The University of Hong Kong

Department of Statistics and Actuarial Sciences

STAT8003 Time Series Forecasting - Group Project

Daily Temperature Forecast Using ARIMA & Deep Learning Models

Name	UID

Table of Contents

Abstract

While temperature forecasting is a sophisticated study that requires considerable domain expertise, the strong seasonality nature of daily temperature data allows for reasonable prediction using statistical and deep learning time series models that leverage the historical pattern alone. This study aims to develop and compare univariate forecasting approaches, including traditional ARIMA model as well as deep learning models like transformer model to generate both short-term and extended forecasts of daily temperature value. While LSTM and N-HiTS models are used for long-term temperature forecasts.

Motivation

The changing climate of the Earth is affecting extreme weather patterns worldwide. One of the most significant impacts of climate change is the rise in global temperatures, which has led to an increase in extreme weather events such as heat waves and cold snaps. Accurate temperature predictions are essential in helping individuals and organizations prepare for these events and minimize their impact. For example, accurate temperature predictions can help people plan their daily activities, such as deciding what to wear or when to exercise. This study aims to develop and compare univariate forecasting approaches, including traditional ARIMA model as well as deep learning models like transformer model to generate both short-term and extended forecasts of daily temperature value.

Data & Preliminary Analysis

This study utilizes daily maximum temperature observations recorded by the Hong Kong Observatory[1]. The full dataset spans from 1 Jan 1884 to 30 Nov 2023.

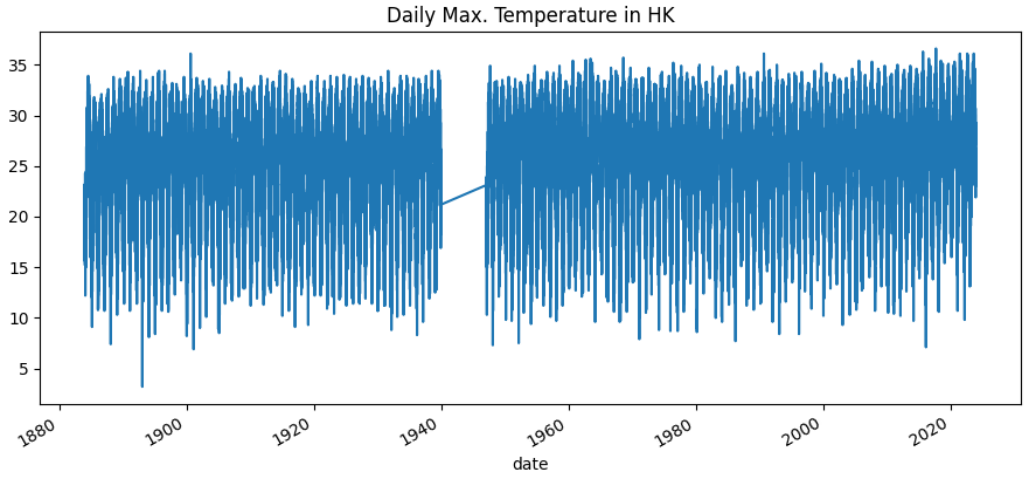


Figure 1. Daily Max Temperature Recorded By Hong Kong Observatory

However, as the primary focus is on developing models for forecasting more recent temperatures, only data from January 1st, 1960 onwards is utilized. This reduces the time period covered while avoiding gaps in observations from 1940 to 1946 when incomplete records were collected. By restricting the analysis to data from 1960 onward, a robust and consistent dataset is available for both training predictive models and evaluating their performance on out-of-sample future temperatures. The statistics of data from 1960 onward is as below.

Count	23345
Mean	25.92
Std	5.38
Min	7.1
Max	36.6

[1] Data from: https://data.gov.hk/en-data/dataset/hk-hko-rss-daily-temperature-info-hko

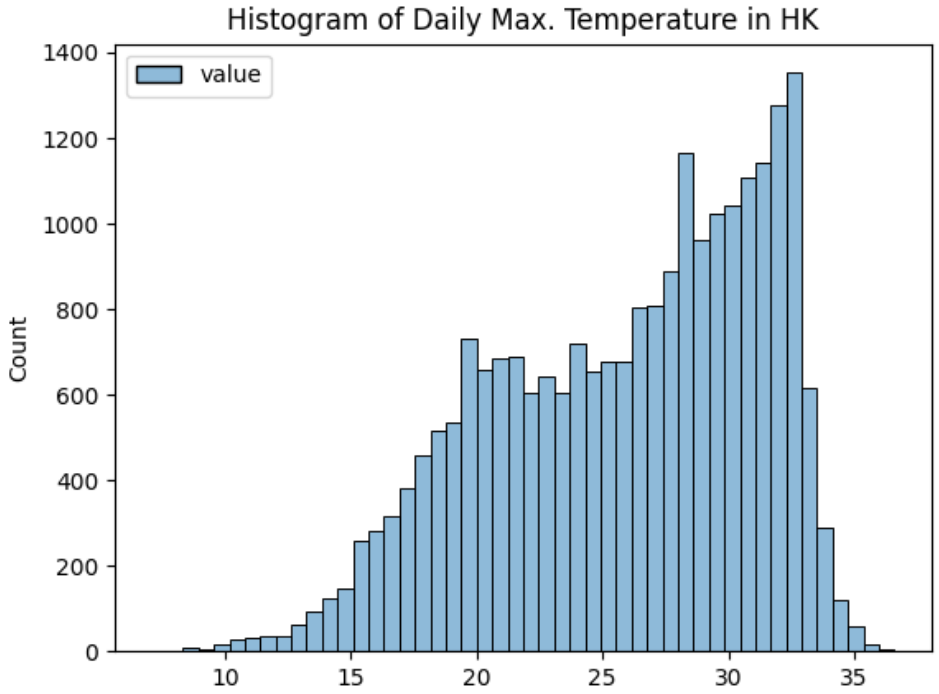


Figure 2. Distribution of the Temperature Data Set

Methodology

Models

Different training and testing set period is defined as below:

Model	**Training**	**Testing**
ARIMA	Dec 01, 2020 to Nov 25, 2023	Nov 26, 2022 to Nov 30, 2023
Transformer	Jan 01, 1960 to Nov 30, 2020	Dec 1, 2020 to Nov 30, 2023
N-HITS	Use Cross-validation for data dated from Jan 01, 1960 to Nov 30, 2023. (# 23,345 observations), where the ratio for training, validation and testing to be 8:1:1.	
LSTM		

Model Evaluation

MAE, MSE, RMSE and AIC will be used to evaluate the forecasting performance of the fitted model, which lower the value, better the forecasting performance.

Analysis & Model Fitting

Approach 1: ARIMA Model

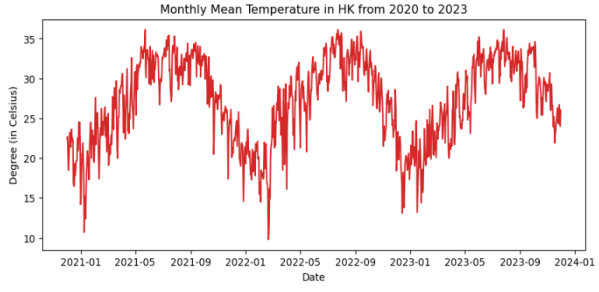
The ARIMA model is a generalization of the Autoregressive Moving Average (ARMA) model and is used to gain better insights into the data and predict future trends.

The ARIMA model has three parameters: p, d, and q. The p parameter represents the number of autoregressive terms, the d parameter represents the number of nonseasonal differences, and the q parameter represents the number of moving average terms. The optimal values of these parameters can be determined by checking the ACF/PACF plot or using the grid search technique.

For this model, three years of data will be used for model development, and the last five values will be masked and used for validation.

Data Transformation

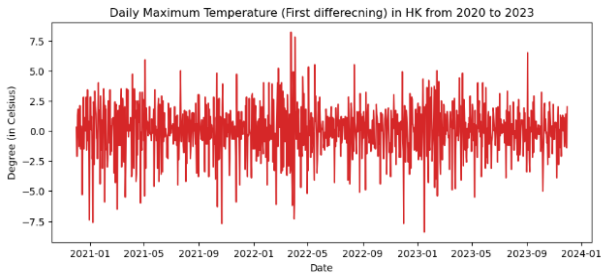
Before fitting the ARIMA model, the stationarity assumptions need to be met. From Figure 3, it is evident that the data has a strong seasonal pattern and is not stationary. Additionally, the result of the Augmented Dickey-Fuller test also supports our assumption that the time series is not stationary (i.e., the null hypothesis is retained at the 5% significance level).




```
Orignal Time Series (Daily Max. Temp from 2020 - 2023)
Results of Dickey-Fuller Test
================================================
Test Statistic                  -2.480418
p-value                          0.120312
#Lags Used                      11.000000
Number of Observations Used   1083.000000
Criterical Value (1%)           -3.436403
Criterical Value (5%)           -2.864212
Criterical Value (10%)          -2.568193
dtype: float64
================================================
```

Figure 3. Time plot and Augmented Dickey-Fuller test of the daily maximum temperature in HK from 2020 to 2023

Since a strong seasonal pattern is observed, it is suggested to perform first differencing to neutralize the seasonal component. From the plot below (Figure 4) and the result of the Augmented Dickey-Fuller test (p-value < 0.05), it is suggested that the time series is stationary after first-order differencing.




```
After first order differencing (Daily Max. Temp from 2020 - 2023)
Results of Dickey-Fuller Test
================================================
Test Statistic                -1.225097e+01
p-value                        9.542726e-23
#Lags Used                     1.400000e+01
Number of Observations Used    1.079000e+03
Criterical Value (1%)         -3.436425e+00
Criterical Value (5%)         -2.864222e+00
Criterical Value (10%)        -2.568198e+00
dtype: float64
================================================
```

Figure 4. Time plot and Augmented Dickey-Fuller test of the daily maximum temperature in HK from 2020 to 2023 after first order differencing

Model specification

For the model specification, we can identify it based on the sample ACF and PACF patterns shown below:

	AR	MA	ARMA
ACF	Geometric decay	Cutoff at q significant lags	Geometric decay
PACF	Cutoff at p significant lags	Geometric decay	Geometric decay

Based on the sample ACF and PACF plot (Figure 5) and grid search based on AIC (Figure 6), three potential models are identified:

Model 1: As the sample PACF decays geometrically and sample ACF shows significant spikes at lag 1 to lag 3 which suggests the MA(3) model. Therefore, the potential model will be ARIMA(0,1,3).

