Time Series Analysis and Forecasting:

Minimum Temperature Forecasting.

Puja Ramesh Sonawane.

University of Central Florida.

Data Mining II

Presented To: Dr. Aron Smith

UCF

Table of Contents

Introduction

This project tries to address the forecasting problem where the task is to forecast the minimum temperature for one future period.

Dataset used in this project have daily recorded minimum temperature over 10 years from 1981-1990. Data shows linear trend and yearly seasonality. To address the problem several models are built and analyzed such as Persistence Model, Autoregression Model, Moving Average Model, ARIMA Model, SARIMA model and Neural Network Model. After the analysis, it is observed that Autoregression Model and Neural Network Model gave better results among all models. For Neural network model, several features are generated like date-time features, lag features and window features. Data is divided into training and validation data for training of model and evaluation models.

Finally, Neural Network Model and Autoregression Model is selected as reliable model among all, on the basis of evaluation criteria used.

Keywords: Forecasting, Persistence Model, Autoregression Model, Moving Averages, SARIMA, Neural Network, Feature Engineering.

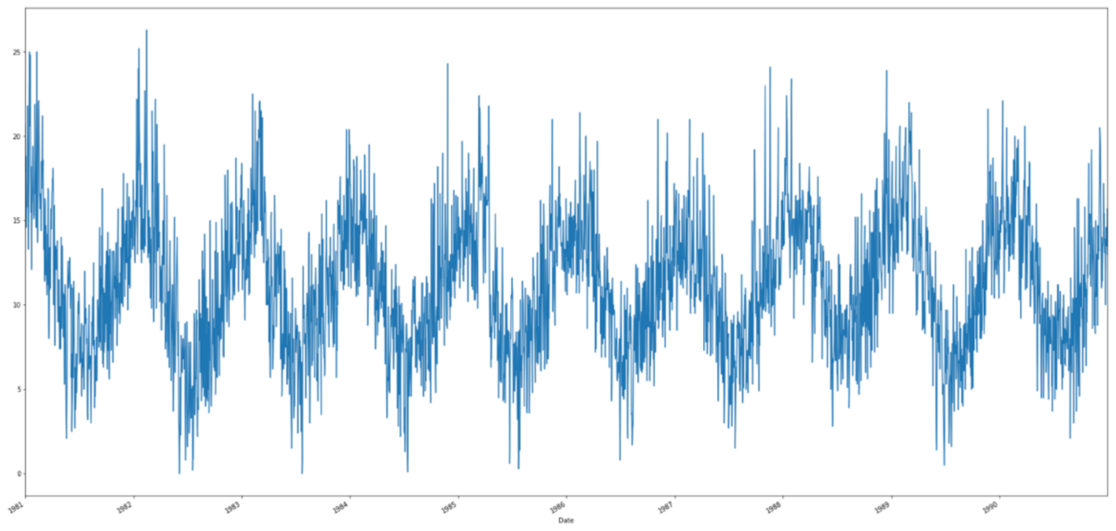
Data exploration

Dataset consist of 3650 observation with two features as Date and Temp. Observations are recorded daily with minimum temperature for 10 years from 1981-1990.

There were no missing values in the data. Data distribution is as follow:

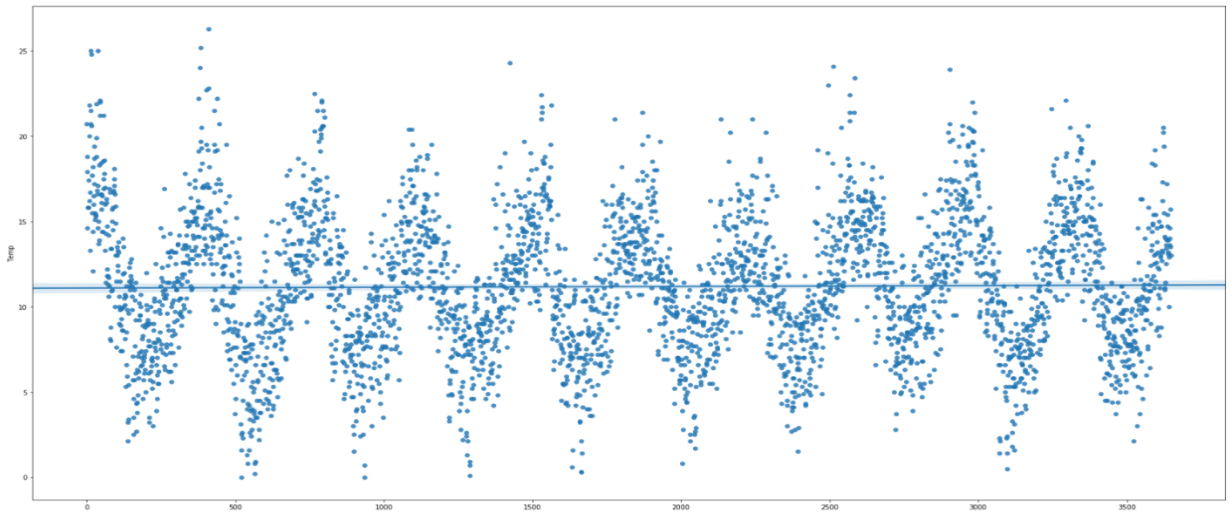
```
plt.figure(figsize=(30,15))
d_ind["Temp"].plot()
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x13d0b5d50>
```



```
plt.figure(figsize=(30,15))
sns.regplot(x= df.index.values, y=df['Temp'])
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x10f856e90>
```

