UNIVERSITI SAINS MALAYSIA

CDS513 PREDICTIVE BUSINESS ANALYTICS

School of Computer Sciences, USM, Penang

ASSIGNMENT 2

TIMES SERIES AND FORECASTING

"Minimum Daily Temperatures Dataset"

CHAN HUAN YANG P-COM0068/19

Step 1: Description of the problem

Climate, one of the main elements of the natural environment, has a determining role in natural and human life. Temperature, being one of the most important climatic parameters, has a direct impact on the evaporation, snow melting, frost and an indirect impact on the atmospheric stability and precipitation conditions.

There are several reasons why temperature forecasts are important. They would certainly be missed if they were not there. It is a product of science that impacts the lives of many people. The following is a list of various reasons why temperature forecasts are important:

- Helps people prepare for how to dress (i.e. warm weather, cold weather)
- Helps businesses and people plan for power production and how much power to use (i.e. power companies, where to set thermostat)
- Helps farmers and gardeners plan for crop irrigation and protection (irrigation scheduling, freeze protection)
- Helps people plan outdoor activities

This dataset describes the minimum daily temperatures over 10 years (1981-1990) in the city Melbourne, Australia. The units are in degrees Celsius. The source of the data is credited as the Australian Bureau of Meteorology. The important statistical properties of the temperature time series are given in Figure 1. The entire record (3650 days) was divided into two parts as training (date < '1990-01-01') and testing (date >= '1990-01-01') periods.

