples')
N_plot_seattle.set_ylabel('MAPE %')
plt.tight_layout()
plt.xticks(np.arange(7,22,2))
plt.yticks(np.arange(35,90,5))
plt.legend()
plt.title('MAPE for Washington DC (Car2Go)', fontsize=32)
plt.savefig(f"S3_T7_BC_Washington DC.png", dpi=300)
plt.show()
plt.figure(figsize=(10, 8), tight_layout=True)
temp_city = GNS_dict['Torino']
#Expanding Line Plot
N_plot_Milano = sns.lineplot(x=temp_city[temp_city['Mode']=='expanding']['N'],y=temp_city[temp_city['Mode']=='expanding']['MAPE']*100,data=temp_city, marker="o", label='Expanding Window Strategy'
#Sliding Line Plot
N_plot_Milano = sns.lineplot(x=temp_city[temp_city['Mode']=='sliding']['N'],y=temp_city[temp_city['Mode']=='sliding']['MAPE']*100,data=temp_city, marker="o", label='Sliding Window Strategy',linew
N_plot_Milano.set_xlabel('No. Past Samples')
N_plot_Milano.set_ylabel('MAPE %')
plt.tight_layout()
plt.xticks(np.arange(7,22,2))
plt.yticks(np.arange(35,55,5))
plt.legend()
plt.title('MAPE for Torino (Car2Go)', fontsize=32)
plt.savefig(f"S3_T7_BC_Torino.png", dpi=300)
plt.show()
plt.figure(figsize=(10, 8), tight_layout=True)
temp_city = GNS_dict['Torino Enjoy']
#Expanding Line Plot
N_plot_Milano_enj = sns.lineplot(x=temp_city[temp_city['Mode']=='expanding']['N'],y=temp_city[temp_city['Mode']=='expanding']['MAPE']*100,data=temp_city, marker="o", label='Expanding Window Strat
#Sliding Line Plot
N_plot_Milano_enj = sns.lineplot(x=temp_city[temp_city['Mode']=='sliding']['N'],y=temp_city[temp_city['Mode']=='sliding']['MAPE']*100,data=temp_city, marker="o", label='Sliding Window Strategy',
N_plot_Milano_enj.set_xlabel('No. Past Samples')
N_plot_Milano_enj.set_ylabel('MAPE %')
plt.tight_layout()
plt.xticks(np.arange(7,22,2))
plt.yticks(np.arange(35,55,5))
plt.legend()
plt.title('MAPE for Torino (Enjoy)', fontsize=32)
plt.savefig(f"S3_T7_BC_TorinoEnjoy.png", dpi=300)
plt.show()
```

2.7.3 Optimal Hyper-parameters

```
days_to_train = 21
p_values_best = {'Washington DC':2, 'Torino':2, 'Torino Enjoy':2}
q_values_best = {'Washington DC':4, 'Torino':4, 'Torino Enjoy':4}
ARIMA_results = {}
tic = time.perf_counter()
for city in cities:
    print(city)
    ARIMA_results[city] = ARIMA_fit(results[city], city, p=p_values_best[city], q=q_values_best[city],train_size=days_to_train*24, test_size=days_to_test*24,mode="expanding")
toc = time.perf_counter()
print(f"Job Finished in {toc-tic} seconds.")
```

```
plt.rc('axes', labelsize=20)
plt.rc('ytick', labelsize=20)
plt.rc('xtick', labelsize=20)
plt.rc('legend', fontsize=14)
plt.rc('figure', titlesize=18)
plt.rcParams.update({'axes.titlesize': 'small'})
fig, axes = plt.subplots(1, 3, figsize=(22, 5), tight_layout=True)
fig.tight_layout()
error = np.abs(np.subtract(ARIMA_results['Washington DC'][0], ARIMA_results['Washington DC'][1]))
axes[0].plot(ARIMA_results['Washington DC'][0], color='black',label='Real Bookings', linewidth=3)
axes[0].plot(ARIMA_results['Washington DC'][1], color='limegreen', label='Prediction', linewidth=2)
axes[0].bar(np.arange(0,len(ARIMA_results['Washington DC'][0]),1), error, color='orangered', label='Error', width=1)
axes[0].set_title(f"Washington DC (Car2Go): ARIMA ({p_values_best['Washington DC']},0,{q_values_best['Washington DC']}) with Expanding Window", fontsize=18)
axes[0].legend(loc='best')
axes[0].set_xlabel('Test [h]')
axes[0].set_ylabel('No. Rentals')
axes[0].set_ylim(0, 250)
error = np.abs(np.subtract(ARIMA_results['Torino'][0], ARIMA_results['Torino'][1]))
axes[1].plot(ARIMA_results['Torino'][0], color='black', label='Real Bookings', linewidth=3)
axes[1].plot(ARIMA_results['Torino'][1], color='limegreen', label='Prediction', linewidth=2)
axes[1].bar(np.arange(0,len(ARIMA_results['Torino'][0]),1), error, color='orangered', label='Error', width=1)
axes[1].set_title(f"Torino (Car2Go): ARIMA ({p_values_best['Torino']},0,{q_values_best['Torino']}) with Expanding Window", fontsize=18)
axes[1].legend(loc='best')
axes[1].set_xlabel('Test [h]')
axes[1].set_ylim(0, 500)
error = np.abs(np.subtract(ARIMA_results['Torino Enjoy'][0], ARIMA_results['Torino Enjoy'][1]))
axes[2].plot(ARIMA_results['Torino Enjoy'][0], color='black', label='Real Bookings', linewidth=3)
axes[2].plot(ARIMA_results['Torino Enjoy'][1], color='limegreen', label='Prediction', linewidth=2)
axes[2].bar(np.arange(0,len(ARIMA_results['Torino Enjoy'][0]),1), error, color='orangered', label='Error', width=1)
axes[2].set_title(f"Torino (Enjoy): ARIMA ({p_values_best['Torino Enjoy']},0,{q_values_best['Torino Enjoy']}) with Expanding Window", fontsize=18)
axes[2].legend(loc='best')
axes[2].set_xlabel('Test [h]')
axes[2].set_ylim(0, 500)
plt.savefig('S3_T7_ARIMAFinal.png', dpi=300)
plt.show()
error_results = [
get_errors(ARIMA_results['Washington DC'][0], ARIMA_results['Washington DC'][1]),
get_errors(ARIMA_results['Torino'][0], ARIMA_results['Torino'][1]),
get_errors(ARIMA_results['Torino Enjoy'][0], ARIMA_results['Torino Enjoy'][1])]
print(pd.DataFrame(error_results, index=["Washington DC (Car2Go)", "Torino (Car2Go)", "Torino(Enjoy)"],columns=["MSE", "MAE", "MAPE", "R2"]))
```