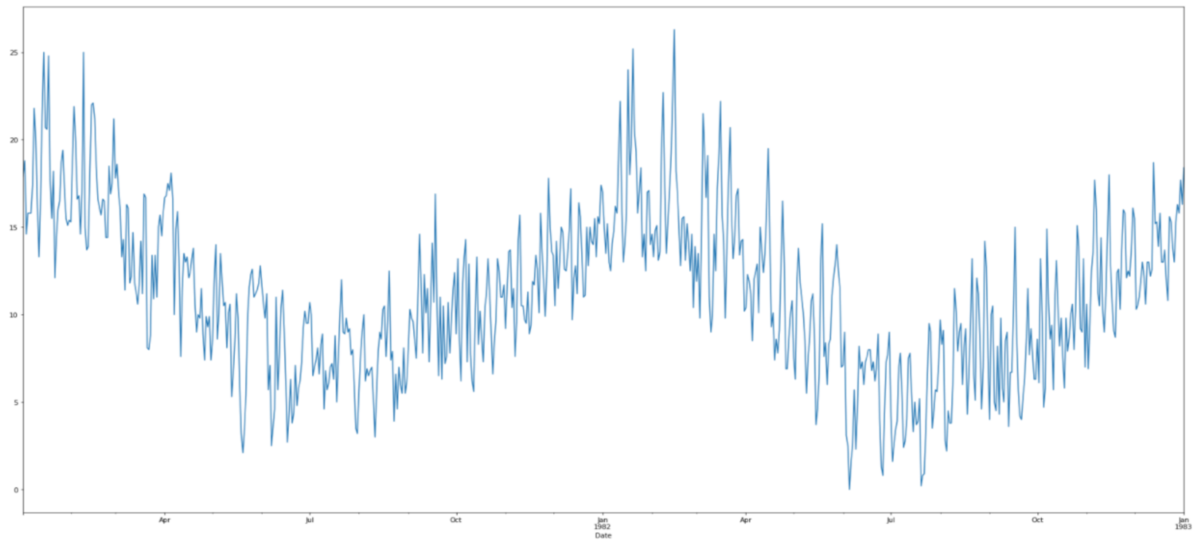


As its clear in the line plot above, dataset have yearly seasonality and linear, slightly increasing trend line.

Zoomed line plot of original dataset for two years to understand seasonality and trend better:

```
plt.figure(figsize=(30,15))
d81to83["Temp"].plot()
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x10ebac590>
```



From the above line plot, for both the years, the lowest value of minimum temperature recorded in month of June, July and highest value of minimum temperature recorded in May, in summer season.

Time series can be decomposed in three subparts.

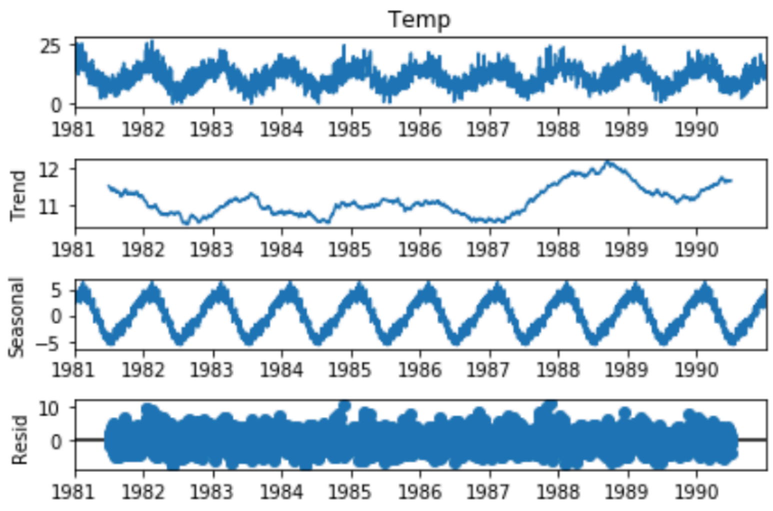
Series Decomposition: Trend+ Seasonal+ Noise. (Additive model: Linear Trend)

The time series used in this project shows linear trend, therefore it is additive.