Figure 1: Statistical properties of the Temperature Time Series

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3650 entries, 0 to 3649
Data columns (total 2 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   Date    3650 non-null   object
 1   Temp    3650 non-null   float64
dtypes: float64(1), object(1)
memory usage: 57.2+ KB
None
              Temp
count  3650.000000
mean     11.177753
std       4.071837
min       0.000000
25%       8.300000
50%      11.000000
75%      14.000000
max      26.300000
min date:1981-01-01 00:00:00, max date:1990-12-31 00:00:00
```

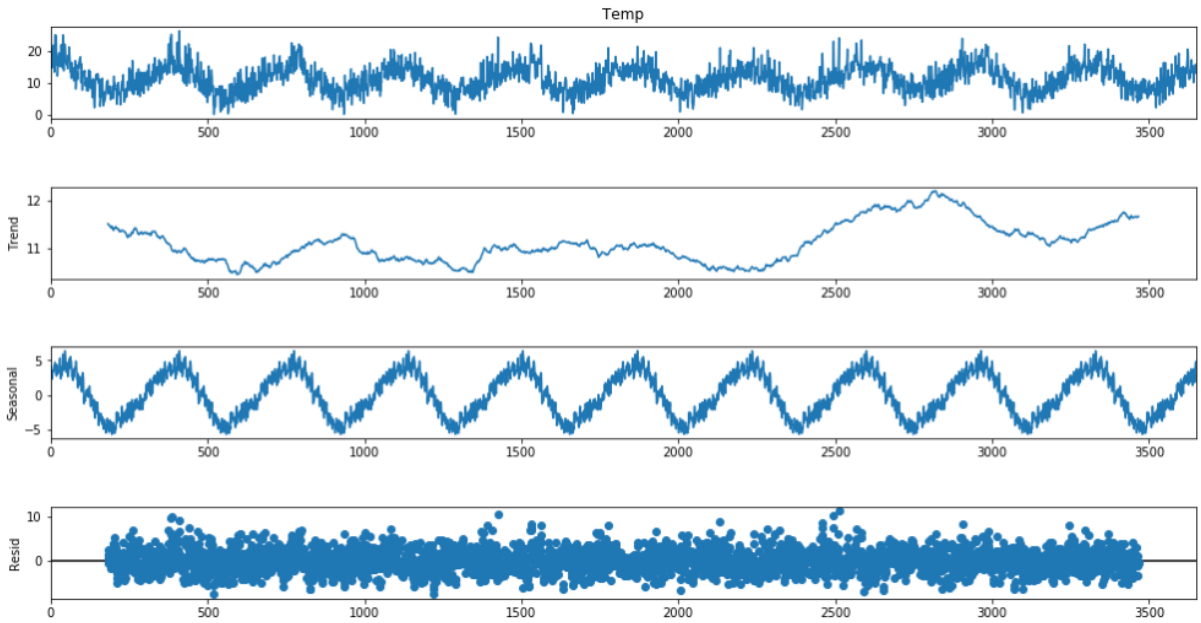
There are 4 components of Time Series:

- Trend - Upward & downward movement of the data with time over a large period of time. Eq: Appreciation of Dollar vs rupee.
- Seasonality - seasonal variances. Eq: Ice cream sales increases in Summer only
- Noise or Irregularity - Spikes & troughs at random intervals

- Cyclicity - behavior that repeats itself after large interval of time, like months, years etc.

We can use Python seasonal_decompose method to separate our Time Series data into four graphs (Observed, Trend, Seasonal, Residual). The frequency chosen is 365 because this data is long term.

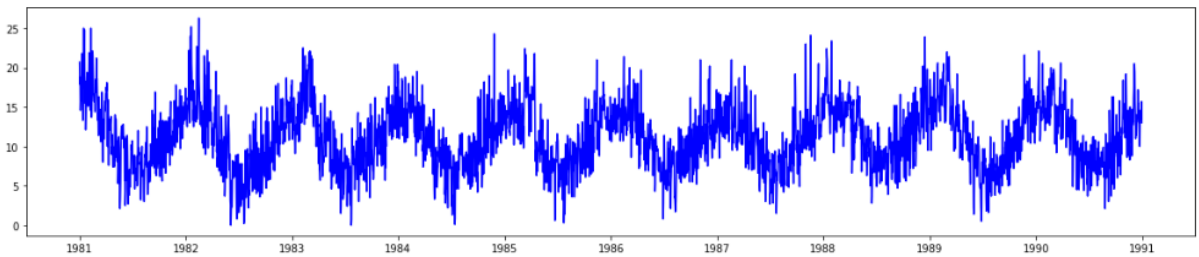
Figure 2: Observed, trend, seasonal, residual plot of the Temperature Time Series



As we can see from the plots, our data showing seasonal properties (temperature tends to be higher during beginning and ending of the years). Our data also showing cyclic properties as the temperature pattern repeats itself after 1 year (365 days). However there is no clear evident that there is an trend in the data.

Step 2: Plot given dataset as time series

Figure 3: Chart of the Temperature Time Series



Step 3: Difference data to make data stationary

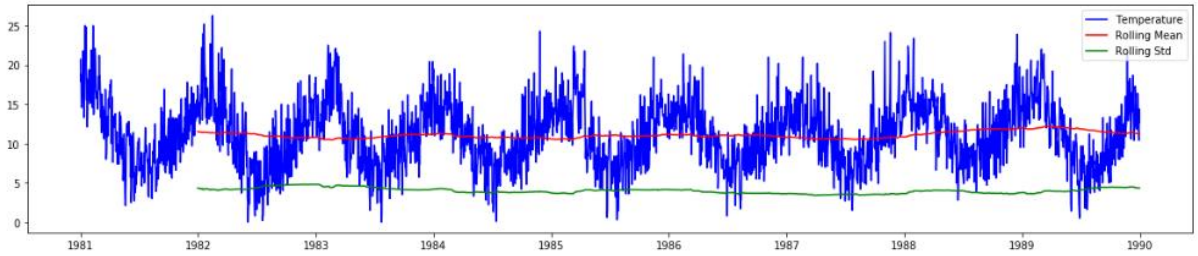
Trend & Seasonality are two reasons why a Time Series is not stationary & hence need to be corrected. Before applying any statistical model on a Time Series, the series has to be stationary, which means that, over different time periods,

- It should have constant mean.
- It should have constant variance or standard deviation.
- Auto-covariance should not depend on time.

I've done two ways to check for stationarity of a Time Series:

- Rolling Statistics - Plot the moving average or moving standard deviation to see if it varies with time. It is a visual technique.
- ADCF Test - Augmented Dickey–Fuller test is used to gives us various values that can help in identifying stationarity. The Null hypothesis says that a TS is non-stationary. It comprises of a Test Statistics & some critical values for some confidence levels. If the Test statistics is less than the critical values, we can reject the null hypothesis & say that the series is stationary. THE ADCF test also gives us a p-value. Acc to the null hypothesis, lower values of p is better.

Figure 4: Rolling statistics and ADCF Test of the Temperature Time Series




