**CHAPTER 13:** Sales Forecasting with Auto Regressive Recurrent Networks

- 25 pages

**Traditional Time Series Forecasting**

Demand forecasting is key to many industries, ranging from Airlines to Retail to Telecommunications to HealthCare. Inaccurate and imprecise demand forecasting leads to missed sales and customers, significantly impacting an organization’s bottom line. One of the key challenges facing retailers is effectively managing inventory based on multiple internal and external factors. Inventory management is a complex business problem to solve – the demand for a product changes by location, weather, promotions, holidays, day of the week, special events, and other external factors, such as store demographics, consumer confidence, and unemployment

In this chapter, we are going to learn the following topics:

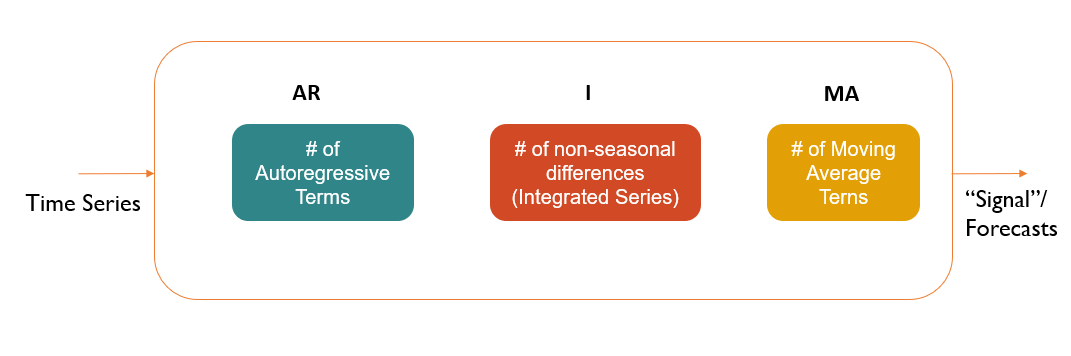
* For time series forecasting, how are traditional techniques, such as ARIMA and Exponential Smoothing different from neural network based techniques
* How does DeepAR (Deep Auto Regressive) from Amazon SageMaker work
  + A brief look at sequence-to-sequence learning – mapping historical sales to future sales
* Deep dive into the DeepAR model architecture
* To illustrate how DeepAR predicts future sales, we will model retail [dataset](https://www.kaggle.com/manjeetsingh/retaildataset) to predict weekly sales, given multiple factors such as holidays, promotions, and macro-economic indicators (unemployment)

ARIMA and Exponential Smoothing to model demand in simple use cases:

*ARIMA (Auto-Regressive Integrated Moving Average)*

ARIMA is time series analytical technique used to capture different temporal structures in univariate data. To model the time series data, differencing is applied across the series to make the data stationary. In other words, the mean and variance of the probability distribution of the time series does not change over time. A specific number of lagged forecasts and forecast errors are used to model time series. This number is iteratively adjusted until the residuals are uncorrelated with target (sales forecast) or all the signal in the data has been picked up.

*Sales Forecast = Constant term + Auto-Regressive terms + Moving Average terms (lags of forecast errors)*



ARIMA consists of three components:

* Number of autoregressive terms – future sales will look like past sales (“past” could be any number of time periods)
* Number of non-seasonal differences – future sales will look like past sales if the difference in sales during last few periods is very small
* Number of moving-average terms (lagged forecast errors) – define error of the ARIMA model as linear combination of errors that occurred at previous time points in the past

In the ARIMA model, the AR terms are positive, while the MA terms are negative – to explain, the Auto-regressive terms have positive impact on sales forecast, while moving average of lagged errors has negative impact.

*Exponential Smoothing*

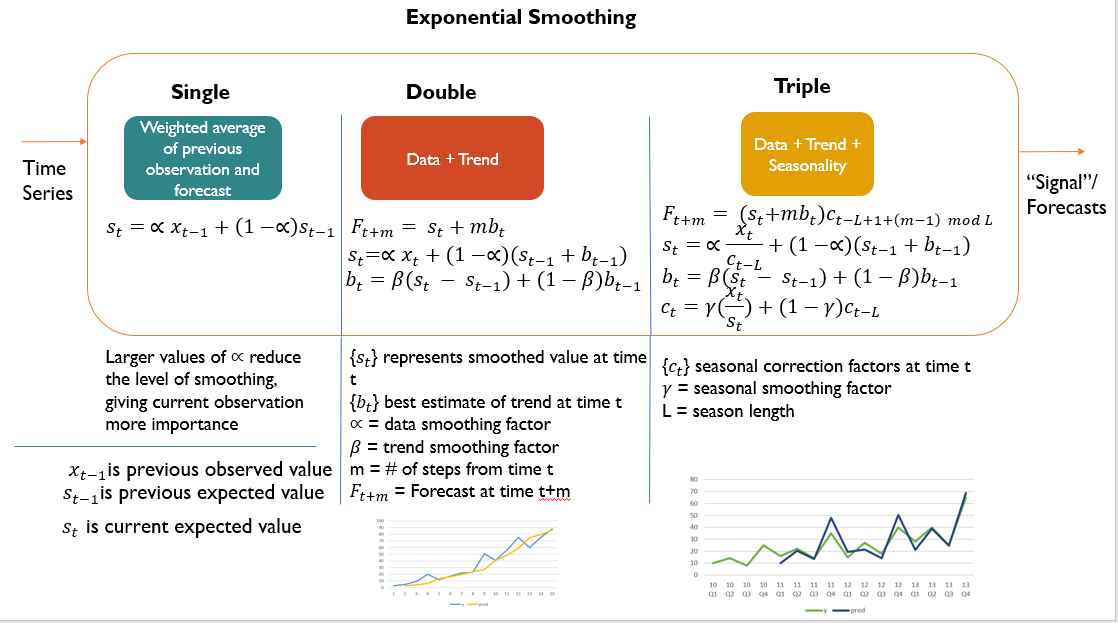
The other alternative to ARIMA is Exponential Smoothing technique, which is also a time series forecasting method for univariate data, where random noise is neglected, revealing underlying time structure. Although it is like ARIMA in that demand forecast is a weighted sum of past observations, the method of applying weights to lagged observations is different -- instead of providing equal weights to past observations, the model employs exponentially decreasing weights for lags. In other words, most recent observations are more relevant than historical ones. Exponential smoothing is used to make short-term forecasts, where we use assume that future patterns and trends will look like current patterns and trends.

There are 3 types of exponential smoothing methods:

*Single exponential smoothing*: As the name indicates, the technique does not account for seasonality or trend. It requires a single parameter, alpha (, to control how much recent observations are important relative to historical ones. Low alpha means there are no irregularities in the data, implying that latest observations are given lower weight. To forecast sales at time t, you will add two products -- and , where is previous observation and is previous forecast. In other words, forecast at time t is weighted average of sales at t-1 and forecast at t-1.

*Double exponential smoothing*: This technique, on the other hand, supports trends in univariate series. In addition to controlling how important recent observations are relative to historical ones, an additional factor is used to control the influence of trend component on demand forecasts. The trend can be either multiplicative or additive and is controlled using a smoothing factor, . See below for how to apply double exponential smoothing

*Triple exponential smoothing*: This one adds support for seasonality. Another new parameter, gamma (, is used to control the influence of the seasonal component on demand forecasts. The seasonality influence can either be multiplicative or additive. In this case, we will look at multiplicative effect.



In all the 3 variants of exponential smoothing, the parameters are derived by optimizing error metrics, such as MSE (mean squared error).

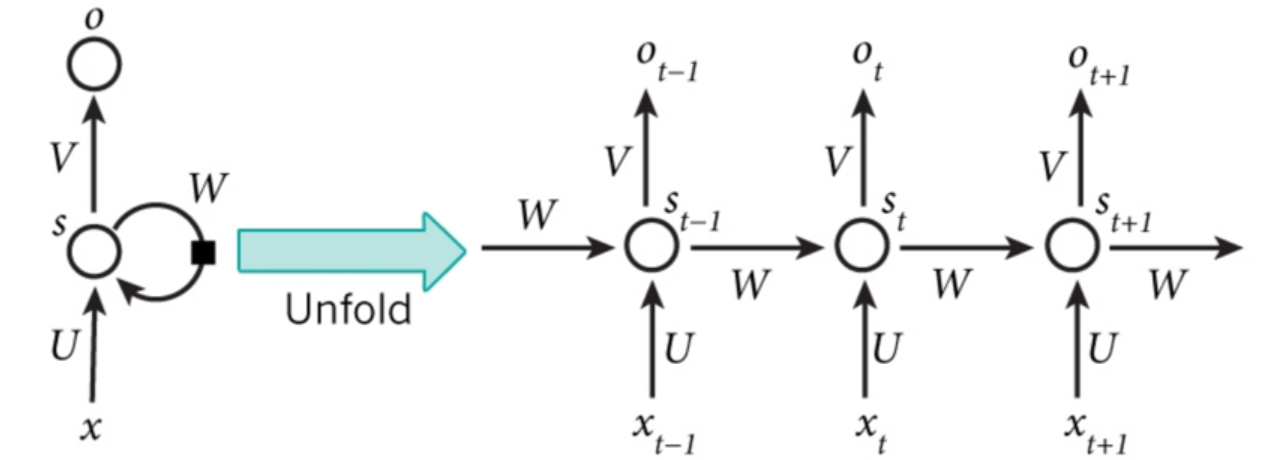
The challenge with techniques such as ARIMA and Exponential Smoothing is that they can model an individual time-series and do not scale well to hundreds and thousands of time-series. Besides, they are all forecasting techniques for univariate time series. As detailed earlier, there could be other factors that impact current and future sales, such as weather, promotions, day of the week, holidays and special-events. Let’s look at a different approach to solving forecasting challenge.