Series Decomposition: Trend* Seasonal* Noise (Multiplicative Model: Quadratic Trend)

```
result_a = seasonal_decompose(d_ind['Temp'], model='additive', freq=365)
plt.figure(figsize=(30,15))
result_a.plot()
```

```
/Users/pujasonawane/Library/Python/3.7/lib/python/site-packages/ipykernel
reWarning: the 'freq' keyword is deprecated, use 'period' instead
  """Entry point for launching an IPython kernel.
```



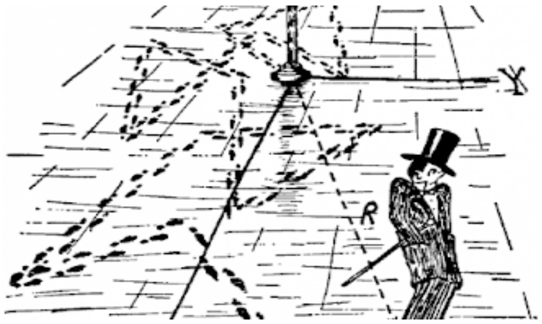
Data Modeling

Persistence Model

Persistence model uses very naive approach for forecasting. It takes the value at time t as predicted value for t+1 future time. However, Persistence model's performance can be used as benchmark for evaluating the other models. If the any other advance model does not perform better than the Persistence model, then time series is random walk and forecasting can't be done using that time series.

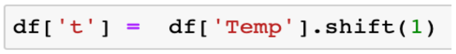
Random Walk: Random walk is condition where there is no pattern in the values recorded. Instead, next value is just random number.

Random Walk Image for reference:



In that case the best forecast can be something similar to previous value. Therefore, Persistence can be used as benchmark.

Model Implementation:



```
df['t'] =  df['Temp'].shift(1)
```

Model Evaluation:

Mean absolute error: 2.025

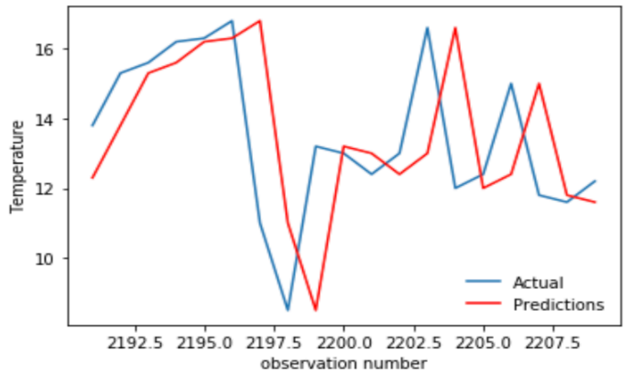
```
mae = mean_absolute_error(test_y, predictions)
mae
```

```
2.0254794520547943
```

Graphical Presentation of actual and predicted values of temperature:

```
pyplot.plot(test_y.iloc[1:20], label="Actual")
pyplot.plot(predictions.iloc[1:20], color="red",label="Predictions")
pyplot.legend(loc='lower right', frameon=False)
pyplot.xlabel('observation number')
pyplot.ylabel('Temperature')
```

```
Text(0, 0.5, 'Temperature')
```



Autoregression Model

Model Implementation and Model Evaluation:

Autoregression model is similar to the regression model with similar equation as:

$$\hat{Y} \quad = B_0 + B_1 \times X_1$$

In this model, several X variables are the various lag values of the Temp variable.

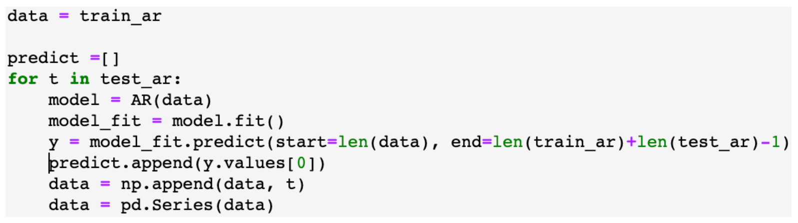
Here, temp, lag1 and lag12 are shown.

Dataset:

	Date	Temp	t	lag1Temp	lag12Temp
Date					
1982-01-02	1982-01-02	15.0	17.0	-2.0	0.8
1982-01-03	1982-01-03	13.5	15.0	-1.5	-2.4
1982-01-04	1982-01-04	15.2	13.5	1.7	5.9
1982-01-05	1982-01-05	13.0	15.2	-2.2	-3.4
1982-01-06	1982-01-06	12.5	13.0	-0.5	-0.5

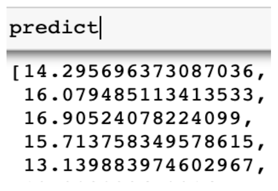
For evaluating the model, dataset is divided into train and validation dataset. Training dataset is used for model training and validation is used for model performance evaluation.