```
ADF Statistic: -4.345532
p-value: 0.000370
```

After performing the stationarity test both the methods return that the Time Series is stationary. So no preprocessing is required, however still time series differentiation has been performed. The lag value is 365. The reason for this is that the series is daily temperature data (365 days).

Differencing is a method of transforming a time series dataset. It can be used to remove the series dependence on time, so-called temporal dependence. This includes structures like trends and seasonality. Differencing is performed by subtracting the previous observation from the current observation. However, inverting the process is required when a prediction must be converted back into the original scale. This process can be reversed by adding the observation at the prior time step to the difference value. In this way, a series of differences and inverted differences can be calculated.

I am differencing the dataset manually. This involves developing a new function that creates a differenced dataset. The function would loop through a provided series and calculate the differenced values at the specified interval or lag. We can see that the function is careful to begin the differenced dataset after the specified interval to ensure differenced values can, in fact, be calculated. A default interval or lag value of 1 is defined. This is a sensible default. Another function named inverse_difference() also was created to invert the difference operation for a single forecast. It requires that the real observation value for the previous time step also be provided.

Figure 5: Custom function to difference and invert different series

```
# create a differenced series
def difference(dataset, interval=1):
    diff = list()
    for i in range(interval, len(dataset)):
        value = dataset[i] - dataset[i - interval]
        diff.append(value)
    return np.array(diff)

# invert differenced value
def inverse_difference(history, yhat, interval=1):
    return yhat + history[-interval]
```

Next, the difference transform is applied and the result is plotted. The plot shows that the seasonality signal is no longer obvious now.

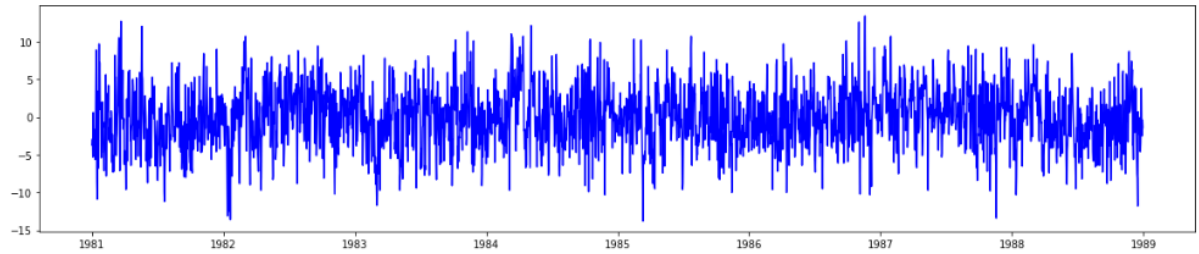
Figure 6: Line plot of temperature after differencing

```
# Differencing to remove seasonality
day_in_year = 365
train_differenced = difference(train['Temp'], day_in_year)
train_differ = pd.DataFrame(list(zip(train['Date'].tolist(), train_differenced)), columns =['Date', 'Temp'])

# Line plot after differencing
plt.figure(figsize=(20,4))
plt.plot(train_differ['Date'], train_differ['Temp'], 'b')
```

executed in 364ms, finished 22:16:51 2020-04-19