Model 2: As the ACF decays geometrically-alike after lag 2, it suggests MA(2). And the PACF decays geometrically after lag 2, which suggests the AR(2) model. Therefore, the potential model will be ARIMA(2,1,2).

Model 3: By using the Grid Search method, ARIMA(1,1,2) has the lowest AIC. It suggests that ARIMA(1,1,2) has a better fit.

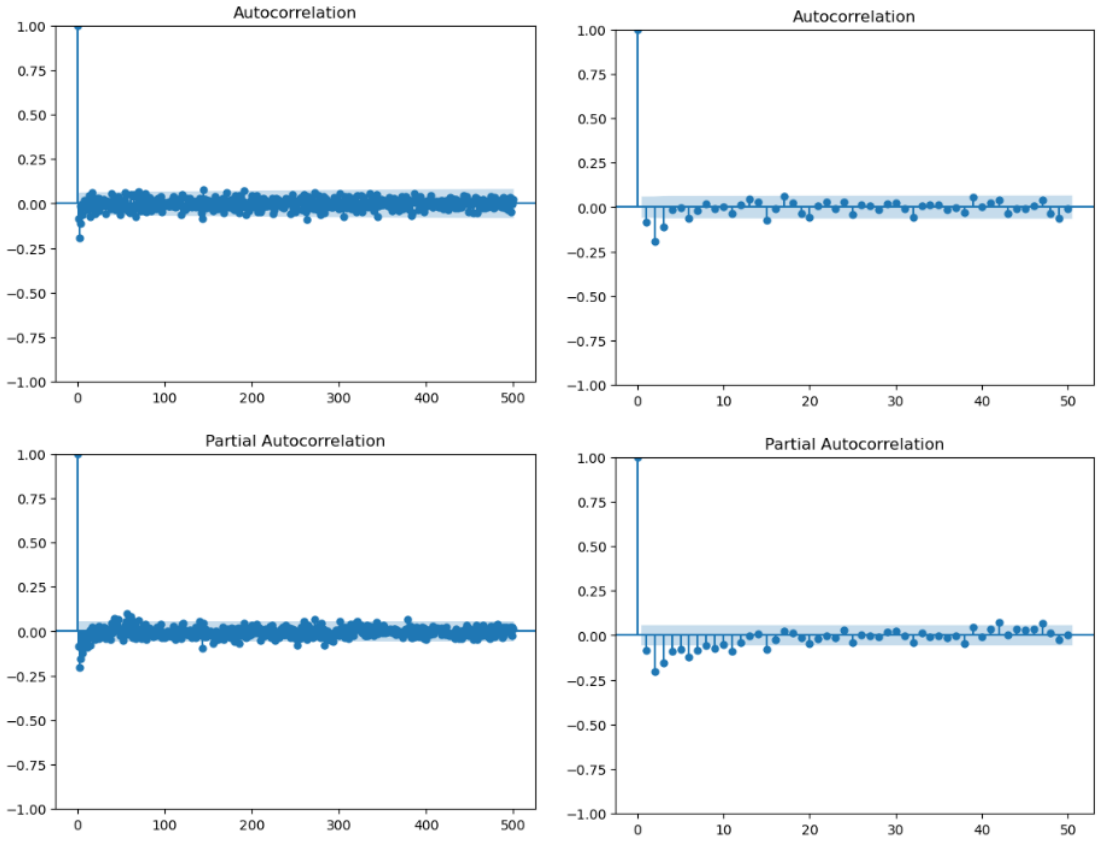


Figure 5: Sample PACF and ACF of first order differencing time series

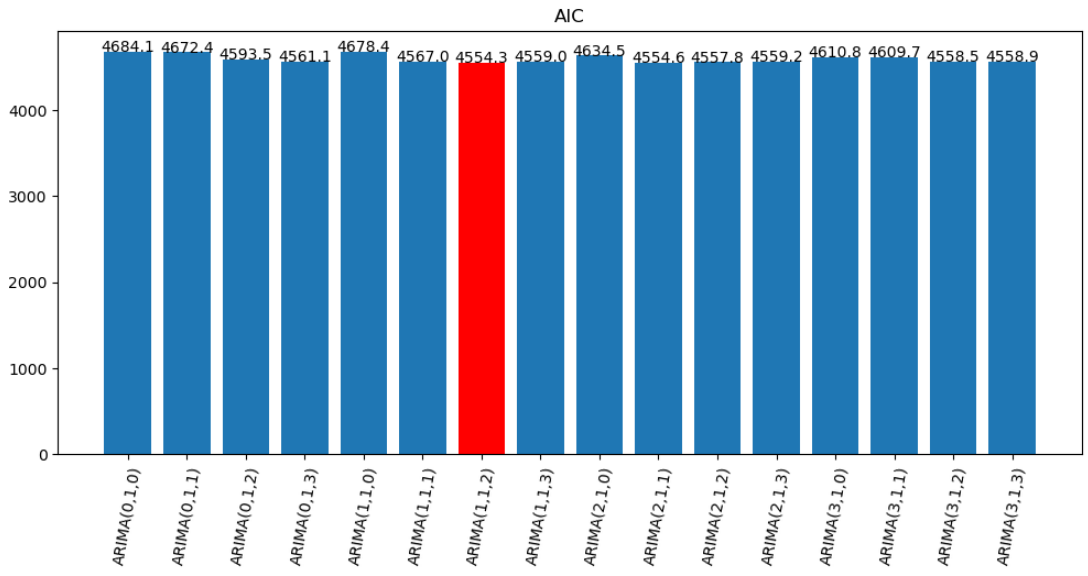


Figure 6. Grid Search result based on AIC values

Model Estimation and Diagnostic Checking

1. Parameter Estimation

By fitting this model using the maximum likelihood method, we get the following estimated coefficients below:

Model 1 - ARIMA(0,1,3)				**Model - 2 - ARIMA(2,1,2)**				**Model 3 - ARIMA(1,1,2)**			
AIC: 4561.063				AIC:4557.754				AIC:4554.330			
	Coefficient	Standard Error	p-value		Coefficient	Standard Error	p-value		Coefficient	Standard Error	p-value
MA(1)	-0.2036	0.028	0.000	AR(1)	0.2808	0.221	0.203	AR(1)	0.4877	0.061	0.000
MA(2)	-0.2884	0.030	0.000	AR(2)	0.1428	0.156	0.361	MA(1)	-0.6845	0.066	0.000
MA(3)	-0.1841	0.027	0.000	MA(1)	-0.4849	0.219	0.027	MA(2)	-0.1722	0.045	0.000
sigma2	3.829	0.132	0.000	MA(2)	-0.3547	0.203	0.081	sigma2	3.8054	0.129	0.000
				sigma2	3.8101	0.130	0.000				

For Model 1 and Model 3, all the estimated parameters are significant, and Model 3 has a lower AIC (4557.754) than Model 1 (4561.063), which indicates that Model 3 has a better fit.

For Model 2, it is similar to Model 3, which has an additional AR(2) parameter. However, the AR(2) parameter is not insignificant, and the AR(1) parameter changes significantly (from 0.4877 to 0.2808) and becomes insignificant (p-value = 0.203 >0.05). This indicates that it is over-parameterized. To confirm Model 3 is an adequate model, we have fitted another Model 4 - ARIMA (1, 1, 3). The value of AR(1) in Model 4 changes significantly (change from 0.4877 to 0.1744) and becomes insignificant (p-value =0.213 > 0.05), which is also over-parameterized. Therefore, we can confirm that Model 3 - ARIMA (1, 1, 2) is an adequate model.

Model 4 - ARIMA(1,1,3)			
AIC: 4559.020			
	Coefficient	Standard Error	p-value
AR(1)	0.1744	0.140	0.213
MA(1)	-0.3710	0.139	0.008
MA(2)	-0.2441	0.048	0.000
MA(3)	-0.1310	0.056	0.019
sigma2	3.8148	0.131	0.000

2. Test for serial autocorrelation (Ljung-Box test)

To check whether there is correlation in the error, the Ljung-Box test has been conducted at different lags for all potential models. All of the p-values in different lag are above 0.05 (i.e. 5% significance level). Therefore, we can retain the null hypothesis and conclude that there is no correlation in error.