**How DeepAR Model Works**

The DeepAR algorithm offered by Sagemaker is generalized deep learning model that learns demand across several related time series. Unlike traditional forecasting methods, in which individual time series is modeled, DeepAR models thousands or millions of related time series.

Examples include forecasting load for servers in a data center or forecasting demand for all products that a retailer offers, energy consumption of individual households. The unique thing about this approach is that a substantial amount of data on past behavior of similar or related time series can be leveraged for forecasting an individual time series. This approach addresses overfitting issue, time and labor intensive manual feature engineering and model selection steps required by traditional techniques.

DeepAR, a forecasting method based on autoregressive neural networks, learns a global model from historical data of all time-series in the data set. DeepAR employs LSTM (Long Short-Term Memory), a type of Recurrent Neural Network (RNN), to model time series. The main idea of RNNs is to capture sequential information. Unlike normal neural networks, the inputs (and outputs) are dependent on each other. RNNs thus have a “memory” that captures information about what has been estimated so far.



*A recurrent neural network and illustration of sequential learning as the time steps are unfolded. Source: Nature; Image Credits*[*WildML*](http://www.wildml.com/2015/09/recurrent-neural-networks-tutorial-part-1-introduction-to-rnns/)

* is input at a time t
* is hidden state at time t. This state is computed based on previous hidden state and current input. ; function f is an activation function
* is output at time t; .The activation function f can vary depending on the use case. For example, softmax activation function is used when we need to predict which of the classes does the input belong to. In other words, whether the image being detected is a cat or a dog or a giraffe.
* The network weights, U, V and W, remain same across all the time steps.

RNNs have interesting applications in fields, such as:

* Natural Language Processing – from generating image captions to generating text to machine translations, RNNs can act as generative models
* Autonomous Cars – to conduct dynamic facial analysis
* Time series – used in econometrics (finance and trend monitoring) and in demand forecasting

However, general RNNs fail to learn long term dependencies due to the gap between recent and older data. LSTMs, on the other hand, can solve this challenge: the inner cells of LSTMs can carry information unchanged through special structures called gates – input, forget, output. Through these cells, LSTMs can control what information to retain or erase.

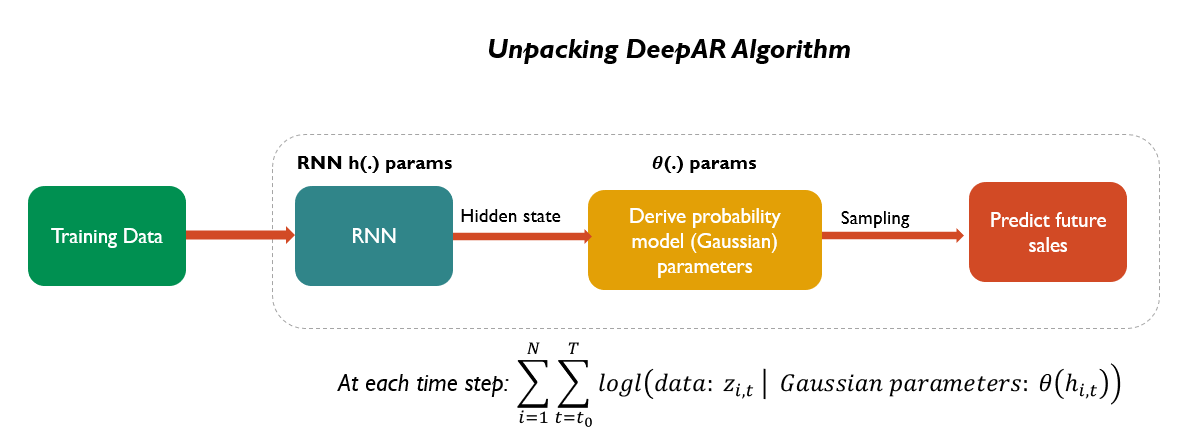
*Model Architecture*

The DeepAR algorithm employs LSTM network and probability models to identify non-linear structures in time series data and provide probabilistic estimates of forecasts.

*Optimize RNN parameters* *Hidden state* *Optimize probability model parameters*

*Recurrent Neural Network (LSTM) 🡪 Probability Model (for real values)* 🡪 *Forecasts*

The model is auto regressive in that it consumes observations from the last time step as input. It is also recurrent since uses previous output of the network as input at the next time step. During training phase, the hidden or the encoded state of the network, at each time step, is computed based on current covariates, previous observation and previous network output. The hidden state is then used to compute parameters for a probability model that characterizes the behavior of time series (product demand, for example). In other words, we assume demand to be random variable following a specific probability distribution. Once we have the probability model that can be defined through a set of parameters (say, mean and variance), it can be used to estimate forecasts. DeepAR used Adam optimizer, a Stochastic Gradient Descent algorithm, to optimize maximum log likelihood of training data given gaussian model parameters. Using this approach, we can derive (optimize) both probability model parameters and LSTM parameters to accurately estimate forecasts.



Training:

We assume that the DeepAR model distribution consists of a product of likelihood factors:

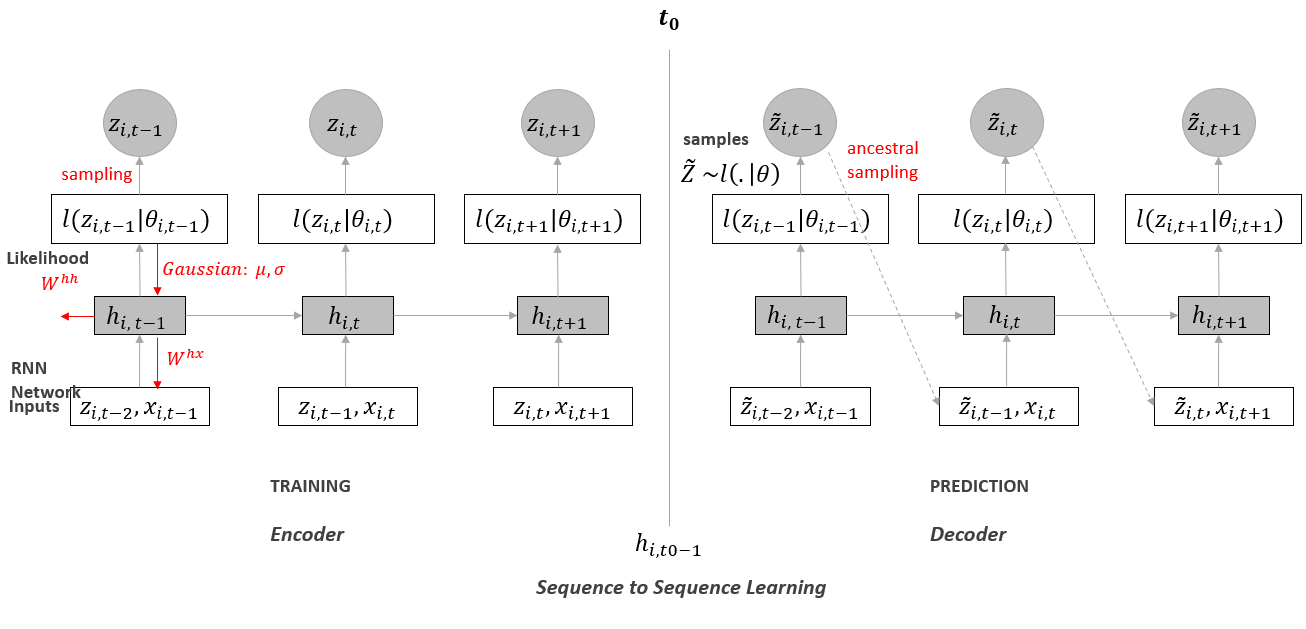
= . The likelihood function is parametrized by the output of an autoregressive recurrent network

Where h is a function implemented by Multi-layer recurrent neural network. The likelihood function is a fixed distribution whose parameters are given by a function , where is input to the function.