For model evaluation walk forward validation is used.



```
data = train_ar

predict =[]
for t in test_ar:
    model = AR(data)
    model_fit = model.fit()
    y = model_fit.predict(start=len(data), end=len(train_ar)+len(test_ar)-1)
    predict.append(y.values[0])
    data = np.append(data, t)
    data = pd.Series(data)
```



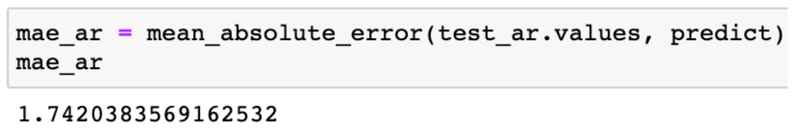
```
predict
```

```
[14.295696373087036,
 16.079485113413533,
 16.90524078224099,
 15.713758349578615,
 13.139883974602967,
```

Note: Partial output is shown to keep the document short.

This model performs very well.

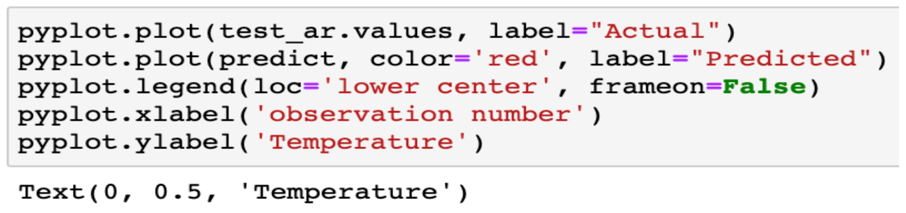
Mean Absolute Error: 1.74



```
mae_ar = mean_absolute_error(test_ar.values, predict)
mae_ar
```

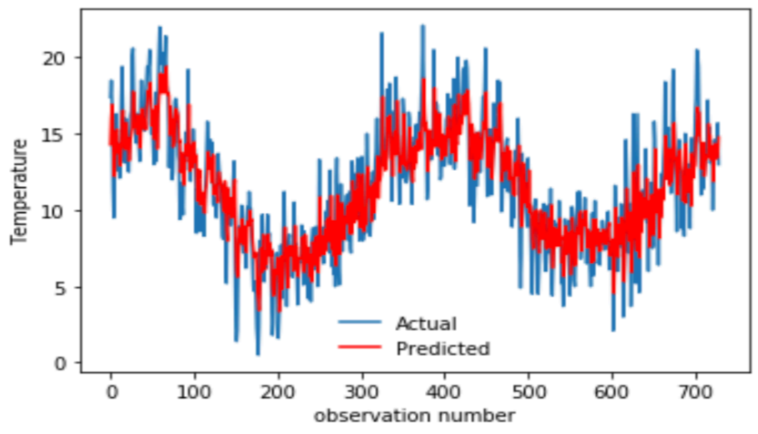
```
1.7420383569162532
```

Graphical representation of actual and predicted values:



```
pyplot.plot(test_ar.values, label="Actual")
pyplot.plot(predict, color='red', label="Predicted")
pyplot.legend(loc='lower center', frameon=False)
pyplot.xlabel('observation number')
pyplot.ylabel('Temperature')
```

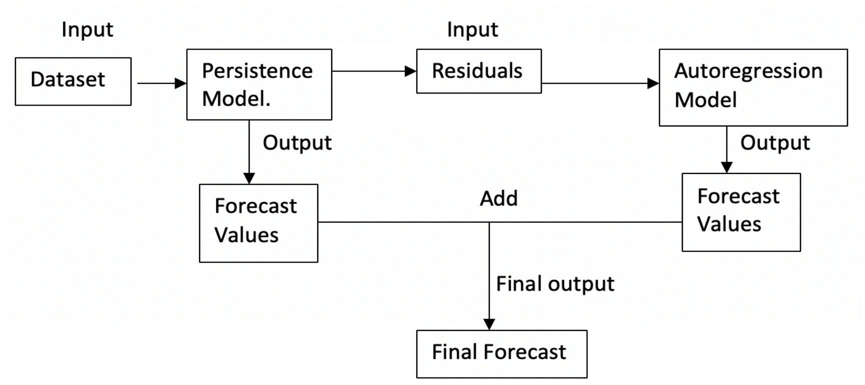
```
Text(0, 0.5, 'Temperature')
```



Moving Averages Model

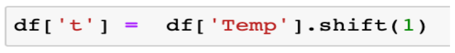
Moving Averages model is combination of two forecasting. First forecasting model can be Autoregression Model or Persistence Model and second forecasting is the forecasting on the residuals of first model.