	Residual analysis (Ljung-Box Test) p-value											
Lag	**1**	**3**	**5**	**7**	**10**	**14**	**30**	**60**	**120**	**180**	**365**	**730**
ARIMA(0,1,3)	0.675766	0.733721	0.896334	0.453652	0.69086	0.809303	0.916282	0.13272	0.733022	0.900124	0.524948	0.862781
ARIMA(2,1,2)	0.768492	0.530186	0.68508	0.706231	0.895569	0.902106	0.912999	0.193385	0.746724	0.852806	0.477933	0.893381
ARIMA(1,1,2)	0.926903	0.872313	0.896828	0.849017	0.962854	0.959057	0.951041	0.248649	0.799035	0.890917	0.530065	0.917437

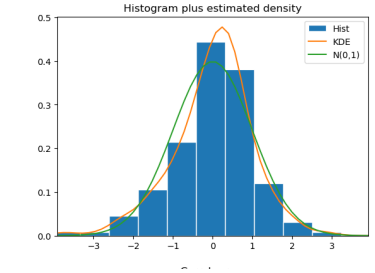
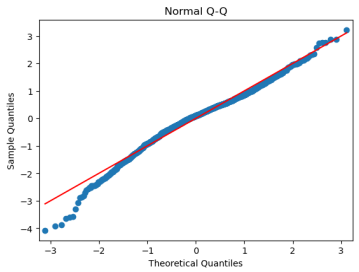
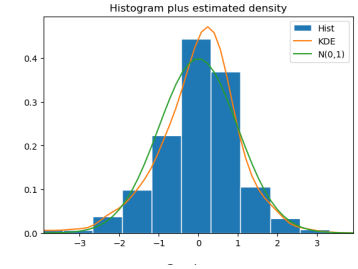
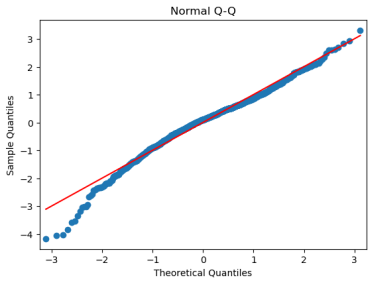
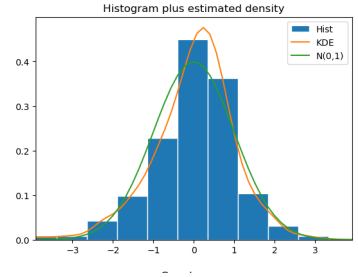
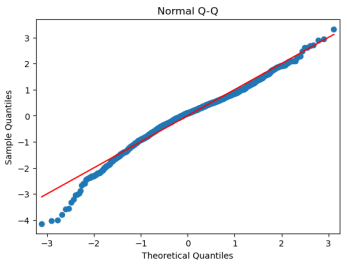
3. Test for heteroskedasticity (Goldfeld-Quandt test)

To check whether there is heteroskedasticity in standardized residuals, a two-sided Goldfeld-Quandt test has been conducted. The p-value of the Goldfeld-Quandt test in all models is above 0.05, which retains the null hypothesis at the 5% significance level. Therefore, we can conclude that there is no heteroskedasticity for all three models.

Goldfeld-Quandt test	**GQ Value**	**p-value**
Model 1 - ARIMA(0,1,3)	0.848	0.118
Model 2 - ARIMA(2,1,2)	0.862	0.159
Model 3 - ARIMA(1,1,2)	0.858	0.145

4. Test for normality of the residuals

To test for normality of the residuals, we can examine the histogram and Quantile-Quantile plot of the standardized residuals. Based on these plots, the non-normality is confirmed for all three models. The non-normality might be due to outlier data in lower temperature. This means that the ARIMA model is not able to capture the extreme temperature scenario, especially in lower temperature values.

Model 1 - ARIMA (0, 1, 3)	Model 2 - ARIMA (2, 1, 2)	Model 3 - ARIMA (1, 1, 2)
 	 	 

5. Model Selection

Based on the result mentioned above, the proposed model will be Model 1 and Model 2 in view of similar residual patterns (i.e. autocorrelation, heteroskedasticity and normality) and the parameter estimation (i.e. all are significant). By comparing the AIC of Model 1 and Model 3, we can see that Model 3 has a lower AIC. Therefore, we will use Model 3 for forecasting.

Models Performance

Forecast

Based on the final proposed model, which is ARIMA(1, 1, 2), below are the 5-period ahead predicted results compared with the true data as validation (Figure 7). The predicted value does not deviate a lot from the true values, and all the true values are within the 95% confidence interval. However, even though the forecast result of ARIMA(1, 1, 2) is acceptable, it is important to note that the normality assumption does not hold, and is not able to capture the extreme temperature scenario, especially in lower temperature. Given the non-normality of the residuals, we have tried to use deep learning models (e.g. Transformer Model, N-HiTS and LSTM) to predict the temperature.

ARIMA(1, 1, 2)	**Degrees Celsius**				
Date	**Test - True Value**	**Predicted Value**	**Standard Error**	**95% Lower Bound**	**95% Upper Bound**
26/11/2023	25.30	24.5873	1.9507	20.7639	28.4107
27/11/2023	26.70	24.8413	2.5021	19.9372	29.7454
28/11/2023	25.40	24.9652	2.7112	19.6514	30.2790
29/11/2023	24.00	25.0256	2.8236	19.4915	30.5597
30/11/2023	26.00	25.0551	2.9007	19.3698	30.7403

Model	**Mean Absolute Error (MAE)**	**Mean Square Error (MSE)**	**Root Mean Square Error (RMSE)**
ARIMA(1, 1, 2)	0.99534	1.219289878	1.104214598

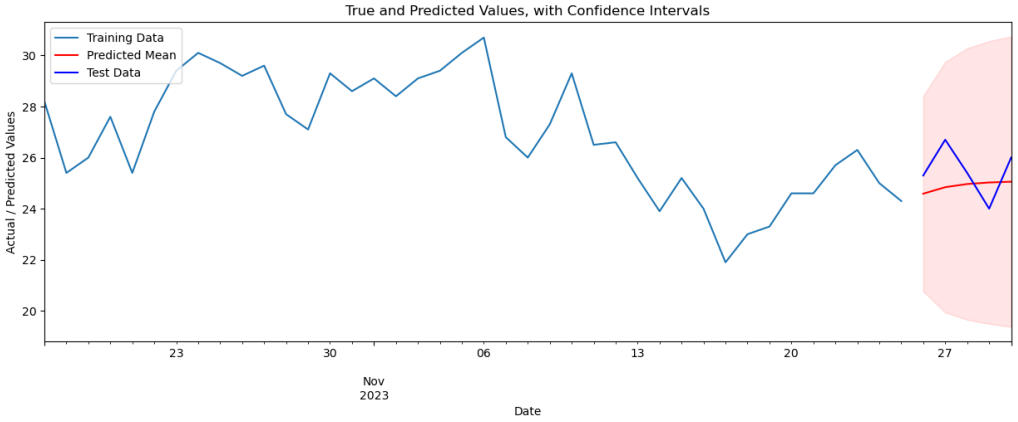


Figure 7: 5-period ahead predicted results based on ARIMA(1, 1, 2)

Approach 2: Transformer Model

Transformer Architecture

The Transformer Model is one of the most popular deep learning models in recent years, especially for natural language processing (NLP) tasks that involve sequential data. Transformer is very efficient in processing and predicting sequential data (e.g. NLP, time series data) using the multi-head attention mechanisms, this allows the transformer to identify and capture both long term patterns, as well as short term relationships between each data point. Given that time series data, like temperature recordings, can be considered sequential in nature, the Transformer is a suitable candidate for this forecasting problem.

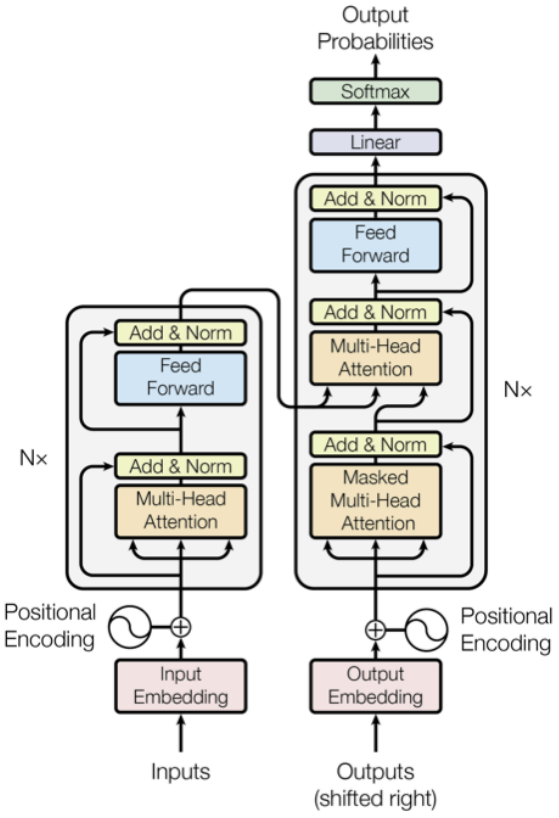


Figure 8. The architecture of Transformer (Vaswani, 2017)