The time series or observations are fed to DeepAR as part of training. At each time step, current covariates, previous observation and previous network output are used. The model uses back propagation through time (BPTT) to compute gradient descent after each iteration. In particular, Adam Optimizer is used to conduct BPTT. Through the Stochastic Gradient descent algorithm, Adam, we arrive at optimal network weights via back propagation

At each time step t, the inputs to the network are covariates , the target at the previous time step , as well as the previous network output . The network output is then used to compute of the likelihood function . For prediction, the history of the time series is fed in for t < , then in the prediction range for t >= , a sample is drawn and fed back for the next point until the end of the prediction range t = . Repeating this process results in several traces

The sequential learning from observations in the training phase is passed on to prediction phase, , output of the initial encoder network



*Source: Probabilistic Forecasting with Autoregressive Recurrent Networks (*[*link*](https://arxiv.org/abs/1704.04110)*)*

Given the model parameter , we can obtain joint sample through ancestral sampling . Samples obtained this way can be used to compute quantiles of distribution of sum of values for some time range in the future.

*Likelihood model*

*What are Probability Models?* Rather than try to tease out the effects of various marketing, personal, and situational variables on sales demand, we embrace the notion of randomness and view the behavior of interest as the outcome of some probabilistic process.

Some of the applications of Probability models include -

* Predict behavior in future periods, be it in aggregate or at a more granular level
* Make inferences about the behavior, given summary measures
* Summarize and interpret patterns of market-level behavior

In our case, we view demand forecasting as a probabilistic process as opposed to a deterministic one

The likelihood function or probability model should be chosen based on the statistical properties of the time series data.

We consider Gaussian distribution for modeling real values (for example, weekly item sales in retail context). The log likelihood and its gradients w.r.t the parameters can be evaluated. We parametrize Gaussian by its mean ℝ+ and standard deviation parameter ℝ+

*Estimating model parameters*

We view the parameters of the distribution as latent traits for a given item/product category. As noted earlier, the Gaussian parameters are calculated via hidden state, , of the network at time step t

For Gaussian, *log likelihood (data z | parameters and )* = )

Softplus activation function, , is used to ensure that standard deviation of Gaussian distribution is greater than zero.

Once the parameters of Gaussian are calculated from the network output, the distribution can be used to generate forecast samples

*Key benefits of DeepAR model*

* Minimal manual feature engineering is needed to capture complex behavior – seasonal behavior and dependencies on given covariates is learned
* DeepAR makes probabilistic forecasts in the form of Monte Carlo samples that can be used to compute consistent quantile estimates for all sub-ranges in the prediction horizon.
* By learning from similar items, DeepAR can provide forecasts for items with little to no history at all; in here all single-item forecasting methods fail
* A wide range of likelihood functions are supported, allowing users to choose one that is appropriate for the statistical properties of the data - Because the data in the forecasting domain is very erratic, violating the basic assumptions of Gaussian errors (normal distribution of errors, stationarity, homoscedasticity of the time series), forecasting methods have included more suitable likelihood functions, such as zero inflated Poisson distribution and negative binomial distribution

All these benefits not only set DeepAR apart from traditional forecasting approaches, but also enable it to produce accurate forecast distributions learned from historical behavior of all the time series jointly, relative to other global methods. Also, probabilistic forecasts provide optimal decision under uncertainty as opposed to point estimates

**Model Sales through DeepAR**

As noted earlier, managing inventory is a complex activity to handle for retailers. Holidays, special-events and markdowns can have a significant impact on how a store performs and in turn how a department within a store performs.

The kaggle [dataset](https://www.kaggle.com/manjeetsingh/retaildataset) contains historical sales for 45 stores, with each store belonging to a specific type (location and performance) and size. The retailer runs several promotional markdowns throughout the year. These markdowns precede holidays, such as SuperBowl, Labor Day, Thanksgiving, and Christmas.

*Brief description about the dataset*

Features data:

Contains data related to the store, department, and regional activity for given dates

* Store - Numeric store ID for each store
* Date - Important dates for store
* Fuel price - Current fuel prices
* Markdowns - Markdowns are the discount you take on merchandise in your retail store from the original sale price marked.
* CPI - The Consumer Price Index (CPI) is a measure that examines the weighted average of prices of a basket of consumer goods and services, such as transportation, food and medical care.
* Unemployment - Current unemployment rate
* IsHoliday - Whether it's holiday or not on particular date

Sales data:

Historical sales data covering 3 years, from 2010 to 2012.

* Store - Numeric store ID for each store
* Dept - Numeric department ID for each department of store
* Date - Important dates for store
* Weekly Sales - Weekly sales to meassure sales performance of each store
* IsHoliday - Is it holiday or not on particular date

Store data:

Anonymized information about the 45 stores, including type and size of the store

* Store - Numeric store ID for each store
* Type - Type of store
* Size - Size of store

In the below section, we will look at input and output formats, including hyperparameters, of SageMaker DeepAR algorithm

* 2 input channels – the algorithm takes training and test JSONs as input through two channels

*The training JSON contains only 134 weeks of sales, while test JSON contains sales from all 143 weeks.*

**Training JSON**

{

Start: The starting date of weekly sales

Target: Weekly sales

Cat: Category or Department used to group sales

Dynamic\_feat: Dynamic features used to explain variation in sales. Beyond holidays, these features can include price, promotion and other covariates.

}

{"**start**":"2010-01-01 00:00:00","**target**":[19145.49, 17743.27, 14700.85, 20092.86, 17884.43, 19269.09, 22988.12, 17679.72, 16876.61, 14539.77, 16026.23, 14249.85, 15474.07, 22464.57, 19075.56, 20999.38, 18139.89, 13496.23, 15361.65, 16164.48, 15039.44, 14077.75, 16733.58, 16552.23, 17393.2, 16608.36, 21183.71, 16089.01, 18076.54, 19378.51, 15001.62, 14691.15, 19127.39, 17968.37, 20380.96, 29874.28, 19240.27, 17462.27, 17327.15, 16313.51, 20978.94, 28561.95, 19232.34, 20396.46, 21052.61, 30278.47, 47913.44, 17054.1, 15355.95, 15704.19, 15193.36, 14040.86, 13720.49, 17758.99, 24013.25, 24157.54, 22574.19, 12911.72, 20266.06, 18102.13, 21749.04, 22252.73, 21672.82, 15231.31, 16781.35, 14919.64, 15948.11, 17263.32, 16859.26, 13326.75, 17929.47, 15888.17, 13827.35, 16180.46, 22720.76, 15347.18, 15089.43, 14016.56, 17147.61, 14301.9, 16951.62, 16623.8, 19349.35, 24535.59, 18402.46, 19320.64, 20048.28, 14622.65, 19402.27, 19657.79, 18587.11, 20878.24, 19686.7, 23664.29, 20825.85, 27059.08, 15693.12, 29177.6, 45362.67, 20011.27, 13499.62, 15187.32, 16988.52, 14707.59, 20127.86, 23249.25, 20804.15, 19921.62, 16096.04, 18055.34, 17727.24, 16478.45, 16117.33, 15082.89, 15050.07, 17302.59, 20399.83, 17484.31, 14056.35, 16979.18, 17279.4, 14494.48, 14661.37, 13979.33, 13476.7, 18898.57, 13740.2, 15684.97, 15266.29, 16321.69, 15728.07, 17429.51, 17514.05, 20629.24], "**cat**":[15], "**dynamic\_feat**":[[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0]]}

**Test JSON**

{"**start**":"2010-01-01 00:00:00","**target**":[19145.49, 17743.27, 14700.85, 20092.86, 17884.43, 19269.09, 22988.12, 17679.72, 16876.61, 14539.77, 16026.23, 14249.85, 15474.07, 22464.57, 19075.56, 20999.38, 18139.89, 13496.23, 15361.65, 16164.48, 15039.44, 14077.75, 16733.58, 16552.23, 17393.2, 16608.36, 21183.71, 16089.01, 18076.54, 19378.51, 15001.62, 14691.15, 19127.39, 17968.37, 20380.96, 29874.28, 19240.27, 17462.27, 17327.15, 16313.51, 20978.94, 28561.95, 19232.34, 20396.46, 21052.61, 30278.47, 47913.44, 17054.1, 15355.95, 15704.19, 15193.36, 14040.86, 13720.49, 17758.99, 24013.25, 24157.54, 22574.19, 12911.72, 20266.06, 18102.13, 21749.04, 22252.73, 21672.82, 15231.31, 16781.35, 14919.64, 15948.11, 17263.32, 16859.26, 13326.75, 17929.47, 15888.17, 13827.35, 16180.46, 22720.76, 15347.18, 15089.43, 14016.56, 17147.61, 14301.9, 16951.62, 16623.8, 19349.35, 24535.59, 18402.46, 19320.64, 20048.28, 14622.65, 19402.27, 19657.79, 18587.11, 20878.24, 19686.7, 23664.29, 20825.85, 27059.08, 15693.12, 29177.6, 45362.67, 20011.27, 13499.62, 15187.32, 16988.52, 14707.59, 20127.86, 23249.25, 20804.15, 19921.62, 16096.04, 18055.34, 17727.24, 16478.45, 16117.33, 15082.89, 15050.07, 17302.59, 20399.83, 17484.31, 14056.35, 16979.18, 17279.4, 14494.48, 14661.37, 13979.33, 13476.7, 18898.57, 13740.2, 15684.97, 15266.29, 16321.69, 15728.07, 17429.51, 17514.05, 20629.24, 17730.73, 18966.48, 20781.46, 22979.73, 16402.34, 20037.44, 18535.65, 16809.01, 19275.43], "**cat**":[15], "**dynamic\_feat**":[[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0]]}

* Hyperparameters – DeepAR supports a range of hyperparameters. See below for some of the key hyperparameters. For detailed list, check out amazon docs [here](https://docs.aws.amazon.com/sagemaker/latest/dg/deepar_hyperparameters.html).