```
[<matplotlib.lines.Line2D at 0x27ba30fc1c8>]
```

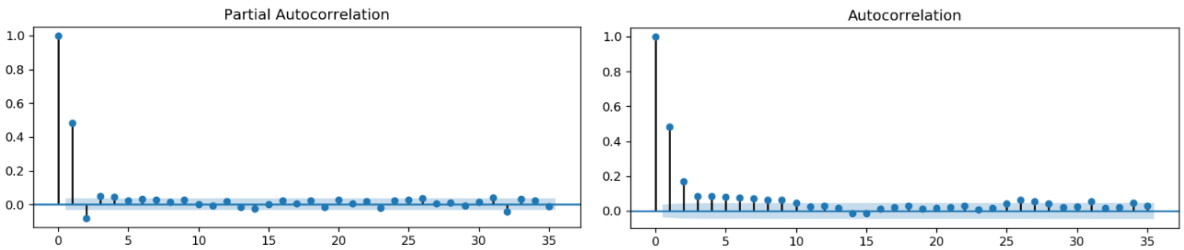


I am now using the seasonally differenced as my dataset. I also plot the partial autocorrelation (PACF) and autocorrelation (ACF) chart to find the order of AR term (p) and MA term (q) respectively. Below are some observations from the plots:

- The ACF shows first significant lag at 1
- The PACF shows first significant lag at 1
- A good starting point for the p and q values is also 1.

This quick analysis suggests an ARIMA(1,0,1) on the stationary data may be a good starting point.

Figure 7: Partial autocorrelation (PACF) and Autocorrelation (ACF) chart



Step 4: Model

ARIMA

The entire record (3650 days) was divided into two parts as training set (date < '1990-01-01') and testing set (date >= '1990-01-01') periods. The ACF and PACF plots suggest that an ARIMA(1,0,1) or similar may be the best that I can do. To confirm this analysis, I can grid search a suite of ARIMA hyperparameters and check that no models result in better out of sample MAE performance.

I will search values of p, d, and q for combinations (skipping those that fail to converge), and find the combination that results in the best performance on the test set. I will use a ARIMA grid search class to explore all combinations in a subset of integer values. I will search all combinations of the following parameters:

- p: 1 to 3
- d: 0
- q: 1 to 3

Figure 8: ARIMA grid search class

```
# Grid Search ARIMA Hyperparameters Class
class grid_search_arima(object):
    def __init__(self, ori_train_list, train_list, test_list, p_list, d_list, q_list, step, seasonal_day):
        self.ori_train_list = ori_train_list
        self.train_list = train_list
        self.test_list = test_list
        self.p_list = p_list
        self.d_list = d_list
        self.q_list = q_list
        self.step = step
        self.seasonal_day = seasonal_day
        self.model_dict = {}

    def run(self):
        history = self.ori_train_list
        for p in self.p_list:
            for d in self.d_list:
                for q in self.q_list:
                    try:
                        model = ARIMA(self.train_list, order=(p, d, q))
                        model_fit = model.fit(disp=0)
                        forecast = model_fit.forecast(steps=self.step)[0]

                        history = self.ori_train_list.copy()
                        predicted = []
                        day = 1

                        for yhat in forecast:
                            inverted = inverse_difference(history, yhat, self.seasonal_day)
                            history.append(inverted)
                            predicted.append(inverted)
                            day += 1

                        model_name = "{},{},{}".format(p,d,q)
                        self.model_dict[model_name] = {'aic':model_fit.aic,
                                                       'bic':model_fit.bic,
                                                       'res':model_fit.resid.tolist(),
                                                       'prediction':predicted,
                                                       'mae':mean_absolute_error(self.test_list, predicted),
                                                       'mre':mean_relative_error(predicted,self.test_list)}
                        print ("({},{},{}) success".format(p,d,q))
                        print(model_fit.summary())