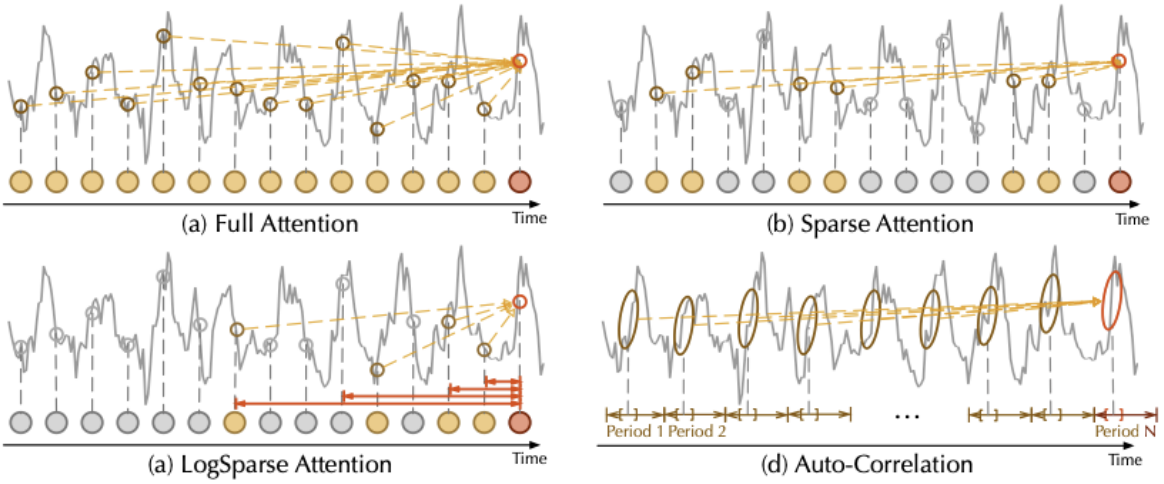


Figure 9. Attention mechanism in Transformer (Wu, 2021)

Implementation

The transformer model is implemented using Pytorch. Before building and training the model, temperature data will first scale to the range between 1 and 0 using MinMaxScaler(), this can avoid the bias and potentially reduce the training time by allowing the optimizer to learn more efficiently, which leads to faster convergence.

A rolling forecasting approach is used where the model performs 1-step ahead predictions. Specifically, the previous 365 days of daily temperature values are provided as inputs to forecast the next day's temperature. Which mean $[Temp_{t-365}, Temp_{t-364}, \dots, Temp_{t-1}]$ will be passed to the model as input to predict $Temp_t$. This rolling forecast process is applied during both model training and evaluation to simulate real-world 1-day-ahead predictions using only historical point in time data. By employing a consistent 1-year lookback window that rolls through the full datasets, this methodology allows assessing the model's daily forecasting performance over many time periods.

The structure of the transformer model is as below.

```
TransformerModel(
  (encoder): Linear(in_features=1, out_features=64, bias=True)
  (pos_encoder): PositionalEncoding(
    (dropout): Dropout(p=0.2, inplace=False)
  )
  (transformer_encoder): TransformerEncoder(
    (layers): ModuleList(
      (0-1): 2 x TransformerEncoderLayer(
        (self_attn): MultiheadAttention(
          (out_proj): NonDynamicallyQuantizableLinear(in_features=64, out_features=64, bias=True)
        )
        (linear1): Linear(in_features=64, out_features=2048, bias=True)
        (dropout): Dropout(p=0.1, inplace=False)
        (linear2): Linear(in_features=2048, out_features=64, bias=True)
        (norm1): LayerNorm((64,), eps=1e-05, elementwise_affine=True)
        (norm2): LayerNorm((64,), eps=1e-05, elementwise_affine=True)
        (dropout1): Dropout(p=0.1, inplace=False)
        (dropout2): Dropout(p=0.1, inplace=False)
      )
    )
  )
  (decoder): Linear(in_features=64, out_features=1, bias=True)
)
```

Models Performance

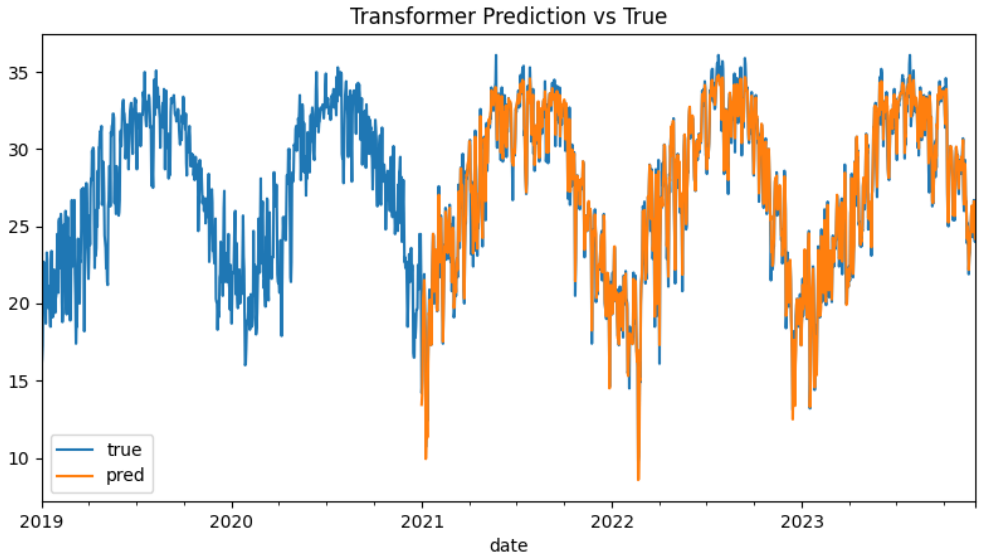


Figure 10. Predicted Temperature vs True Temperature

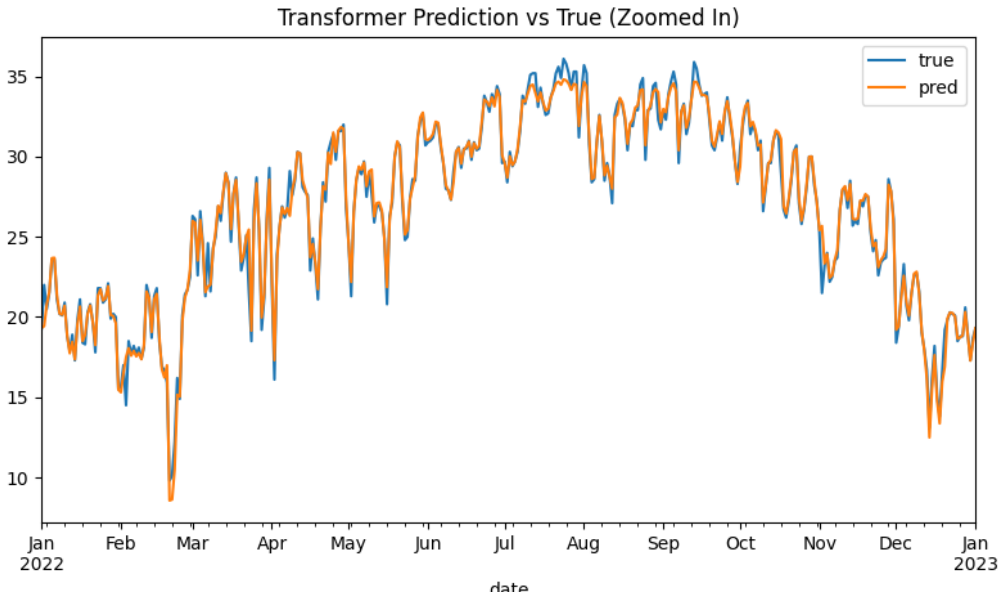


Figure 11. Predicted Temperature vs True Temperature (Zoomed In)

The result indicated that transformer models effectively captured the seasonal trend and short term pattern present in the temperature data. Most predictions were found very close to the true observed value. Also, in comparison to the traditional approach, transformers did not show a lagging effect that are commonly seen in ARIMA models. This also shows that the multi-headed self attention mechanism appears well-adapt to distinctly modeling relationships between different past observations and the next time step across different seasonal and lag timescales simultaneously. Overall, the transformer model provides a promising prediction by properly capturing the seasonal and sequential dependencies of the temperature data.

Mean Absolute Error (MAE)	0.3452
Mean Squared Error (MSE)	0.3670
Root Mean Squared Error (RMSE)	0.6058

Approach 3: Long Term Prediction using Deep Learning Methods

In this project, we would like to extend the prediction to a long-term period using deep learning methods. Particularly, we have selected the N-HiTS and LSTM as the suitable candidates for this task. Since deep learning methods may require much more data compared to the traditional statistical methods. The dataset has been extended from Jan 01, 1960 to Nov 30, 2023. (# 23,345 observations), where the ratio for training, validation and testing to be 8:1:1.