**Time frequency:** indicate whether the time series is hourly, weekly, monthly, or yearly

**Context length:** How many time steps in the past should the algorithm look at for training

**Prediction length:** The number of data points to predict

**Number of cells:** The number of neurons to use in each of the hidden layers

**Number of layers:** The number of hidden layers

**Likelihood function:** We will choose gaussian model, since weekly sales are real values

**epochs:** Maximum number of passes over the training data

**Mini batch size:** size of the mini batches used during training

**Learning rate:** the pace at which the loss is optimized

**Dropout rate:** for each epoch, what % of hidden neurons are not updated

**Early stopping patience:** training stops after a designated number of unsuccessful epochs, those in which the loss doesn’t improve

* Inference – For a given department, we send 134 weeks of sales, along with the department category and holiday flag across all the weeks. Below is the JSON output from the model endpoint:

{

"predictions": [

{

"quantiles": {

"0.9": [...],

"0.5": [...]

},

"samples": [...],

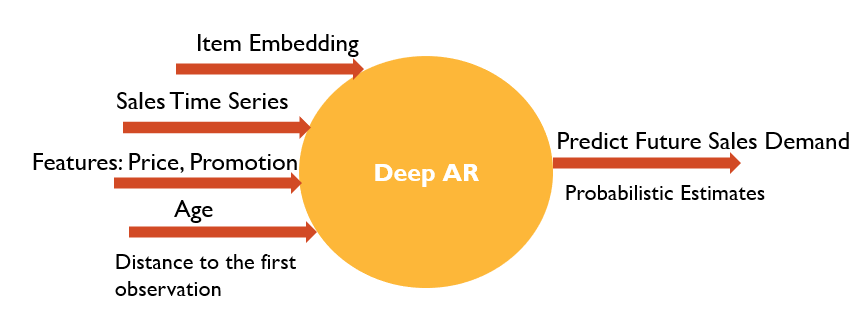
"mean": [...]

}

]

}

Beyond item grouping, sales time series, and dynamic features, DeepAR captures distance to first observation for new items or products. Because the algorithm learns item demand across multiple time series, it is able to estimate demand even for newly introduced items -- the length of weekly sales across all-time series need not remain the same. Additionally, the algorithm can also handle missing values, with missing values replaced with “Nan”.



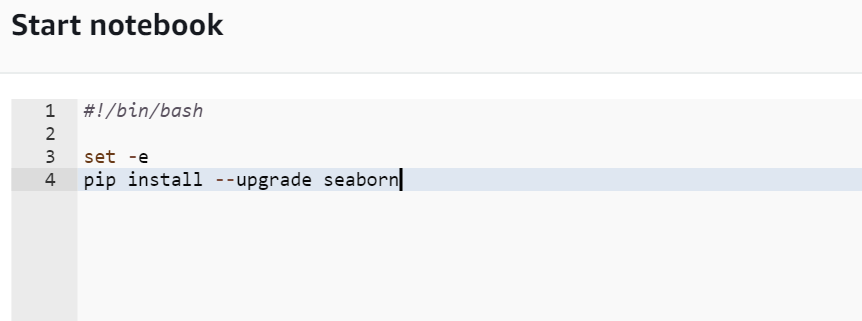
*Exploratory Data Analysis*

Although there are 45 stores, we will select one store, store number 20, to analyze performance across different departments across three years. The main idea here is that using DeepAR, we can learn the sales of items across different departments.

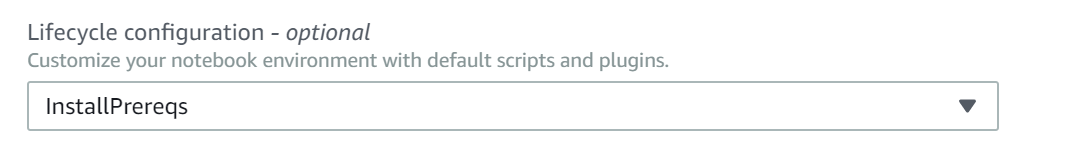
In SageMaker, through Lifecyle Configurations, we can custom install python packages before notebook instances are started. This eliminates the need to manually track packages required before the notebooks are executed.

For exploring the retail sales data, we will need latest version, 0.9.0, of seaborn installed.

In SageMaker, under Notebook, click on Lifecycle Configurations. Under Start notebook, enter the command to upgrade seaborn python package.



Edit Notebook settings by clicking on notebook instance, selecting *Actions*, and picking *Update Settings*. Under *Update Settings 🡪 Lifecycle configuration* section, select the name of the newly created Lifecyle Configuration. This option enables SageMaker to manage all python pre-requisites before the notebook instances are made available



Let’s merge the data across Sales, Store, and Features csv files:

import numpy #library to compute linear algebraic equations

import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns #is a Python statistical data visualization lib based on matplotlib

import warnings

warnings.simplefilter(action='ignore')

warnings.simplefilter(action='ignore', category=FutureWarning)

features = pd.read\_csv('Features data set.csv')

sales = pd.read\_csv('sales data-set.csv')

stores = pd.read\_csv('stores data-set.csv')

features.shape #There are 8,190 store, date and holiday combinations

sales.shape #There are 421,570 sales transactions

stores.shape #There are 45 stores in question

merged\_df = features.merge(sales, on=['Store', 'Date', 'IsHoliday']).merge(stores, on=['Store'])

merged\_df.head()

Convert IsHoliday to numerical form; convert Date field to pandas Date format

merged\_df = features.merge(sales, on=['Store', 'Date', 'IsHoliday']).merge(stores, on=['Store'])

merged\_df.head()

Write merged dataset to csv

merged\_df.to\_csv('retailsales.csv')

Now, let’s look at the distribution of each of the key factors that may impact sales

#Create a figure and a set of subplots

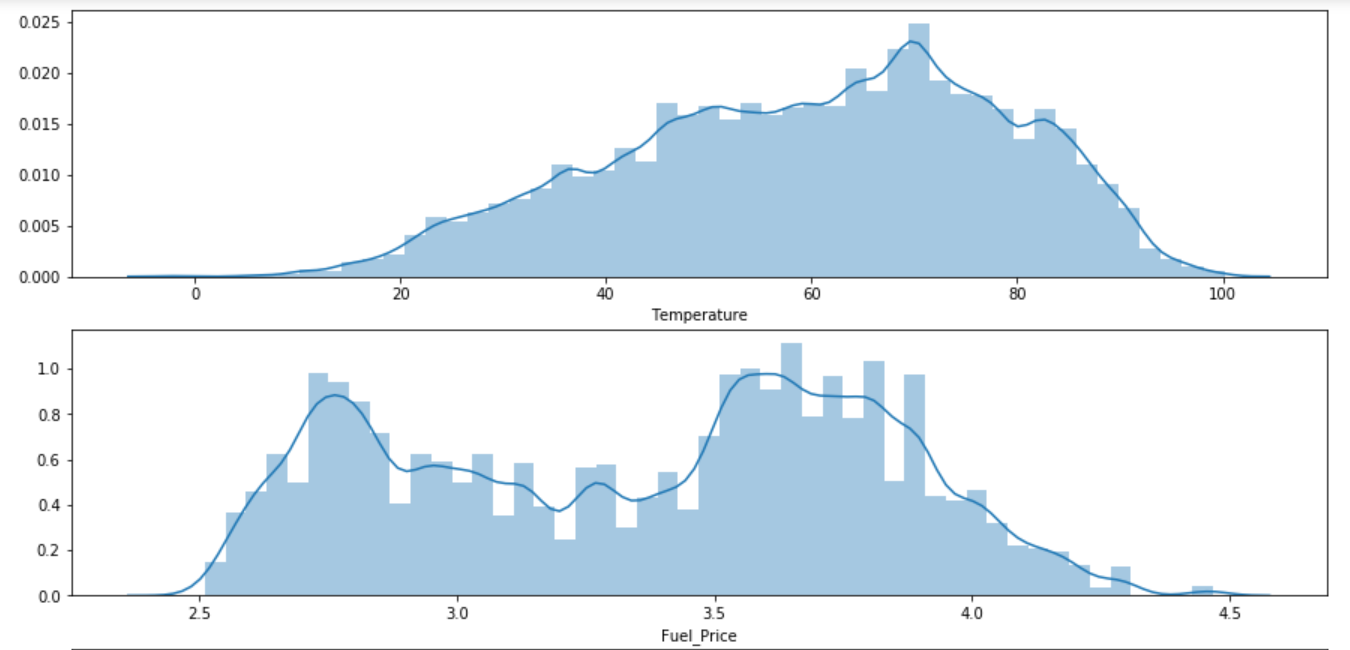
f, ax = plt.subplots(4, figsize=(15, 15)) #f=figure; ax=axes