Following is graphical representation of Moving averages Model:



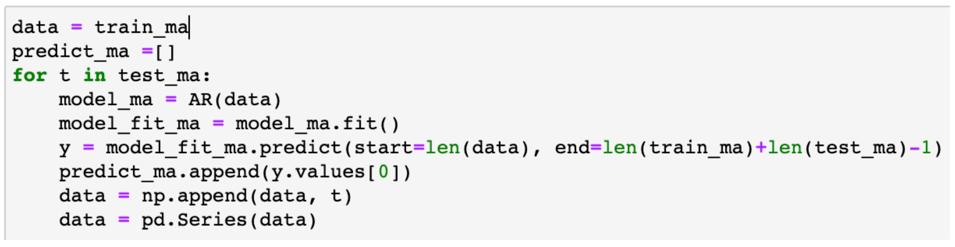
Model Implementation:

Persistence Model:



```
df['t'] =  df['Temp'].shift(1)
```

Autoregression Model:



```
data = train_ma
predict_ma =[]
for t in test_ma:
    model_ma = AR(data)
    model_fit_ma = model_ma.fit()
    y = model_fit_ma.predict(start=len(data), end=len(train_ma)+len(test_ma)-1)
    predict_ma.append(y.values[0])
    data = np.append(data, t)
    data = pd.Series(data)
```

Final predicted output:

```
final_pred= df_ma.t[df_ma.shape[0]-730:]+predict_ma
```

```
final_pred
```

```
2920    14.558443
2921    14.514161
2922    16.296274
2923    17.148942
2924    15.963437
```

Note: Partial output is shown to keep document short.

Model Evaluation:

Mean Absolute Error= 1.74

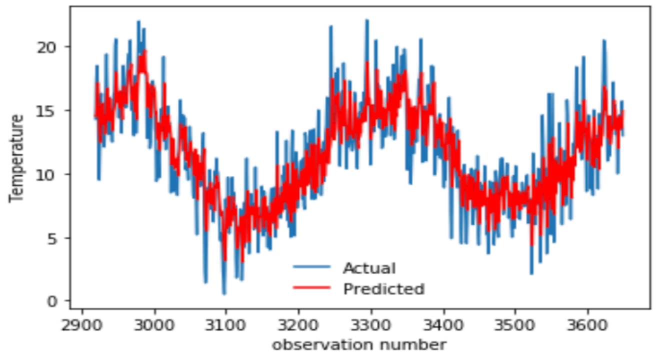
```
mae_ma = mean_absolute_error(test_yma, final_pred)
mae_ma
```

```
1.745427925110435
```

Graphical representation of actual and predicted values:

```
pyplot.plot(test_yma, label="Actual")
pyplot.plot(final_pred, color='red', label="Predicted")
pyplot.legend(loc='lower center', frameon=False)
pyplot.xlabel('observation number')
pyplot.ylabel('Temperature')
```

```
Text(0, 0.5, 'Temperature')
```



ARIMA (Autoregression Integrated Moving Averages)

ARIMA implementation in python takes three parameters as follow:

p: order of auto regression.

d: Number of differencing to deal with trend.

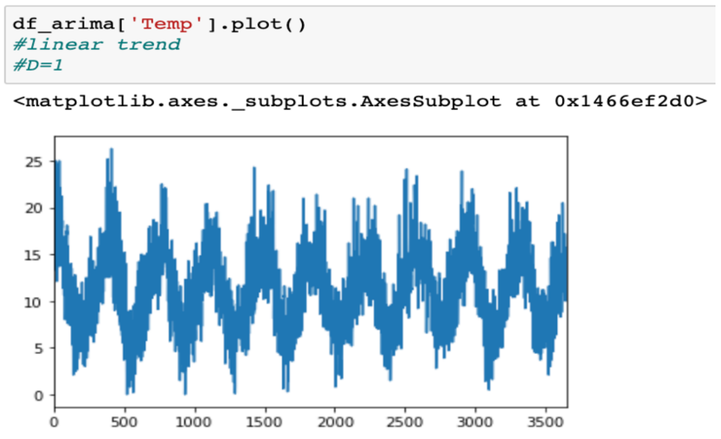
q: Window size for moving average.

Model Implementation:

Deciding value of “d”:

Line graph shows linear trend; therefore, value of d will be 1.

d=1.



```
df_arima['Temp'].plot()
#linear trend
#D=1
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x1466ef2d0>
```

Autocorrelation plot:

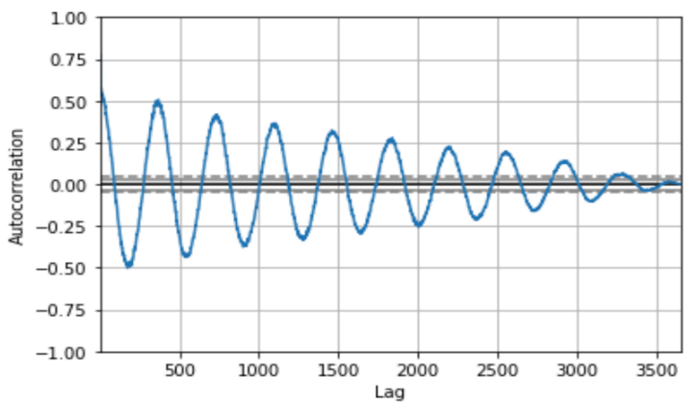
Autocorrelation is the correlation between the recorded values of variable and the lag values of the variable.