N-HiTS Architecture

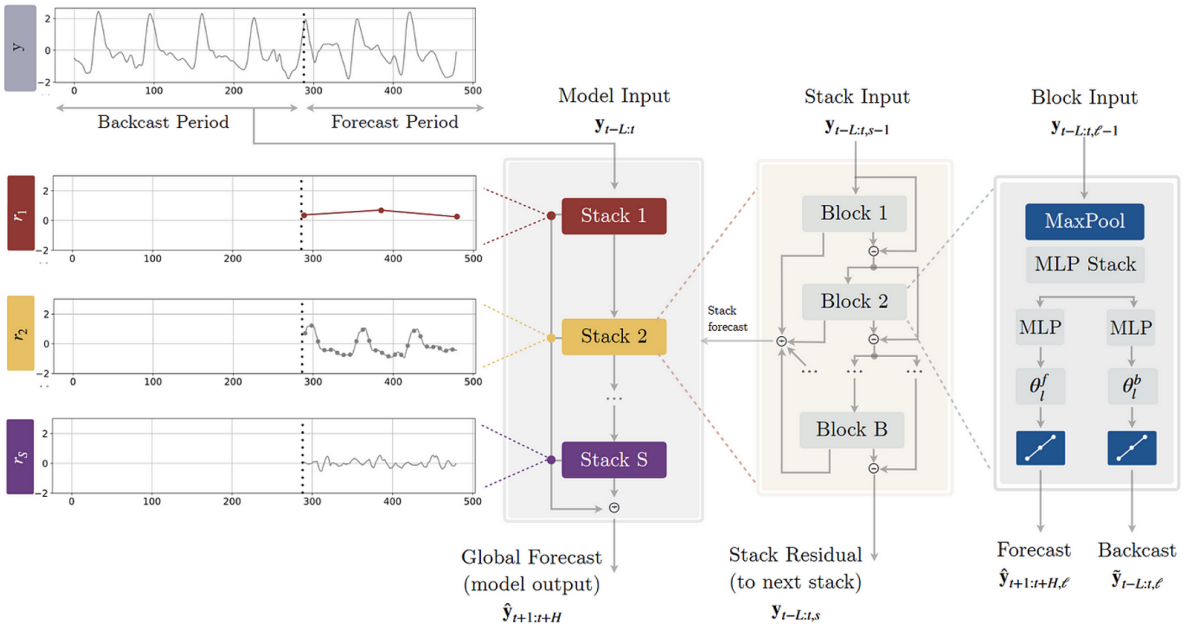


Figure 12. N-HiTS Architecture

N-HiTS (Neural Hierarchical Time Series) is the enhanced version of N-BEATS (Neural Basis Expansion Analysis for Interpretable Time Series Forecasting) with lower computation cost and better accuracy. From its architecture diagram, N-HiTS model is composed of Stacks and Blocks. Stacks are responsible for capturing different trends and patterns in the series, while blocks are the key components for the stack, they learn the complex temporal patterns in the time series. The final prediction is obtained by summing the partial predictions from each stack.

The main modification in N-HiTS is the introduction of multi-rate signal sampling, enabling it to capture both short-term and long-term effects in the time series. It is achieved by including the MaxPool layer at the block-level. Since the MaxPool layers perform downsampling by selecting the maximum value within a given set of values. Each stack has its own kernel size which determines the sampling rate. A larger kernel size directs the stack's focus towards long-term effects in the time series, while a smaller kernel size emphasizes short-term effects within the series.

LSTM Architecture

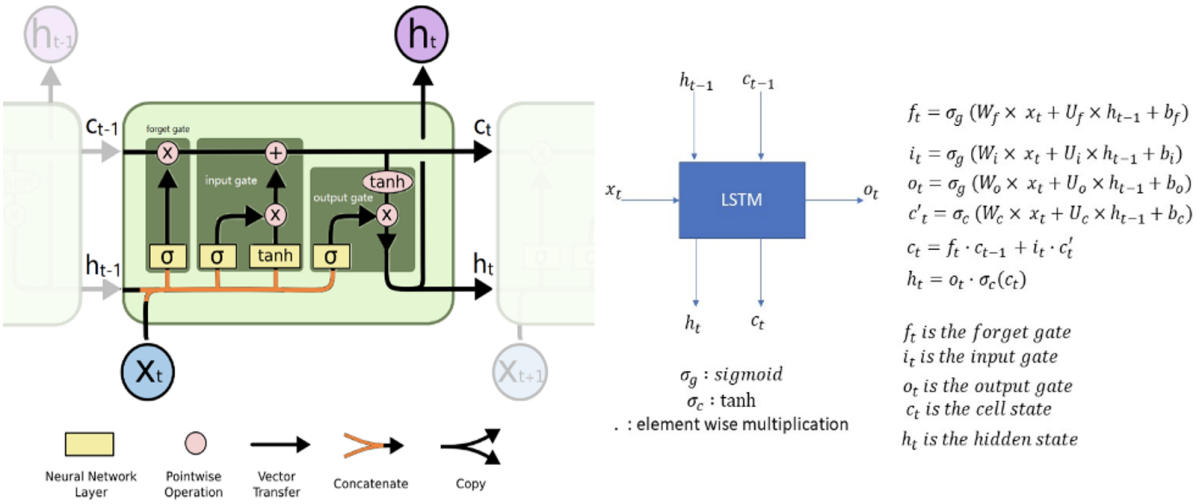


Figure 13. LSTM Architecture

LSTM (Long Short-Term Memory) is a type of recurrent neural network (RNN) that can capture and retain long-term dependencies in sequential data. From its architecture, it contains the memory cell which stores information over time utilizing a set of "gates" – the forget gate, input gate and output gate. At the forget date, it takes in the input data and the previous hidden state, and determines what information from the previous time step is irrelevant and should be discarded. Then at the input gate, it determines the amount of new information to be added to the memory cell as long-term memory. Finally , the output gate ensures that crucial information from the long-term memory is utilized to update the new hidden state.

Implementation

We use “neuralforest” open source package in python to fit the models with hyper-parameters tuning and cross validation. First, we pre defined the parameters searching space for each of the models. We set the horizon to be 365 days which is the prediction period and the input size to be the multiples of the horizons. Due to computational time, 30 combinations are randomly chosen in the pre-defined parameters space when performing the cross-validation. Refer to the appendix for the full set of pre-defined parameters.

After the hyper-parameters tunings, below are the parameters selected for the best performing models, both NHiTS and LSTM have the input size of 1460, which means that they take in 4 years (365*4) data in predicting the next 365 days' temperature.

Model	NHiTS	LSTM
Parameters	{'input_size': 1460, 'start_padding_enabled': True, 'n_blocks': [1, 1, 1, 1, 1], 'mlp_units': [[64, 64], [64, 64], [64, 64], [64, 64], [64, 64]], 'n_pool_kernel_size': [8, 4, 2, 1, 1], 'n_freq_downsample': [1, 1, 1, 1, 1], 'learning_rate': 0.0004110143838851897, 'scaler_type': 'robust', 'max_steps': 1000, 'batch_size': 4, 'windows_batch_size': 512, 'random_seed': 2, 'h': 365, 'loss': MAE(), 'valid_loss': MAE()}	{'input_size': 1460, 'encoder_hidden_size': 128, 'encoder_n_layers': 4, 'learning_rate': 0.003758491243094816, 'scaler_type': 'robust', 'max_steps': 1000, 'batch_size': 1, 'random_seed': 7, 'h': 365, 'loss': MAE(), 'valid_loss': MAE()}

Models Performance

Since both models target long-term prediction, they may not capture the daily temperature fluctuation as shown in the graph. However, we can obviously see the trend for the one year horizon. Among both models, NHiTS performs slightly better than LSTM, with RMSE of 2.585.

Model	Mean Absolute Error (MAE)	Mean Square Error (MSE)	Root Mean Square Error (RMSE)
NHiTS	2.001	6.680	2.585
LSTM	2.195	7.969	2.823