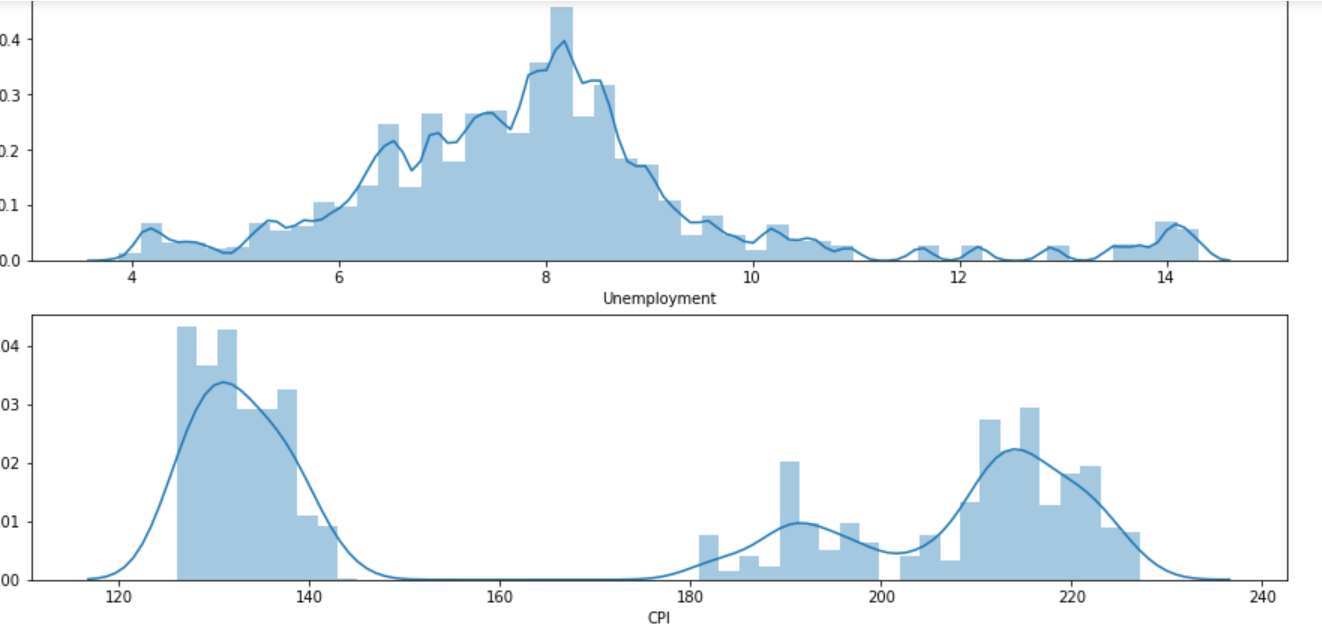
sns.distplot(merged\_df.Temperature, ax=ax[0])

sns.distplot(merged\_df.Fuel\_Price, ax=ax[1])

sns.distplot(merged\_df.Unemployment, ax=ax[2])

sns.distplot(merged\_df.CPI, ax=ax[3])





Let’s see how each of the features are correlated to each other. First, let’s look at the scatter plot between Sales (target) and holidays, temperature, Consumer Price Index, Unemployment, Store Type

f, ax = plt.subplots(6, figsize=(20,20))

#Fuel price between $3.25 and $3.75 is generating higher weekly sales

sns.scatterplot(x="Fuel\_Price", y="Weekly\_Sales", data=merged\_df, ax=ax[0])

#Temperature between 50 and 65 degrees is generating higher weekly sales

sns.scatterplot(x="Temperature", y="Weekly\_Sales", data=merged\_df, ax=ax[1])

#Holiday sales are higher than non-holiday sales

sns.scatterplot(x="IsHoliday", y="Weekly\_Sales", data=merged\_df, ax=ax[2])

#No material impact of CPI on Weekly Sales

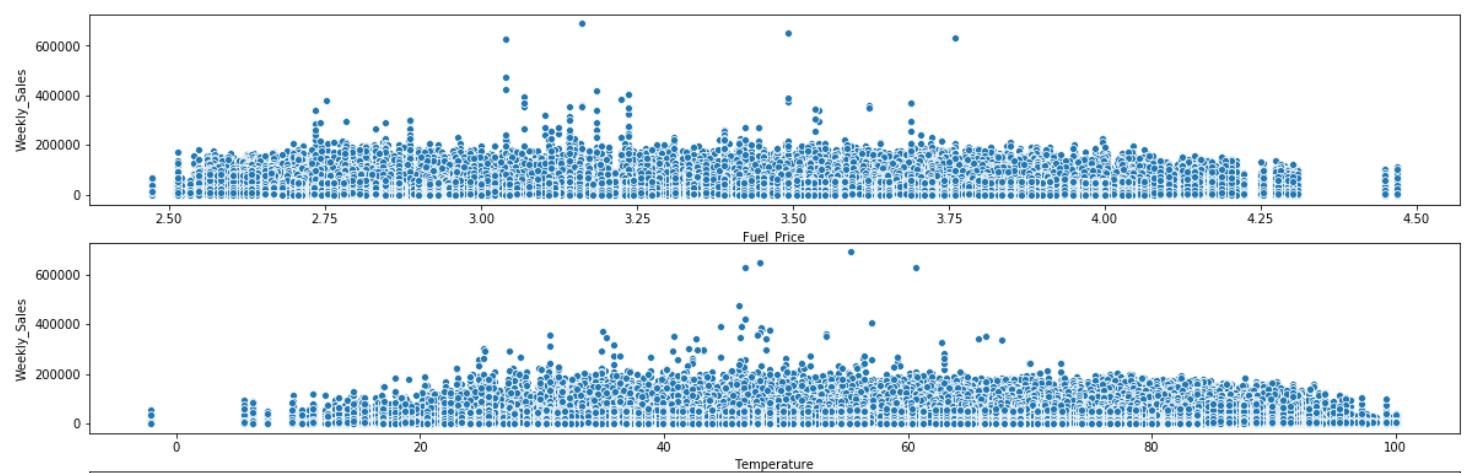
sns.scatterplot(x="CPI", y="Weekly\_Sales", data=merged\_df, ax=ax[3])

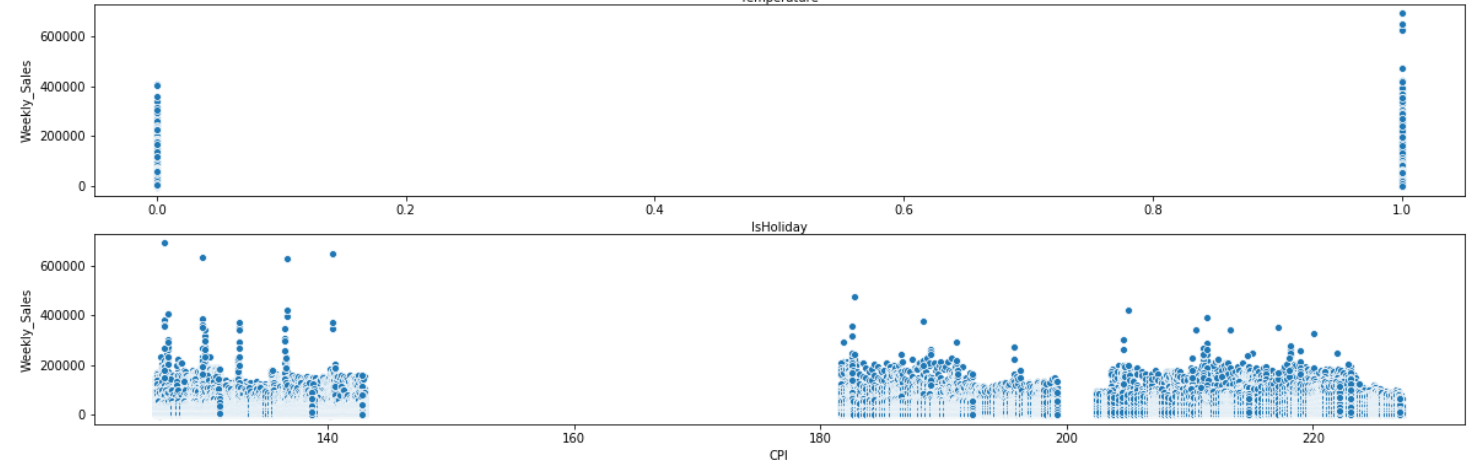
#Weekly sales are higher when unemployment rate is lower (7 to 8.5)