Autocorrelation plot shows the degree of correlation between variable and it’s all possible lag variables.

This lag value is equal to the value of q which window size of moving averages in ARIMA.

```
autocorrelation_plot(df_arima['Temp'])
#q=70
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x13acd6e50>
```



In this project, optimal value of q is approximately equal to 50, but creating model with 50 lag variables is computationally high cost process. Therefore, in this project "q" is considered to be 10.

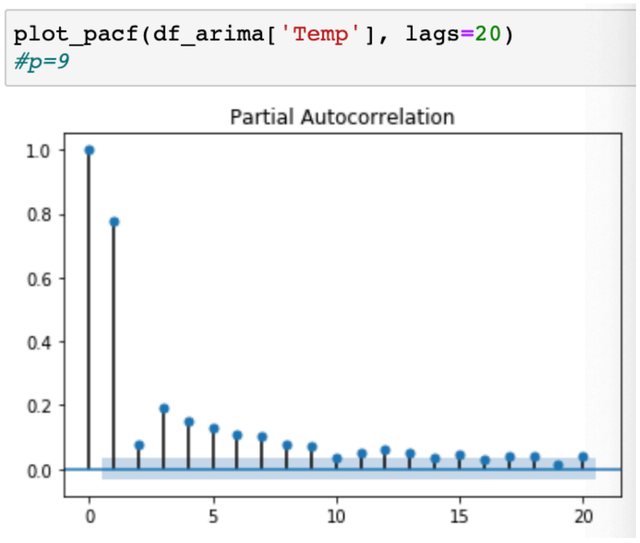
q=10.

Partial Autocorrelation plot:

Partial Autocorrelation plot helps to decide the value of "p" which is order of autoregression.

Lagged values with partial autocorrelation coefficient outside of 95% confidence interval can be the order of autocorrelation.

Partial Autocorrelation plot:


```
plot_pacf(df_arima['Temp'], lags=20)
#p=9
```

From the plot above, p= 9

Model:

```
try:
    m_arima = ARIMA(df_arima['Temp'], order=(9,1,10)).fit()
except:
     m_arima = ARIMA(df_arima['Temp'], order=(9,1,10)).fit(start_params=[1,.1,.1,.1])
```

Model Evaluation:

ARIMA model does not perform very well on this dataset because of high seasonality in data and walk forward validation is not implemented due to need of high computational power. But results can be improved.

Mean Absolute Error: 3.46

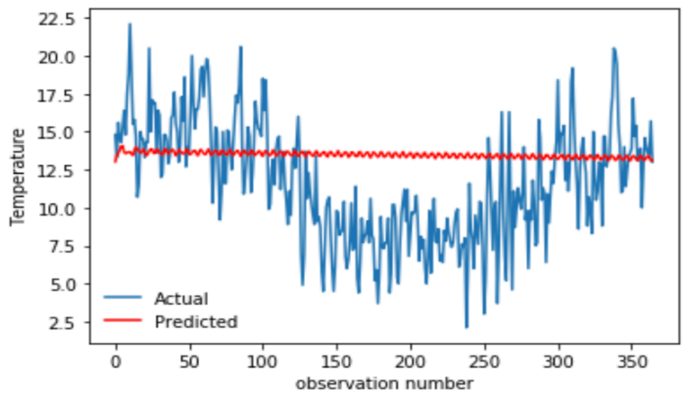
```
mae_arima = mean_absolute_error(test_arima,pred)
mae_arima
```

```
3.463122921160078
```

Graphical representation of actual and predicted data:

```
pyplot.plot(test_a["Temp"], label="Actual")
pyplot.plot(pred, color='red', label="Predicted")
pyplot.legend(loc='lower left', frameon=False)
pyplot.xlabel('observation number')
pyplot.ylabel('Temperature')
```

```
Text(0, 0.5, 'Temperature')
```



SARIMA (Seasonal Autoregression Integrated Moving Averages)

This model is very similar to ARIMA but can handle the seasonal component of time series.

Model Implementation.

SARIMA implementation in python takes extra four parameters in addition to parameters taken by ARIMA are as follow:

P= Seasonal Autoregressive Order.

D= Seasonal Differencing Order.

Q= Seasonal Moving Average Order.

m: The number of time steps for single seasonal period.

Model Implementation:

```
model = SARIMAX(train_sarima, order=(9,1,10), seasonal_order=(1,1,1,12))
```

```
model_sarima=model.fit()
```

```
forecast_s=model_sarima.forecast(len(test_sarima))
```

```
forecast_s
```

```
3285     12.848868
3286     13.150116
3287     13.428241
3288     13.449686
3289     13.403741
```

Note: Partial output is shown.