                    except:
                        print ("({},{},{}) failed".format(p,d,q))
```

Supervised Machine Learning

Same as ARIMA, the entire record (3650 days) was divided into two parts as training set (date < '1990-01-01') and testing set (date >= '1990-01-01') periods. However extra steps have to be done to transform the time series to supervised machine learning by adding lags. Lags are basically the shift of the data one step or more backward in the time. Total 10 different lag values have been chosen. They are 1, 2, 3, 4, 5, 6, 7 ,365,

730, 1095. Lags 365, 730, and 1095 were chosen to represent the yearly cyclic/seasonal nature of data.

Figure 9: Dataset for supervised machine learning

	Date	Lag_1	Lag_2	Lag_3	Lag_4	Lag_5	Lag_6	Lag_7	Lag_365	Lag_730	Lag_1095	Temp
1095	1984-01-01	18.0	20.4	16.1	11.1	13.9	15.0	17.5	18.4	17.0	20.7	19.5
1096	1984-01-02	19.5	18.0	20.4	16.1	11.1	13.9	15.0	15.0	15.0	17.9	17.1
1097	1984-01-03	17.1	19.5	18.0	20.4	16.1	11.1	13.9	10.9	13.5	18.8	17.1
1098	1984-01-04	17.1	17.1	19.5	18.0	20.4	16.1	11.1	11.4	15.2	14.6	12.0
1099	1984-01-05	12.0	17.1	17.1	19.5	18.0	20.4	16.1	14.8	13.0	15.8	11.0
...	...	...	...	...	...	...	...	...	...	...	...	...
3645	1990-12-27	14.6	12.9	10.0	13.9	13.2	13.1	15.4	13.3	9.5	16.2	14.0
3646	1990-12-28	14.0	14.6	12.9	10.0	13.9	13.2	13.1	11.7	12.9	14.2	13.6
3647	1990-12-29	13.6	14.0	14.6	12.9	10.0	13.9	13.2	10.4	12.9	14.3	13.5
3648	1990-12-30	13.5	13.6	14.0	14.6	12.9	10.0	13.9	14.4	14.8	13.3	15.7
3649	1990-12-31	15.7	13.5	13.6	14.0	14.6	12.9	10.0	12.7	14.1	16.7	13.0

2555 rows × 12 columns

Total two different supervised machine learning models are created to predict the minimum temperature of day I year 1990. They are Random Forest Regression and Support Vector Regression. There is no intuition for the hyperparameter setting in both the models.

Random Forest is an ensemble machine learning technique capable of performing both regression and classification tasks using multiple decision trees and a statistical technique called bagging. Meanwhile, Support Vector Regression(SVR) is quite different than other Regression models. It uses the Support Vector Machine(SVM, a classification algorithm) algorithm to predict a continuous variable.

Figure 10: Python's code of Random Forest Regression

```
# Instantiate RandomForestRegressor object
regr = RandomForestRegressor(n_estimators=20, random_state=0, criterion='mae')

# Fit regressor object with training set
regr.fit(X_train, y_train)

# Make prediction using regressor model
regr_prediction = regr.predict(X_test).tolist()
```

Figure 11: Python's code of Support Vector Regression

```
# Instantiate Support Vector Regressor object
clf = SVR(C=1.0, epsilon=0.3)