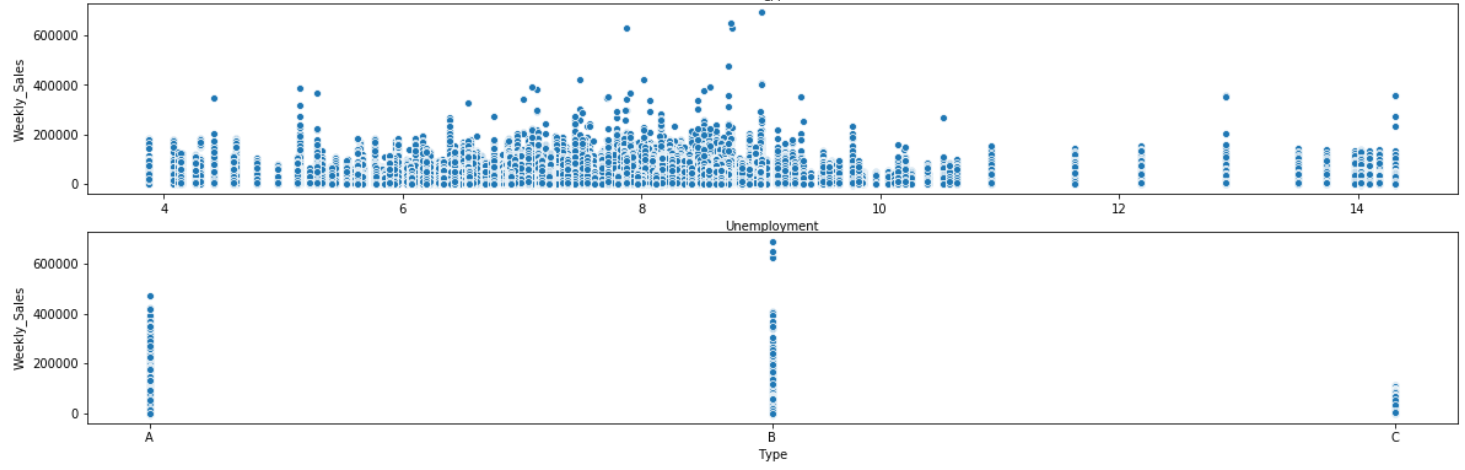
sns.scatterplot(x="Unemployment", y="Weekly\_Sales", data=merged\_df, ax=ax[4])

# B type stores have higher weekly sales

sns.scatterplot(x="Type", y="Weekly\_Sales", data=merged\_df, ax=ax[5])





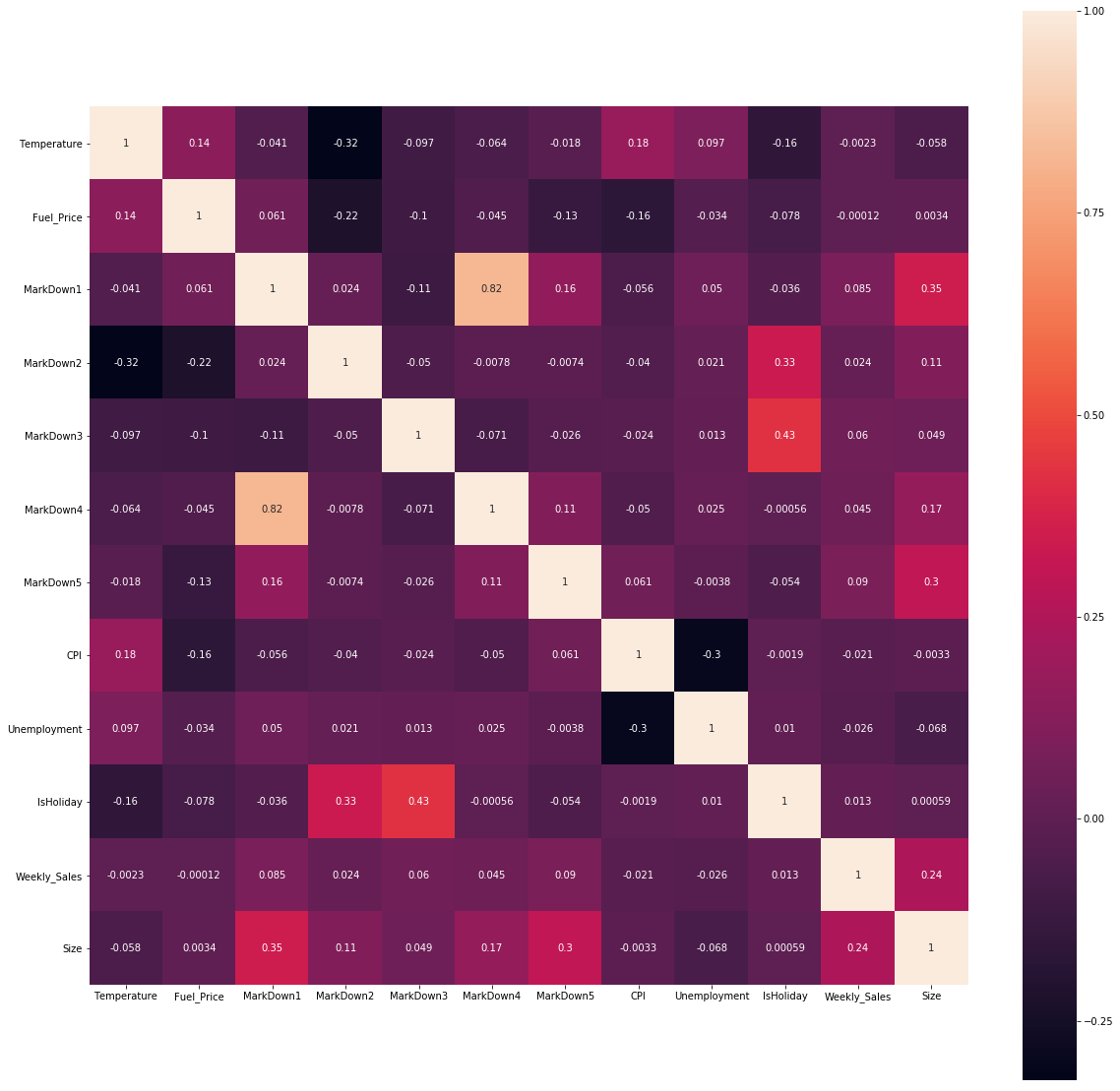


Second, let’s look at heatmap across all the features to identify what features impact sales

#Use seaborn's heatmap

f, ax = plt.subplots(figsize=(20, 20))

sns.heatmap(merged\_df.drop(['Store', 'Dept'], axis=1).corr(), square=True, annot=True) #heatmap is drawn along the columns



From the scatter plot and heat map, it is apparent that –

* Holidays are when markdowns are happening
* Sales are higher during holidays
* Type B stores generate higher sales
* Lower fuel prices (between $3 and $3.75) generate higher sales
* Ideal temperature (between 50 and 65 degrees) generate higher sales

For our further modeling, we will pick the best performing store, store 20, to model sales across different departments and years. For each of the time steps in the time series, we will also pass whether a particular day was observed as holiday or not.