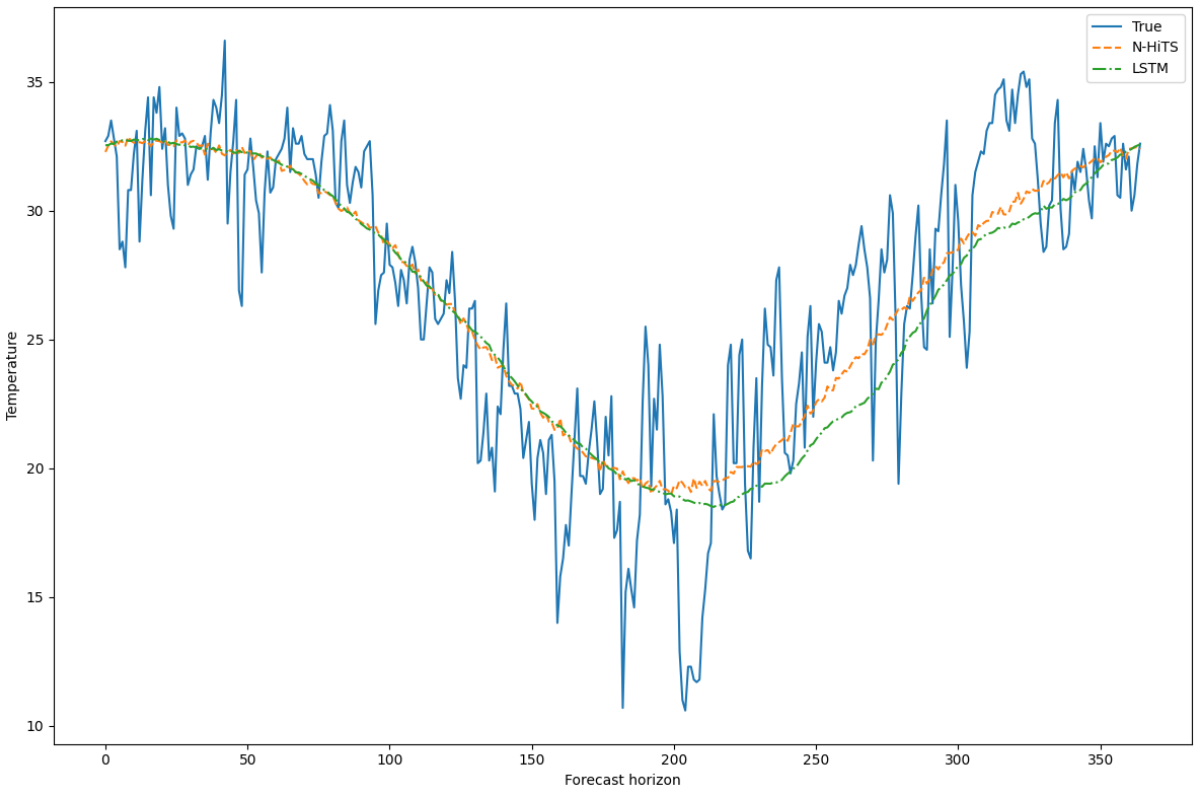


Figure 14. N-HiTS & LSTM Predictions vs True Value

Discussion & Conclusion

The ARIMA model showed some limitations in capturing extreme temperature scenarios, particularly at lower temperatures, due to the normality assumption not holding. Additionally, external factors such as humidity and the seasonal effects can impact temperature. To improve accuracy, exogenous variables can be incorporated into the ARIMA model (i.e., ARIMX) to capture the influence of these factors on the response series.

This study also compared ARIMA, Transformer, N-HiTs and LSTM models for temperature forecasting. While the traditional ARIMA approach demonstrated reasonably good predictive performance, it is less able to capture the rich sequential dependencies that deep learning methods like Transformers can model via self-attention mechanisms. However, the performance gap between ARIMA and Transformers was not large in terms of RMSE and MAE. Furthermore, the N-HiTs and LSTM models show potential for extracting long-term seasonal trends.

Overall, while ARIMA remains a suitable baseline as it is simpler and requires less computational power, deep learning models like transformers demonstrate its ability to better capture how predicted values correlate with observations at various lag times in the historical period. In the future study, both traditional methods and deep learning methods can be used jointly to leverage the advantages from both approaches, for example, we could adopt a two-step approach which uses deep learning models to capture the long-term patterns and the seasonal patterns, then use ARIMA to capture the short-term patterns. In addition, both types of models can consider incorporating exogenous variables for a better prediction with more holistic information.

Reference

Applications of Deep Neural Networks. (n.d.). GitHub. Retrieved December 31, 2023, from https://github.com/jeffheaton/app_deep_learning/blob/main/t81_558_class_10_3_transformer_timeseries.ipynb

Vaswani, A. (2017, June 12). *[1706.03762] Attention Is All You Need*. arXiv. Retrieved December 31, 2023, from https://arxiv.org/abs/1706.03762

Wu, H. (2021, June 24). *[2106.13008] Autoformer: Decomposition Transformers with Auto-Correlation for Long-Term Series Forecasting*. arXiv. Retrieved December 31, 2023, from https://arxiv.org/abs/2106.13008

Boris N. Oreshkin, Dmitri Carpov, Nicolas Chapados, Yoshua Bengio (2020, Feb 20) *[1905.10437] N-BEATS: Neural basis expansion analysis for interpretable time series forecasting. arXiv. Retrieved December 31, 2023, from https://arxiv.org/abs/1905.10437*

Cristian Challu, Kin G. Olivares, Boris N. Oreshkin, Federico Garza, Max Mergenthaler-Canseco, Artur Dubrawski (2022, Nov 29). *[2201.12886] N-HiTS: Neural Hierarchical Interpolation for Time Series Forecasting. Retrieved December 31, 2023, from https://arxiv.org/abs/2201.12886*

Christian Bakke Vennerød, Adrian Kjærran, Erling Stray Bugge (2021, May 14). *[2105.06756] Long Short-term Memory RNN. Retrieved December 31, 2023, from https://arxiv.org/abs/2105.06756*