# Fit regressor object with training set
clf.fit(X_train, y_train)

# Make prediction using regressor model
clf_prediction = clf.predict(X_test).tolist()
```

Step 5: Results and Discussions

ARIMA

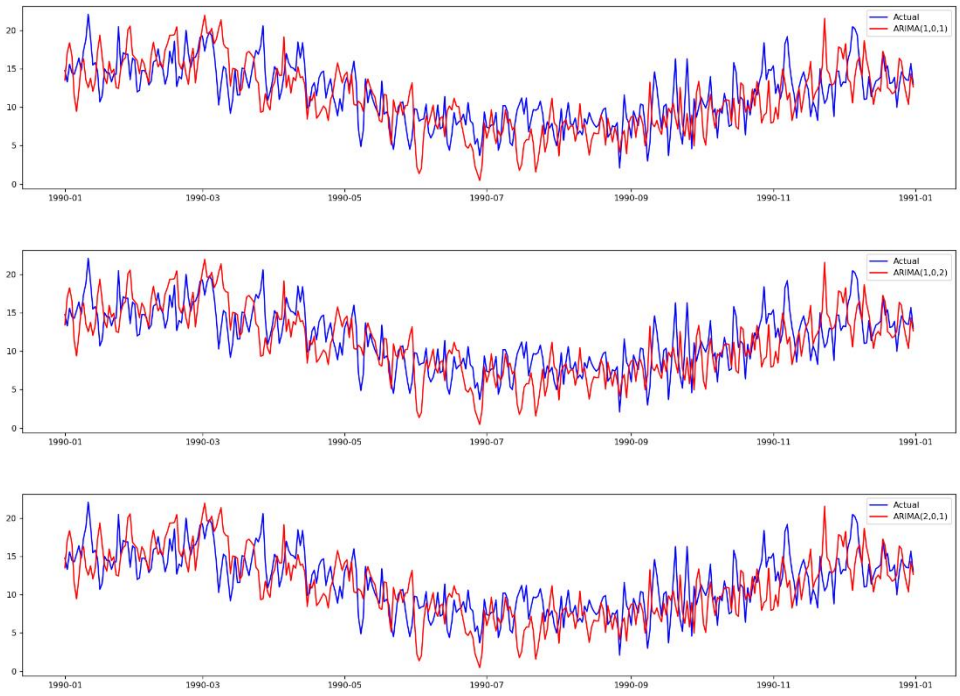
Below are the model results and reports generated from those ARIMA models that converge without error:

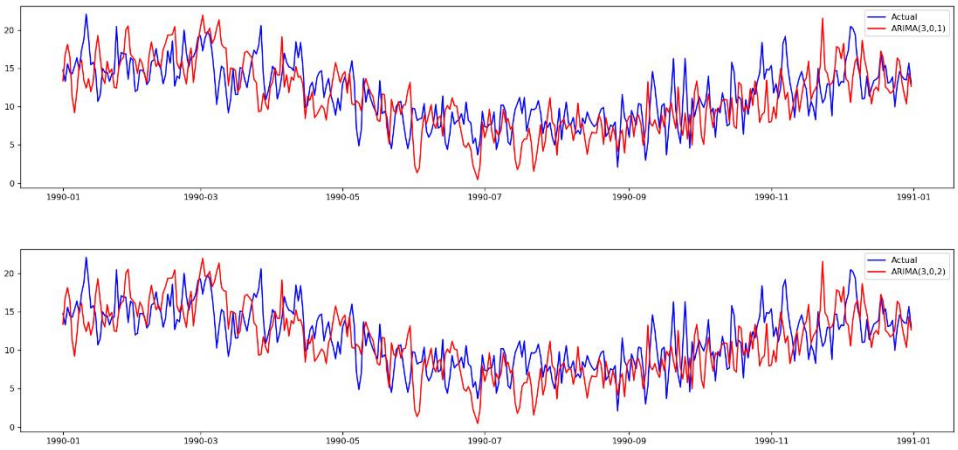
Figure 12: Performance of ARIMA models

p,d,q	aic	bic	mae	mre
1,0,1	15537	15561	2.87	0.28
1,0,2	15536	15566	2.87	0.28
2,0,1	15539	15569	2.87	0.28
3,0,1	15524	15560	2.87	0.28
3,0,2	15526	15568	2.87	0.28

The Akaike Information Criteria (AIC) is a widely used measure of a statistical model. It basically quantifies the goodness of fit, and the simplicity/parsimony, of the model into a single statistic. When comparing multiple models, the one with the lower AIC is generally “better”. By checking the Figure 12, ARIMA model with parameter (3,0,1) is the best fit ARIMA model as it showing lowest AIC and BIC values.

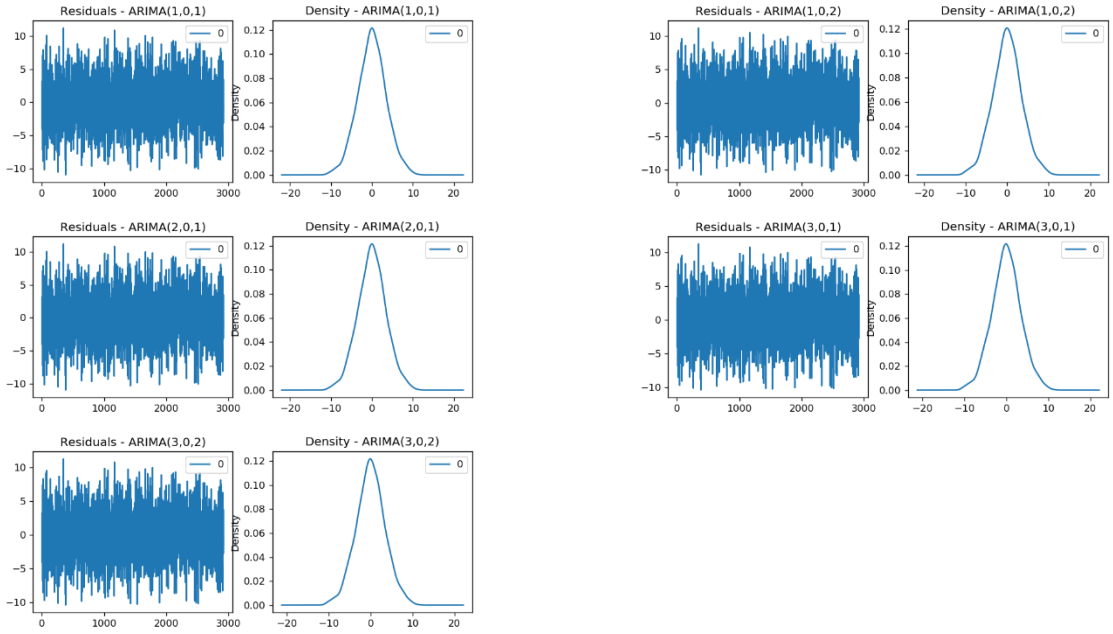
Figure 13: Actual temperature vs predicted temperature using ARIMA