Let’s begin with preparing the dataset for modeling:

* Create module named ‘retailsales.py’ to create JSON files that DeepAR can consume for training and validation
* Create module named ‘salesinference.py’ to build inference data, retrieve and plot predictions

In the *retailsales* module we read store #20 sales, sort sales in ascending order by date, and group them by department. We will then filter departments that have exactly 143 weekly sales. This step is to ensure that there are no missing values in the data. Although DeepAR can handle missing values, we’ve tried to make the problem less complex by only considering departments that have sales in almost all the weeks. We will then use department numbers to group or categorize the time series for the DeepAR algorithm. This grouping will be used by DeepAR to make demand predictions in future by department.

import pandas as pd

import numpy as np

import json

import io

import random

def prepareSalesData(csvfile):

#Read store 20 sales

store20\_sales = pd.read\_csv(csvfile, index\_col=None)

# Create Year column for grouping data

store20\_sales['Date'] = pd.to\_datetime(store20\_sales['Date'])

store20\_sales['Year'] = store20\_sales['Date'].dt.year

#Sort weekly sales by department

store20\_sales = store20\_sales.sort\_values(['Date', 'Dept'], ascending=True).reset\_index(drop=True)

#Select columns of interest

store20\_mod\_sales = store20\_sales[['Year', 'Date', 'Weekly\_Sales', 'Dept', 'IsHoliday']]

#Select departments with 143 weekly sales

store20\_mod\_sales = store20\_mod\_sales.groupby(store20\_mod\_sales.Dept, as\_index=True).filter(lambda x: len(x['Weekly\_Sales']) > 142)

#Map department numbers to categorical variables

dept\_list = store20\_mod\_sales['Dept'].unique()

cat\_values = [i for i in range(0, len(dept\_list))]

df\_dept = pd.DataFrame(dept\_list, index=cat\_values, columns=['Dept'])

df\_dept['cat']=df\_dept.index

store20\_mod\_sales = store20\_mod\_sales.merge(df\_dept, on=['Dept'])

return store20\_mod\_sales

We will then write a function, getTestSales, to create JSON for *testing* dataset. We will first create a dictionary for each department, with department ID being the key and sorted sales being the values.

The function, getTrainSales, on the other hand, is a subset of *testing* dataset. For each of the departments, we will chop the last few weekly sales determined by *prediction length*. The performance of DeepAR is measured by comparing demand predictions during the *prediction length* period with the actual demand.

writeSales function takes sales dictionary as input and write a JSON line for each department.

*Create JSON formatted input data for training*

The JSON line comprises of:

* start date for the time series
* target: sorted time series
* cat: also known as category to group time series
* dynamic\_feat: as known as dynamic features is used to add features, such as holidays, that can impact sales

def getTestSales(prepdsales, test\_key):

#There a total of 66 departments, with 143 weekly sales

#Create a dictionary of weekly sales for a given year dept

testSet = dict(list(prepdsales.groupby('Dept', as\_index=True)))

#Write test json to the current directory

writeSales(test\_key, testSet)

return testSet

def getTrainSales(prepdsales, train\_key, prediction\_length):

testSet = dict(list(prepdsales.groupby('Dept', as\_index=True)))

trainingSet = dict()

for dept in testSet.keys():

trainingSet[dept] = testSet[dept][:-prediction\_length]

writeSales(train\_key, trainingSet)

return trainingSet

def writeSales(filename, data):

#data is dictionary

file=open(filename, 'w')

keys=list(data.keys())

random.shuffle(keys)

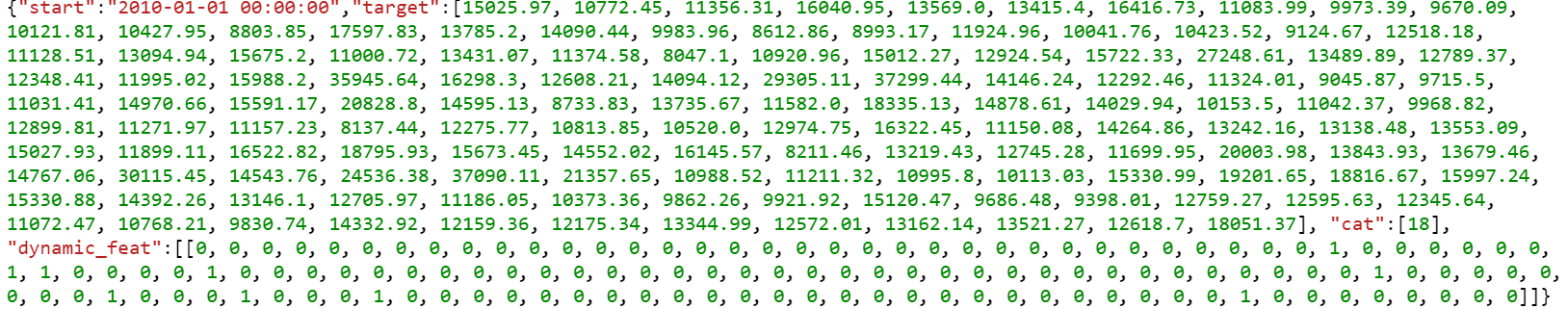
for dept in keys:

#On JSON sample per line; we've 66 samples in total (trainingSet)

line = "\"start\":\"2010-01-01 00:00:00\",\"target\":{}, \"cat\":{}, \"dynamic\_feat\":[{}]".format(data[dept]['Weekly\_Sales'].tolist(), data[dept]['cat'].unique().tolist(), data[dept]['IsHoliday'].tolist())

file.write('{'+line+'}\n')

In the *salesinference* module, we will write a function to build prediction data. The goal here is send the time series from training set to DeepAR, so it can estimate sales in the last 9 weeks, where 9 is the prediction length.



import io

import boto3

import random

import json

import matplotlib.pyplot as plt

import numpy as np

q1 = '0.1 '

q2 = '0.9'

num\_samples = 100 # predict 100 sample series

def buildInferenceData(dept, trainSet, testSet):

#dept\_sales = data[dept]

holidays = []

holidays.append(testSet[dept]['IsHoliday'].tolist())

s = {"start": "2010-01-01 00:00:00", "target": trainSet[dept]['Weekly\_Sales'].tolist(), "cat": trainSet[dept]['cat'].unique().tolist(), "dynamic\_feat": holidays}

series = []

series.append(s)

configuration = {

"output\_types": ["mean", "quantiles", "samples"],

"num\_samples": num\_samples,

"quantiles": [q1, q2]

}

http\_data = {

"instances": series,

"configuration": configuration

}

return json.dumps(http\_data)

***Upload input data to S3***

In the function below, getInferenceSeries, we will parse the JSON results from the DeepAR algorithm to identify mean sales, sales in 10 percentile, and sales in 90 percentile. Note that, using Gaussian, DeepAR generates 100 samples of weekly sales for the next 9 weeks. Therefore, sales in 10 percentile and 90 percentile indicate lower and upper bounds of weekly sales during the prediction period.

*DeepAR JSON Response Format*

{

"predictions": [

{

"quantiles": {

"0.9": [...],

"0.5": [...]

},

"samples": [...],

"mean": [...]

}

]

}

def getInferenceSeries(result):

json\_result = json.loads(result)

y\_data = json\_result['predictions'][0]

y\_mean = y\_data['mean']

y\_q1 = y\_data['quantiles'][q1]

y\_q2 = y\_data['quantiles'][q2]

y\_sample = y\_data['samples'][random.randint(0, num\_samples)]

return y\_mean, y\_q1, y\_q2, y\_sample

The next function, plotResults, diagrams the performance of DeepAR relative to actual sales during the prediction period. For each of the 9 weeks, we will look at mean sales, ground truth sales, sample sales, 10 percentile sales and 90 percentile sales.

def plotResults(prediction\_length, result, truth=False, truth\_data=None, truth\_label=None):

x = range(0, prediction\_length)

y\_mean, y\_q1, y\_q2, y\_sample = getInferenceSeries(result)

plt.gcf().clear() #clear the current plot

mean\_label, = plt.plot(x, y\_mean, label='mean')

q1\_label, = plt.plot(x, y\_q1, label=q1)

q2\_label, = plt.plot(x, y\_q2, label=q2)

sample\_label, = plt.plot(x, y\_sample, label='sample')

if truth:

ground\_truth, = plt.plot(x, truth\_data, label=truth\_label)

plt.legend(handles=[ground\_truth, q2\_label, mean\_label, q1\_label, sample\_label])

else:

plt.legend(handles=[q2\_label, mean\_label, q1\_label, sample\_label])

plt.yticks(np.arange(0.22, 2.0, 0.1)) #tick marks on y axis, .22 to 2, with 0.1 steps

plt.show()

*Refactoring Code*

To modularize the code to test DeepAR, we will package the two modules, retailsales, salesinference.

To package the modules, we will create \_\_init\_\_.py file to import the modules. We will then create setup.py, detailing the pre-requisite packages to be installed

DeepAR project structure.

Project Organization

------------

├── notebooks/ <- All notebooks are residing here.

├── data/ <- Input data is residing here.

├── deepar/ <- Python package with source code of this project.

├──retailsales.py <- Creating training and testing datasets for DeepAR.

├──salesinference.py <- Preparing data for predictions, obtaining and plotting predictions from DeepAR

├── README.md <- The top-level README for developers using this project.

├── setup.py <- Defines pre-requisite packages to install and distribute package.

*setup.py*

import os

from setuptools import setup, find\_packages

def read(fname):

return open(os.path.join(os.path.dirname(\_\_file\_\_), fname)).read()

setup(

name="deepar",

description="DeepAR project structure.",

author="<your-name>",

packages=find\_packages(exclude=['data', 'figures', 'output', 'notebooks']),

long\_description=read('README.md'),

)

*\_\_init\_\_.py*

from . import retailsales

from . import salesinference

We will now install the package for the modules to be available while training DeepAR.