Appendices

Code For ARIMA

```
import pandas as pd
from pandas import Series
from pandas import read_csv
from pandas import to_datetime
from sklearn.metrics import mean_squared_error
from math import sqrt
import matplotlib.pyplot as plt
from sklearn.preprocessing import MinMaxScaler
from statsmodels.tsa.stattools import adfuller
import statsmodels.api as sm
import warnings
warnings.filterwarnings("ignore")
from statsmodels.tsa.arima.model import ARIMA
from statsmodels.graphics.tsaplots import plot_predict
import numpy as np
#Load Dataset
df = pd.read_csv("CLMMAXT_HKO.csv")
df['Date'] = pd.to_datetime(df['Date'], format='%d/%m/%Y')
df.set_index('Date', inplace=True)
df.sort_index(inplace= True)
#Extract 2020-12-01 to 2023-11-30 time period
temp_df = df.loc['2020-12-01':'2023-11-30']
temp_df

# Plot the daily temperature series
plt.figure(figsize=(10,4), dpi=100)
plt.plot(temp_df.index, temp_df.Degrees_Celsius, color='tab:red')
plt.gca().set(title="Daily Maximum Temperature in HK from 2020 to 2023", xlabel='Date', ylabel="Degree (in Celsius)")
plt.show()

def adf_test(timeseries):
    #Perform Dickey-Fuller test:
    print("Results of Dickey-Fuller Test\n==============================================")
    dftest = adfuller(timeseries, autolag="AIC")
    dfoutput = pd.Series(dftest[0:4], index = [
        "Test Statistic", "p-value", "#Lags Used", "Number of Observations Used"])
    for key, value in dftest[4].items():
        dfoutput["Criterical Value (%s)"%key] = value
    print(dfoutput)
    print("==============================================")
    if dfoutput[0] < dfoutput[4]:
        print("The data is stationary. (Criterical Value 1%)")
    elif dfoutput[0] < dfoutput[5]:
        print("The data is stationary. (Criterical Value 5%)")
    elif dfoutput[0] < dfoutput[6]:
        print("The data is stationary. (Criterical Value 10%)")
    else:
        print("The data is non-stationary, so do differencing!")
```

```
def basic_check(model):
    # create and run statistical tests on model
    norm_val, norm_p, skew, kurtosis = model.test_normality('jarquebera')[0]
    lb_val, lb_p = model.test_serial_correlation(method='ljungbox')[0]
    lb_val_res = sm.stats.acorr_ljungbox(model.resid, lags=[10]).values[0][0].tolist()
    lb_p_res = sm.stats.acorr_ljungbox(model.resid, lags=[10]).values[0][1].tolist()
    het_val, het_p = model.test_heteroskedasticity('breakvar')[0]
    lb_val = lb_val[-1]
    lb_p = lb_p[-1]
    durbin_watson = sm.stats.stattools.durbin_watson(model.filter_results.standardized_forecasts_error[0, model.loglikelihoo

    print('Normality: val={:.3f}, p={:.3f}'.format(norm_val, norm_p));
    print('Ljung-Box (stand res): val={:.3f}, p={:.3f}'.format(lb_val, lb_p));
    print('Ljung-Box (res): val={:.3f}, p={:.3f}'.format(lb_val_res, lb_p_res));
    print('Heteroskedasticity: val={:.3f}, p={:.3f}'.format(het_val, het_p));
    print('Durbin-Watson: d={:.2f}'.format(durbin_watson))

def measure_rmse(actual, predicted):
    #Get the RMSE
    return sqrt(mean_squared_error(actual, predicted))

#ADF test and basis test for orignal series
print("Orignal Time Series (Daily Max. Temp from 2020 - 2023)")
adf_test(temp_df.Degrees_Celsius)
temp_df.plot(figsize=(10,4))

#ADF test and basis test for first order differecing series
print("After first order differencing (Daily Max. Temp from 2020 - 2023)")
temp_diff1 = temp_df.diff(1)
temp_diff1 = temp_diff1.dropna()
adf_test(temp_diff1)

plt.figure(figsize=(10,4), dpi=100)
plt.plot(temp_diff1.index, temp_diff1.Degrees_Celsius, color='tab:red')
plt.gca().set(title="Daily Maximum Temperature (First differecning) in HK from 2020 to 2023", xlabel='Date', ylabel="Degree
plt.show()

#Split the data for train and test for PACF and ACF plot
train = temp_df.loc['2020-12-01':'2023-11-25']
test = temp_df.loc['2023-11-26':'2023-11-30']
train_diff1 = temp_diff1.loc['2020-12-01':'2023-11-25']
test_diff1 = temp_diff1.loc['2023-11-26':'2023-11-30']

#Sample ACF and PACF Plot for orignal Series
from statsmodels.graphics.tsaplots import plot_acf, plot_pacf
plot_acf(train['Degrees_Celsius'],lags=500)
plot_pacf(train['Degrees_Celsius'],lags=500)
plot_acf(train['Degrees_Celsius'],lags=50)
plot_pacf(train['Degrees_Celsius'],lags=50)

#Sample ACF and PACF Plot for first order differencing Series
plot_acf(train_diff1['Degrees_Celsius'],lags=500)
plot_pacf(train_diff1['Degrees_Celsius'],lags=500)
plot_acf(train_diff1['Degrees_Celsius'],lags=50)
plot_pacf(train_diff1['Degrees_Celsius'],lags=50)
```

```python
#Plot the residual ACF and PACF and model diagnostics anaylsis for the fitted model
def plot_res_acf_pacf(data, model_result, start_point):
    train_error = measure_rmse(data['Degrees_Celsius'].iloc[start_point:], model_result.fittedvalues[start_point:])
    print("training error (rmse):", train_error)
    res = model_result.resid
    fig, ax = plt.subplots(2,1,figsize=(15,8))
    fig = sm.graphics.tsa.plot_acf(res, lags=544, ax=ax[0])
    fig = sm.graphics.tsa.plot_pacf(res, lags=544, ax=ax[1])
    custom_ylim = (-1, 1)
    plt.setp(ax, ylim=custom_ylim)
    ax[0].set_xlim(0, 544)
    ax[1].set_xlim(0, 544)
    plt.show()
    fig, ax = plt.subplots(2,1,figsize=(15,8))
    fig = sm.graphics.tsa.plot_acf(res, lags=50, ax=ax[0])
    fig = sm.graphics.tsa.plot_pacf(res, lags=50, ax=ax[1])
    custom_ylim = (-0.2, 0.2)
    plt.setp(ax, ylim=custom_ylim)
    ax[0].set_xlim(0, 50)
    ax[1].set_xlim(0, 50)
    plt.show()
    basic_check(model_result)
    model_result.plot_diagnostics(figsize=(14,10))
    plt.show()
    print(model_result.summary())

#Grid Search to select lowest AIC value ARIMA Model
def addlabels(x,y):
    for i in range(len(x)):
        plt.text(i,y[i],y[i], ha = 'center')

def arima_AIC(data, p=3, d=3, q=3):
    best_AIC =["pdq",10000]
    L =len(data)
    AIC = []
    name = []
    for i in range(p):
        for j in range(1,d):
            for k in range(q):
                fitted = sm.tsa.statespace.SARIMAX(data, order=(i,j,k)).fit(max_iter = 50, method = 'powell', disp=False)
                #fitted = model.fit()
                AIC.append(round(fitted.aic,1))
                name.append(f"ARIMA({i},{j},{k})")
                print(f"ARIMA({i},{j},{k}) : AIC={fitted.aic}")
                if fitted.aic < best_AIC[1]:
                    best_AIC[0] = f"ARIMA({i},{j},{k})"
                    best_AIC[1] = fitted.aic
    print("=================================================================================")
    print(f"This best model is {best_AIC[0]} based on argmin AIC.")
    plt.figure(figsize=(12,5))
    plt.bar(name, AIC)
    plt.bar(best_AIC[0], best_AIC[1], color = "red")
    addlabels(name, AIC)
    plt.xticks(rotation=80)
    plt.title("AIC")
    plt.savefig("Arima AIC")
    plt.show()


arima_AIC(train.Degrees_Celsius, 4,2,4)
```

```
# Fit the SARIMAX model for potential model 1
mod_013_000 = sm.tsa.statespace.SARIMAX(train['Degrees_Celsius'],
                                order=(0, 1, 3),
                                seasonal_order=(0, 0, 0, 365),
                                enforce_stationarity=True,
                                enforce_invertibility=True)

results_013_000 = mod_013_000.fit(max_iter = 50, method = 'powell', disp=False)
plot_res_acf_pacf(train,results_013_000, 1)

# Performance the LjungBox test for potential model 1
sm.stats.acorr_ljungbox(results_013_000.resid, lags=[1,2,3,4,5,6,7,8,9,10, 30, 60, 120, 180, 365, 730], return_df=True)

# Fit the SARIMAX model for potential model 2
mod_212_000 = sm.tsa.statespace.SARIMAX(train['Degrees_Celsius'],
                                order=(2, 1, 2),
                                seasonal_order=(0, 0, 0, 365),
                                enforce_stationarity=True,
                                enforce_invertibility=True)

results_212_000 = mod_212_000.fit(max_iter = 50, method = 'powell', disp=False)
plot_res_acf_pacf(train,results_212_000, 366)

# Performance the LjungBox test for potential model 2
sm.stats.acorr_ljungbox(results_212_000.resid, lags=[1,2,3,4,5,6,7,8,9,10, 14, 30, 60, 120, 180, 365, 730], return_df=True)

# Fit the SARIMAX model for potential model 3
mod_112_000 = sm.tsa.statespace.SARIMAX(train['Degrees_Celsius'],
                                order=(1, 1, 2),
                                seasonal_order=(0, 0, 0, 365),
                                enforce_stationarity=True,
                                enforce_invertibility=True)

results_112_000 = mod_112_000.fit(max_iter = 50, method = 'powell', disp=False)
plot_res_acf_pacf(train,results_112_000, 1)
# Performance the LjungBox test for potential model 3
sm.stats.acorr_ljungbox(results_112_000.resid, lags=[1,2,3,4,5,6,7,8,9,10, 30, 60, 120, 180, 365, 730], return_df=True)

# Fit the SARIMAX model for potential model 4 (over-parameterized)
mod_113_000 = sm.tsa.statespace.SARIMAX(train['Degrees_Celsius'],
                                order=(1, 1, 3),
                                seasonal_order=(0, 0, 0, 365),
                                enforce_stationarity=True,
                                enforce_invertibility=True)

results_113_000 = mod_113_000.fit(max_iter = 50, method = 'powell', disp=False)
plot_res_acf_pacf(train,results_113_000, 1)
# Performance the LjungBox test for potential model 4 (over-parameterized)
sm.stats.acorr_ljungbox(results_113_000.resid, lags=[1,2,3,4,5,6,7,8,9,10, 30, 60, 120, 180, 365, 730], return_df=True)
```

```
#Forecast based on final Model ARIMA(1,1,2)
preds_df = results_112_000.get_forecast(steps=5).summary_frame(alpha=0.05)
preds_df = pd.concat([test,preds_df],axis=1)
preds_df

# Plot the training data, predicted means and confidence intervals
fig, ax = plt.subplots(figsize=(15,5))
ax = train['Degrees_Celsius'].iloc[1050:].plot(label='Training Data')
ax.set(
    title='True and Predicted Values, with Confidence Intervals',
    xlabel='Date',
    ylabel='Actual / Predicted Values'
)
preds_df['mean'].plot(ax=ax, style='r', label='Predicted Mean')
test['Degrees_Celsius'].plot(ax=ax, style='b', label='Test Data')
ax.fill_between(
    preds_df.index, preds_df['mean_ci_lower'], preds_df['mean_ci_upper'],
    color='r', alpha=0.1
)
legend = ax.legend(loc='upper left')
plt.show()
```