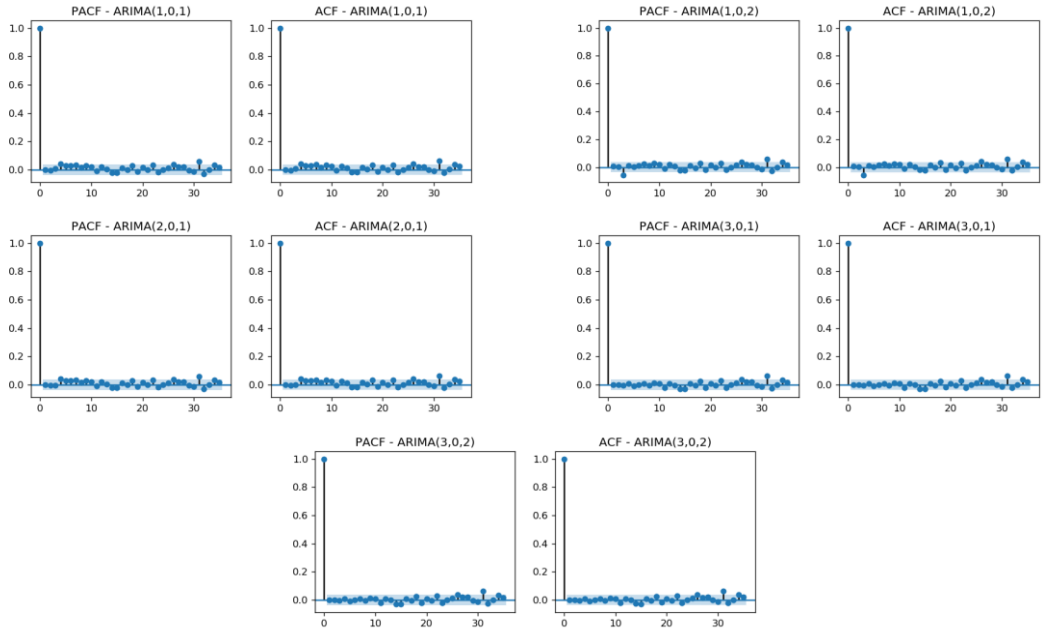
A good final check of models is to review residual forecast errors. Ideally, the distribution of residual errors should be a Gaussian with a zero mean. The plots (Figure 14) of the residual errors shows that the mean values of all ARIMA are very close to zero.

Figure 14: Residual errors of ARIMA models



It is also a good idea to check the time series of the residual errors for any type of autocorrelation. If present, it would suggest that the model has more opportunity to model the temporal structure in the data. By checking Figure 15, The results suggest that very little autocorrelation is present in the time series has been captured by the models.

Figure 15: ACF and PACF of residuals of ARIMA models



Supervised Machine Learning

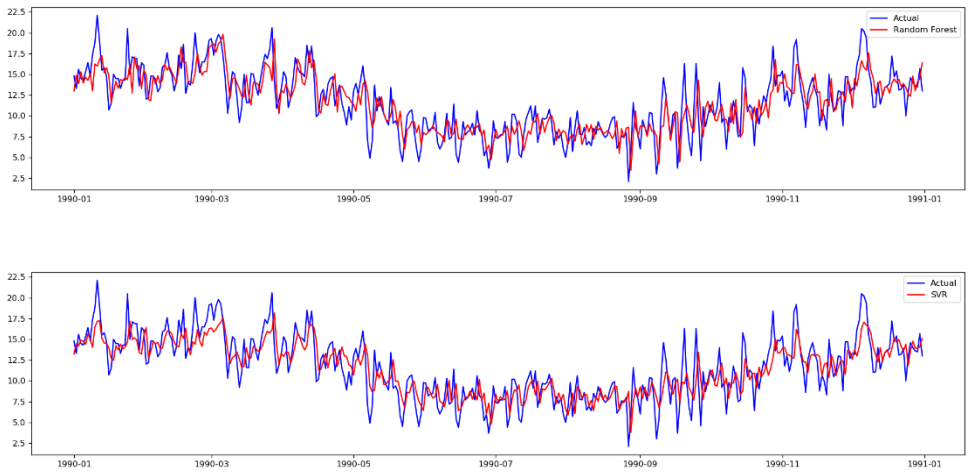
Below are the model results and reports generated from Random Forest Regression and Support Vector Regression.

Figure 16: Performance of Random Forest Regression and Support Vector Regression

Metric	Random Forest (RFR)	Support Vector (SVR)