#Navidate to deep-ar directory to install the deepar package containing commonly used functions

path = ".."

os.chdir(path)

#install predefined functions

!pip install .

#Navigate to the parent directory to train the DeepAR model

# org\_path = ".."

# os.chdir(org\_path)

!pwd

*Bringing it all together*

### *Prepare training and test dataset in JSON format:*

Create json lines to build training and testing dataset. Also, return dictionaries of the training and testing datasets for model training.

import deepar as da

train\_key = 'deepar\_sales\_training.json'

test\_key = 'deepar\_sales\_test.json'

#Prediction and context length for training the DeepAR model

prediction\_length = 9

salesfn = 'data/store20\_sales.csv'

salesdf = da.retailsales.prepareSalesData(salesfn)

testSet = da.retailsales.getTestSales(salesdf, test\_key)

trainingSet = da.retailsales.getTrainSales(salesdf, train\_key, prediction\_length)

### *Upload Input data to S3*

The newly created json files are uploaded to the designated s3 bucket via upload\_data function from Sagemaker session object (Sagemaker Python SDK)

bucket = 'ai-in-aws'

prefix = 'sagemaker/deepar-weekly-sales'

train\_prefix = '{}/{}'.format(prefix, 'train')

test\_prefix = '{}/{}'.format(prefix, 'test')

output\_prefix = '{}/{}'.format(prefix, 'output')

sagemaker\_session = sagemaker.Session()

train\_path = sagemaker\_session.upload\_data(train\_key, bucket=bucket, key\_prefix=train\_prefix)

test\_path = sagemaker\_session.upload\_data(test\_key, bucket=bucket, key\_prefix=test\_prefix)

### *Configuring the training job*

The get\_image\_uri function from SageMaker estimator object is used to obtain uri of the deepar docker image. Once the uri is obtained, deepAR estimator is created. The constructor parameters include docker image uri, execution role, training instance type and count, outpath path to save the trained algorithm and SageMaker session.

role = get\_execution\_role()

output\_path = r's3://{0}/{1}'.format(bucket, output\_prefix)

container = get\_image\_uri(boto3.Session().region\_name, 'forecasting-deepar')

deepAR = sagemaker.estimator.Estimator(container,

role,

train\_instance\_count=1,

train\_instance\_type='ml.c4.xlarge',

output\_path=output\_path,

sagemaker\_session=sagemaker\_session)

*Defining Hyperparameters*

The following hyperparameters are define to characterize the time series, RNN architecture, number of training rounds, learning rate and drop-out rate for regularization.

hyperparameters = {

"time\_freq": 'W', # weekly series

"context\_length": prediction\_length, # how many data points are we going to look at before predicting

"prediction\_length": prediction\_length, # number of data points to predict

"num\_cells": "40", # of cells to use in each of the hidden layers

"num\_layers": "2", # of hidden layers

"likelihood": "gaussian",

"epochs": "300", # max number of passses over the training data

"mini\_batch\_size": "32", # size of the mini batches used during training

"learning\_rate": "0.00001",

"dropout\_rate": "0.05", #for each iteration, a random subset of hidden neurons are not updated

"early\_stopping\_patience": "10" # stop if loss hasn't improved in 10 epochs

}

deepAR.set\_hyperparameters(\*\*hyperparameters) #\*\* = arbitrary number of arguments to functions

*Training the model*

Two channels, train and test jsons, are used to fit the DeepAR model

data\_channels = {"train": train\_path, "test": test\_path}

deepAR.fit(inputs=data\_channels)

[12/05/2018 17:36:00 INFO 140654853965632] #test\_score (algo-1, RMSE): 7307.12501604

[12/05/2018 17:36:00 INFO 140654853965632] #test\_score (algo-1, mean\_wQuantileLoss): 0.198078

[12/05/2018 17:36:00 INFO 140654853965632] #test\_score (algo-1, wQuantileLoss[0.1]): 0.172473

[12/05/2018 17:36:00 INFO 140654853965632] #test\_score (algo-1, wQuantileLoss[0.2]): 0.236177

[12/05/2018 17:36:00 INFO 140654853965632] #test\_score (algo-1, wQuantileLoss[0.3]): 0.236742

[12/05/2018 17:36:00 INFO 140654853965632] #test\_score (algo-1, wQuantileLoss[0.4]): 0.190065

[12/05/2018 17:36:00 INFO 140654853965632] #test\_score (algo-1, wQuantileLoss[0.5]): 0.1485

[12/05/2018 17:36:00 INFO 140654853965632] #test\_score (algo-1, wQuantileLoss[0.6]): 0.178847

[12/05/2018 17:36:00 INFO 140654853965632] #test\_score (algo-1, wQuantileLoss[0.7]): 0.223082

[12/05/2018 17:36:00 INFO 140654853965632] #test\_score (algo-1, wQuantileLoss[0.8]): 0.226312

[12/05/2018 17:36:00 INFO 140654853965632] #test\_score (algo-1, wQuantileLoss[0.9]): 0.170508

[12/05/2018 17:36:00 INFO 140654853965632] #quality\_metric: host=algo-1, test RMSE <loss>=7307.12501604

[12/05/2018 17:36:00 INFO 140654853965632] #quality\_metric: host=algo-1, test mean\_wQuantileLoss <loss>=0.198078483343

#metrics {"Metrics": {"totaltime": {"count": 1, "max": 6360.483169555664, "sum": 6360.483169555664, "min": 6360.483169555664}, "setuptime": {"count": 1, "max": 8.634090423583984, "sum": 8.634090423583984, "min": 8.634090423583984}}, "EndTime": 1544031360.96456, "Dimensions": {"Host": "algo-1", "Operation": "training", "Algorithm": "AWS/DeepAR"}, "StartTime": 1544031360.954762}

**Predict and Evaluate Sales**

In this section, we will deploy and consume the model:

*Deploy the model*

The deploy function of deepAR estimator is used to host an endpoint. The number and type of hosting instances can be specified through parameters initial\_instance\_count and instance\_type.

deepAR\_predictor = deepAR.deploy(initial\_instance\_count=1, instance\_type='ml.m4.xlarge')

*Consume the model*

To assess the model performance, we use department #90. The buildInferencedata function is used to prepare the time series data in JSON format. This data is then sent to the endpoint hosted. The results returned from the endpoint are then plotted against actual sales. As can be seen clearly, the mean estimated sales are close to the actual sales, indicating that the DeepAR algorithm has adequately picked up sales demand across different departments. The department number can be changed to evaluate model performance across all departments. The probabilistic sales estimates thus enable us to estimate demand more accurately than point estimates.

#Predict last 9 weeks of a department and compare to ground truth

deepAR\_predictor.content\_type = 'application/json'

dept = 90

prediction\_data = da.salesinference.buildInferenceData(dept, trainingSet, testSet)

#print(prediction\_data)

result = deepAR\_predictor.predict(prediction\_data)

y\_mean, y\_q1, y\_q2, y\_sample = da.salesinference.getInferenceSeries(result)

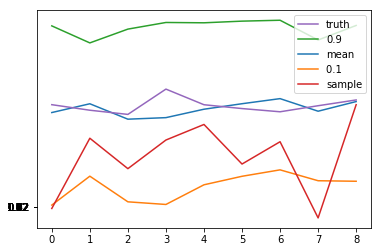
print("Predicted Sales: ", y\_mean)

print("Actual Sales: ", list(testSet[dept]['Weekly\_Sales'][134:]))

da.salesinference.plotResults(prediction\_length, result, truth=True, truth\_data=testSet[dept]['Weekly\_Sales'][134:], truth\_label='truth')

Predicted Sales: [92707.65625, 101316.90625, 86202.3984375, 87715.5625, 95967.359375, 101363.71875, 106354.90625, 94017.921875, 103476.71875]

Actual Sales: [100422.86, 94987.08, 90889.75, 115695.71, 100372.02, 96616.19, 93460.57, 99398.64, 105059.88]



**References**

<https://www.kaggle.com/iammhaseeb/exploratory-data-analysis-on-retail-data>

<https://medium.com/@julsimon/predicting-world-temperature-with-time-series-and-deepar-on-amazon-sagemaker-e371cf94ddb5>

<https://machinelearningmastery.com/exponential-smoothing-for-time-series-forecasting-in-python/>

<https://towardsdatascience.com/unboxing-arima-models-1dc09d2746f8>

<https://docs.aws.amazon.com/sagemaker/latest/dg/deepar.html>

<https://github.com/awslabs/amazon-sagemaker-examples/issues/384>

<https://docs.aws.amazon.com/sagemaker/latest/dg/deepar_how-it-works.html>

<https://docs.aws.amazon.com/sagemaker/latest/dg/deepar-in-formats.html>