Model Validation:

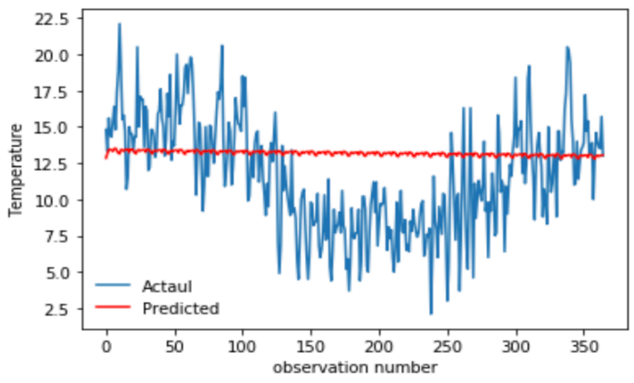
Mean Absolute Error = 3.38

This model also does not perform well as ARIMA.

Graphical representation of actual and predicted values:

```
pyplot.plot(test__["Temp"], label= "Actaul")
pyplot.plot(fore, color='red', label= "Predicted")
pyplot.legend(loc='lower left', frameon=False)
pyplot.xlabel('observation number')
pyplot.ylabel('Temperature')
```

```
Text(0, 0.5, 'Temperature')
```



Artificial Neural Network

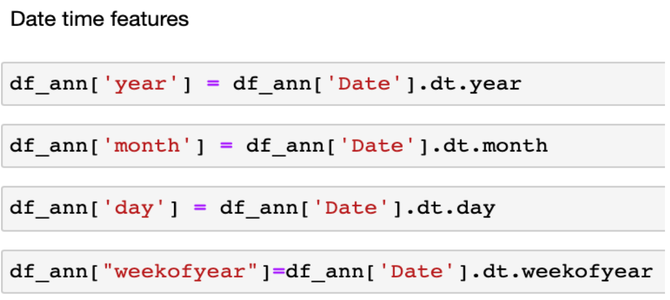
Artificial Neural Network model for time series works same as the ANN model for regression problem. In this project, for ANN model feature engineering is done to get better results.

Feature Engineering:

In time series three types of features can be created which are as follow:

a. Date time features- These are feature related to date and time part of the time series data. Ex. Some of these are day, week, month, year and so on when observation is recorded.

   In this project, four date time features are created from Date column.



Date time features

```
df_ann['year'] = df_ann['Date'].dt.year
```

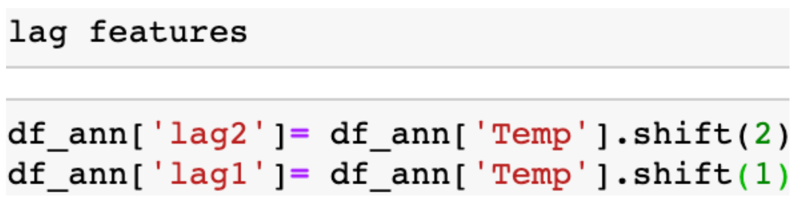
```
df_ann['month'] = df_ann['Date'].dt.month
```

```
df_ann['day'] = df_ann['Date'].dt.day
```

```
df_ann["weekofyear"]=df_ann['Date'].dt.weekofyear
```

b. Lag features:

   Lag features are the values of recorded variable at prior time steps. In this analysis, two lag features are created lag1 and lag2 as future temperature value may depend on the temperature of two previous days.



```
lag features
```

```
df_ann['lag2']= df_ann['Temp'].shift(2)
df_ann['lag1']= df_ann['Temp'].shift(1)
```

c. Window features:

   Window features are summary of values recorder over fix window in prior time.

There are two types of window features:

Rolling Window: In this, window size does not change. It takes the specified summary of values over fixed window size.

In this project, two rolling window features are as follow:

```
df_ann['meanofweek'] = df_ann['Temp'].rolling(window = 7).mean()
```

```
df_ann['minofweek'] = df_ann['Temp'].rolling(window = 7).min()
```

Expanding Window: Window size is not fixed; it includes all previous data till that observation.