Code For Transformer

```python
#!/usr/bin/env python
# coding: utf-8

import pandas as pd
import numpy as np
import statsmodels.api as sm
import matplotlib.pyplot as plt
import seaborn as sns
import pmdarima as pm
import torch
import torch.nn as nn
from torch.utils.data import DataLoader, TensorDataset
from sklearn.preprocessing import RobustScaler, MinMaxScaler, StandardScaler
from sklearn.metrics import mean_absolute_error, mean_squared_error
from torch.optim.lr_scheduler import ReduceLROnPlateau
import warnings
warnings.filterwarnings('ignore')

# # Load Dataset

# Data set URL: https://data.gov.hk/en-data/dataset/hk-hko-rss-daily-temperature-info-hko/resource/1a78c69f-d5ee-4a0d-bb76-1e52aa97ac56

df = pd.read_csv("data/CLMMAXT_HKO_.csv", parse_dates={'date': ['year', 'month', 'day']}, index_col='date')
df = df.drop('data_completeness', axis=1)['1960-01-01':].astype(float)
df

# # Build Model

device = (
    "mps"
    if torch.backends.mps.is_available()
    else "cuda"
    if torch.cuda.is_available()
    else "cpu"
)
print(f"Using device: {device}")

cutoff_date = '2020-01-01'
df_train = df[:cutoff_date]
df_test = df[cutoff_date:]
target_col = 'value'
seq_len_train = 365

temp_train = df_train[target_col].to_numpy().reshape(-1, 1)
temp_test = df_test[target_col].to_numpy().reshape(-1, 1)

scaler = MinMaxScaler()
temp_train = scaler.fit_transform(temp_train).flatten().tolist()
temp_test = scaler.transform(temp_test).flatten().tolist()
```

```python
# Create dataloader and generate dataset
def to_sequences(seq_size, obs):
    x = []
    y = []

    for i in range(len(obs) - seq_size):
        window = obs[i:(i + seq_size)]
        after_window = obs[i + seq_size]
        x.append(window)
        y.append(after_window)
    return torch.tensor(x, dtype=torch.float32).view(-1, seq_size, 1), torch.tensor(y, dtype=torch.float32).view(-1, 1)

x_train, y_train = to_sequences(seq_len_train, temp_train)
x_test, y_test = to_sequences(seq_len_train, temp_test)

# Setup data loaders for batch data
train_dataset = TensorDataset(x_train, y_train)
train_loader = DataLoader(train_dataset, batch_size=16, shuffle=False)

test_dataset = TensorDataset(x_test, y_test)
test_loader = DataLoader(test_dataset, batch_size=16, shuffle=False)

# Positional Encoding for Transformer
class PositionalEncoding(nn.Module):
    def __init__(self, d_model, dropout=0.1, max_len=5000):
        super().__init__()
        self.dropout = nn.Dropout(p=dropout)

        pos_encoder = torch.zeros(max_len, d_model)
        position = torch.arange(0, max_len, dtype=torch.float).unsqueeze(1)
        div_term = torch.exp(torch.arange(0, d_model, 2).float() * (-np.log(10000.0) / d_model))
        pos_encoder[:, 0::2] = torch.sin(position * div_term)
        pos_encoder[:, 1::2] = torch.cos(position * div_term)
        pos_encoder = pos_encoder.unsqueeze(0).transpose(0, 1)
        self.register_buffer('pos_encoder', pos_encoder)

    def forward(self, x):
        x = x + self.pos_encoder[:x.size(0), :]
        return self.dropout(x)

# Model definition using Transformer
class TransformerModel(nn.Module):
    def __init__(self, input_dim=1, d_model=64, nhead=4, num_layers=2, dropout=0.2):
        super().__init__()

        self.encoder = nn.Linear(input_dim, d_model)
        self.pos_encoder = PositionalEncoding(d_model, dropout)
        encoder_layers = nn.TransformerEncoderLayer(d_model, nhead)
        self.transformer_encoder = nn.TransformerEncoder(encoder_layers, num_layers)
        self.decoder = nn.Linear(d_model, 1)

    def forward(self, x):
        x = self.encoder(x)
        x = self.pos_encoder(x)
        x = self.transformer_encoder(x)
        x = self.decoder(x[:, -1, :])
        return x

model = TransformerModel().to(device)
```

```python
# Train the model
criterion = nn.MSELoss()
optimizer = torch.optim.Adam(model.parameters(), lr=0.001)
scheduler = ReduceLROnPlateau(optimizer, 'min', factor=0.5, patience=3, verbose=True)

epochs = 100
early_stop_count = 0
min_val_loss = float('inf')

loss_res = []
for epoch in range(epochs):
    model.train()
    for batch in train_loader:
        x_batch, y_batch = batch
        x_batch, y_batch = x_batch.to(device), y_batch.to(device)

        optimizer.zero_grad()
        outputs = model(x_batch)
        loss = criterion(outputs, y_batch)
        loss.backward()
        optimizer.step()

    # Validation
    model.eval()
    val_losses = []
    with torch.no_grad():
        for batch in test_loader:
            x_batch, y_batch = batch
            x_batch, y_batch = x_batch.to(device), y_batch.to(device)
            outputs = model(x_batch)
            loss = criterion(outputs, y_batch)
            val_losses.append(loss.item())

    val_loss = np.mean(val_losses)
    scheduler.step(val_loss)
    loss_res.append(val_loss)

    if val_loss < min_val_loss:
        min_val_loss = val_loss
        early_stop_count = 0
    else:
        early_stop_count += 1

    if early_stop_count >= 5:
        print("Early stop")
        break
    print(f"Epoch {epoch + 1}/{epochs}, Validation Loss: {val_loss:.4f}")

plt.close('all')
plt.plot(loss_res)
plt.title('Validation Loss vs epoch')
plt.show()
```

```
model.eval()
predictions = []
with torch.no_grad():
    for batch in test_loader:
        x_batch, y_batch = batch
        x_batch = x_batch.to(device)
        outputs = model(x_batch)
        predictions.extend(outputs.squeeze().tolist())

y_pred = scaler.inverse_transform(np.array(predictions).reshape(-1, 1))
y_true = scaler.inverse_transform(y_test.numpy().reshape(-1, 1))

mae = mean_absolute_error(y_true, y_pred)
mse = mean_squared_error(y_true, y_pred)
rmse = np.sqrt(mse)
print(f"Score (MAE): {mae:.4f}")
print(f"Score (MSE): {mse:.4f}")
print(f"Score (RMSE): {rmse:.4f}")

df['pred'] = ([np.nan] * (len(df) - len(y_pred)) + y_pred.flatten().tolist())
df['2019-01-01':].rename(columns={'value': 'true'}).plot(title='Transformer Prediction vs True')
df['2022-01-01':'2023-01-01'].rename(columns={'value': 'true'}).plot(title='Transformer Prediction vs True (Zoomed In)').plot()
```

Code for N-HiTS and LSTM

```
import pandas as pd
import numpy as np
import datetime
import matplotlib.pyplot as plt
from neuralforecast import NeuralForecast
from neuralforecast.models import LSTM, NHITS
from ray import tune
from neuralforecast.losses.pytorch import MQLoss
from neuralforecast.auto import AutoNHITS,AutoLSTM
from neuralforecast.core import NeuralForecast
from neuralforecast.losses.numpy import mae, mse, rmse
import warnings
warnings.filterwarnings('ignore')

#load data
df = pd.read_csv('CLMMAXT_HKO_.csv', header=2)
df = df[:-3]

# filter out invalid record
df = df[~df['數據完整性/data Completeness'].isna()].reset_index(drop=True)

# preprocess
df['年/Year'] = df['年/Year'].astype(int)
df['月/Month'] = df['月/Month'].astype(int)
df['日/Day'] = df['日/Day'].astype(int)

df['Year'] = df['年/Year'].astype(str).str.zfill(4)
df['Month'] = df['月/Month'].astype(str).str.zfill(2)
df['Day'] = df['日/Day'].astype(str).str.zfill(2)

# convert to datetime object
df['datetime'] = df['Year'] + df['Month'] + df['Day']
df['datetime'] = pd.to_datetime(df['datetime'])

# consider just the date after 1960
df_simplified = df[df['年/Year']>=1960].reset_index(drop=True)

# prepare the data following the format required from "neuralforecast"
df_simplified['unique_id'] = 1
df_format = df_simplified[['unique_id','datetime','數值/Value']]
df_format.columns = ['unique_id','ds','y']
df_format['y'] = df_format['y'].astype('float')

# set the validation and testing size
n_time = len(df_format.ds.unique())
val_size = int(.1 * n_time)
test_size = int(.1 * n_time)
```

```
horizon = 365

# Define the parameters searching space
config_nhits = {
    "input_size": tune.choice([horizon, horizon*2, horizon*3]),     # Length of input window
    "start_padding_enabled": True,
    "n_blocks": 5*[1],                                              # Length of input window
    "mlp_units": 5 * [[64, 64]],                                    # Length of input window
    "n_pool_kernel_size": tune.choice([5*[1], 5*[2], 5*[4],
                                       [8, 4, 2, 1, 1]]),           # MaxPooling Kernel size
    "n_freq_downsample": tune.choice([[8, 4, 2, 1, 1],
                                      [1, 1, 1, 1, 1]]),            # Interpolation expressivity ratios
    "learning_rate": tune.loguniform(1e-4, 1e-2),                   # Initial Learning rate
    "scaler_type": tune.choice([None]),                             # Scaler type
    "max_steps": tune.choice([1000]),                               # Max number of training iterations
    "batch_size": tune.choice([1, 4, 10]),                          # Number of series in batch
    "windows_batch_size": tune.choice([128, 256, 512]),             # Number of windows in batch
    "random_seed": tune.randint(1, 20),                             # Random seed
}

config_lstm = {
    "input_size": tune.choice([horizon, horizon*2, horizon*3]),      # Length of input window
    "encoder_hidden_size": tune.choice([64, 128]),                   # Hidden size of LSTM cells
    "encoder_n_layers": tune.choice([2,4]),                          # Number of layers in LSTM
    "learning_rate": tune.loguniform(1e-4, 1e-2),                    # Initial Learning rate
    "scaler_type": tune.choice(['robust']),                          # Scaler type
    "max_steps": tune.choice([500, 1000]),                           # Max number of training iterations
    "batch_size": tune.choice([1, 4]),                               # Number of series in batch
    "random_seed": tune.randint(1, 20),                              # Random seed
}
```

```
# define model
nf = NeuralForecast(
    models=[
        AutoNHITS(h=horizon, config=config_nhits, loss=MQLoss(), num_samples=30),
        AutoLSTM(h=horizon, config=config_lstm, loss=MQLoss(), num_samples=30),
    ],
    freq='D'
)

# perform cross validations
preds_df = nf.cross_validation(df=df_format, val_size=val_size, test_size=test_size, n_windows=None)
preds_df.columns = preds_df.columns.str.replace('-median', '')
```

```
# get evaluation result
y_true = preds_df['y'].values
y_pred_nhits = preds_df['AutoNHITS'].values
y_pred_lstm = preds_df['AutoLSTM'].values

n_series = len(df_format['unique_id'].unique())

y_true = y_true.reshape(n_series, -1, horizon)
y_pred_nhits = y_pred_nhits.reshape(n_series, -1, horizon)
y_pred_lstm = y_pred_lstm.reshape(n_series, -1, horizon)

overall_result = {'models':['NHiTs','LSTM'],
                  'MAE':[mae(y_pred_nhits, y_true), mae(y_pred_lstm, y_true)],
                  'MSE':[mse(y_pred_nhits, y_true), mse(y_pred_lstm, y_true)],
                  'RMSE':[rmse(y_pred_nhits, y_true), rmse(y_pred_lstm, y_true)]
                  }

pd.DataFrame(overall_result)
```

```
# visualisation
fig, ax = plt.subplots(figsize=(12,8))
ax.plot(y_true[0, 0, :], label='True')
ax.plot(y_pred_nhits[0, 0, :], label='N-HiTS', ls='--')
ax.plot(y_pred_lstm[0, 0, :], label='LSTM', ls='-.')
ax.set_ylabel('Temperature')
ax.set_xlabel('Forecast horizon')
ax.legend(loc='best')
plt.tight_layout()
```