Mean Absolute Error	1.83	1.75
Mean Relative Error	0.19	0.18

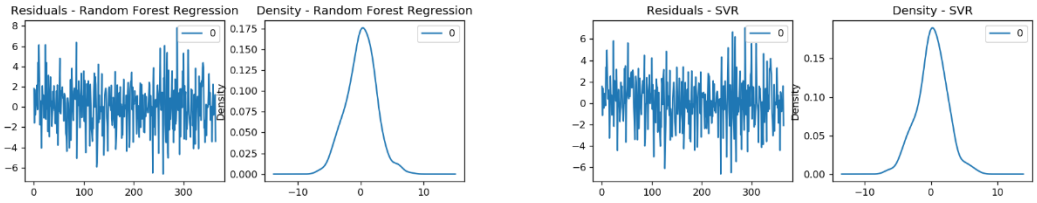
By checking the Figure 16, Support Vector Regression clearly is the better model as it having lowest mean absolute error and mean relative error.

Figure 17: Actual temperature vs predicted temperature using RFR and SVR



The plots (Figure 18) of the residual errors also shows that the mean values of both models are very close to zero.

Figure 18: Residual errors of RFR and SVR



ARIMA vs Supervised Machine Learning

In general by comparing using metric mean absolute error, supervised machine learning are more accurate than the ARIMA models. Support Vector Regression has the lowest mean absolute error among all the models hence it is the best model.

Figure 19: Performance of ARIMA and supervised machine learning models

Model	Mean Absolute Error	Mean Relative Error	Rank
Support Vector Regression	1.75	0.18	1st
Random Forest Regression	1.83	0.19	2nd
ARIMA (1,0,1)	2.87	0.28	3rd
ARIMA (1,0,2)	2.87	0.28	
ARIMA (2,0,1)	2.87	0.28	
ARIMA (3,0,1)	2.87	0.28	
ARIMA (3,0,2)	2.87	0.28	