Following is feature which records the minimum temperature till date,

```
df_ann['mintilldate'] = df_ann['Temp'].expanding().min()
```

Final dataset with all created features:

Temp	year	month	day	weekofyear	lag2	lag1	meanofweek	minofweek	mintilldate
15.8	1981	1	7	2	15.8	15.8	17.057143	14.6	14.6
17.4	1981	1	8	2	15.8	15.8	16.585714	14.6	14.6
21.8	1981	1	9	2	15.8	17.4	17.142857	14.6	14.6
20.0	1981	1	10	2	17.4	21.8	17.314286	14.6	14.6
16.2	1981	1	11	2	21.8	20.0	17.542857	15.8	14.6

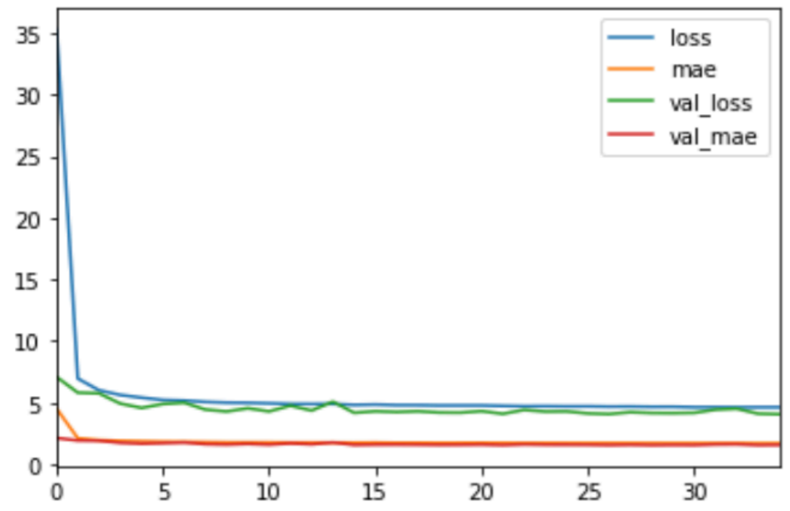
Model Implementation:

ANN model implemented is sequential model. Model architecture is as follow:

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense (Dense)                (None, 30)                300
_________________________________________________________________
dense_1 (Dense)              (None, 30)                930
_________________________________________________________________
dense_2 (Dense)              (None, 1)                 31
=================================================================
Total params: 1,261
Trainable params: 1,261
Non-trainable params: 0
```

Adam optimizer is used and evaluation matric used is mean absolute error.

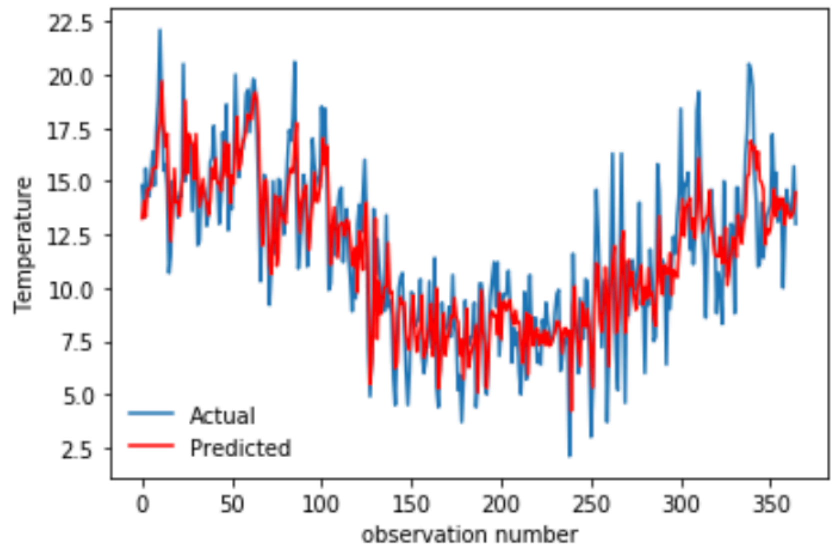
Graphical representation of loss and MAE values:



Mean Absolute Error: validation MAE= 1.56

This model performs well and can be selected as final model for forecasting task.

Graphical representation of actual and predicted values:



Conclusion

All the models studied; their performances are mention above in detail.

In conclusion of this analysis, Artificial Neural Network model can be best model for forecasting among all models. Additionally, Autoregression model performed well. It can be considered as potential model after ANN model for forecasting of minimum temperature.

References

Google.com. 2020. Image: Python Intro #02 | Function – Random Walk – Computation Unit. [online] Available at: <https://www.google.com/imgres?imgurl=http%3A%2F%2Fcu.t-ads.org%2Fwp-content%2Fuploads%2F2015%2F01%2Fdrunkard-650x276.png&imgrefurl=http%3A%2F%2Fcu.t-ads.org%2Fpython-intro-02-random-walk%2F&tbnid=DmH_PFoHz3VlLM&vet=12ahUKEwjU8Yr0o7XqAhUCMlMKHbfeCHcQMyg9egQIARBt..i&docid=R5TLpN21HetNbM&w=650&h=276&q=random%20walk%20images&ved=2ahUKEwjU8Yr0o7XqAhUCMlMKHbfeCHcQMyg9egQIARBt> [Accessed 5 July 2020].

Medium. 2020. Yet Another An End-To-End Time Series Project Tutorial. [online] Available at: <https://towardsdatascience.com/yet-another-an-end-to-end-time-series-project-tutorial-37390bcea38e> [Accessed 5 July 2020].