Predictive Sales Model Documentation

Definition

A predictive sales model is a statistical or machine learning model that uses historical data and other relevant variables to forecast future sales. The model is trained on past sales data to identify patterns and trends that can be used to predict future sales with a certain level of accuracy. The model can take into account various factors that may affect sales, such as marketing campaigns, seasonality, economic indicators, customer behavior, and competitor activity. By analyzing these factors, the model can provide valuable insights and recommendations to help businesses optimize their sales strategies and improve their bottom line. Predictive sales models are commonly used in industries such as retail, e-commerce, manufacturing, and finance.

By referring to this resource created the following model that can be found here

Model creating and explanation

Load the required libs and dataset

```
import warnings
import itertools
import numpy as np
import matplotlib.pyplot as plt
warnings.filterwarnings("ignore")
plt.style.use('fivethirtyeight')
import pandas as pd
import statsmodels.api as sm
import matplotlib
matplotlib.rcParams['axes.labelsize'] = 14
matplotlib.rcParams['xtick.labelsize'] = 12
```

```
matplotlib.rcParams['ytick.labelsize'] = 12
matplotlib.rcParams['text.color'] = 'k'

df = pd.read_excel("Sample - Superstore.xls")
furniture = df.loc[df['Category'] == 'Furniture']
furniture['Order Date'].min(), furniture['Order Date'].max()
```

Data Preprocessing

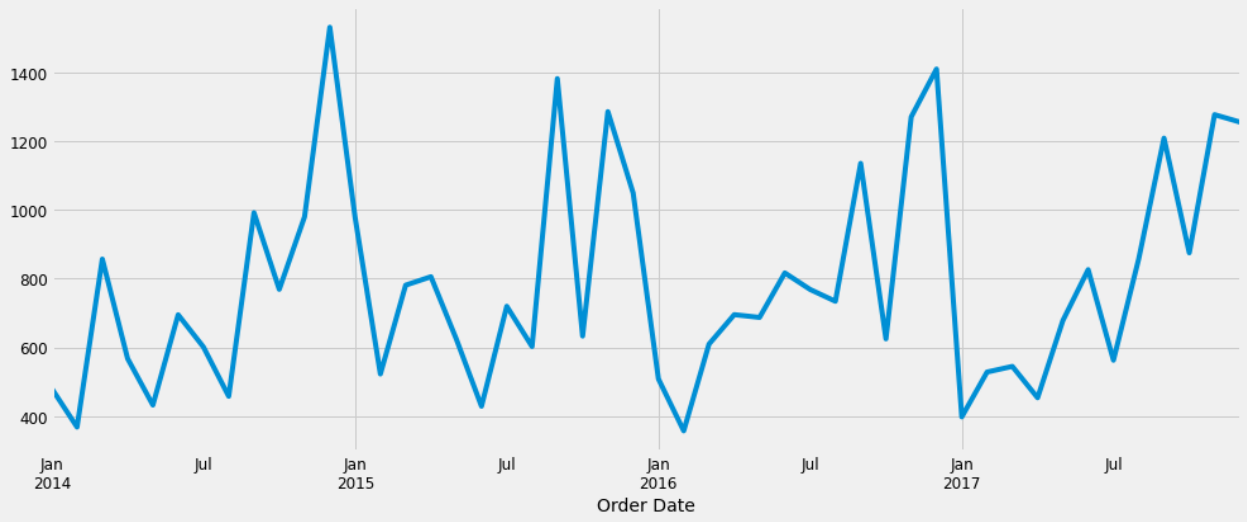
The process of preparing data involves various steps such as eliminating unnecessary columns, checking for missing values, aggregating sales by date, and so on.

```
cols = ['Row ID', 'Order ID', 'Ship Date', 'Ship Mode',
'Customer ID', 'Customer Name', 'Segment', 'Country', 'City',
'State', 'Postal Code', 'Region', 'Product ID', 'Category',
'Sub-Category', 'Product Name', 'Quantity', 'Discount',
'Profit']
furniture.drop(cols, axis=1, inplace=True)
furniture = furniture.sort_values('Order Date')
print(furniture.isnull().sum())
```

Indexing with Time Series Data

To make the datetime data more manageable, we will replace it with the average daily sales value for the corresponding month. To achieve this, we will use the beginning of each month as the timestamp.

Visualizing Furniture Sales Time Series Data



Decomposition Visualizition

After analyzing the sales data, we can observe certain noticeable patterns in the data when we plot it. These patterns are recurring and follow a specific trend over time, which is known as the seasonality pattern. For instance, we notice that sales are typically low at the beginning of the year and peak towards the end of the year. Additionally, within any given year, there is always an overall upward trend in sales, but a few months during the middle of the year experience low sales.
To gain a better understanding of these patterns, we can use a method called time-series decomposition, which allows us to break down our time-series data into three key components: trend, seasonality, and noise. By doing so, we can gain a deeper insight into the individual components that make up our sales data and analyze each one separately.

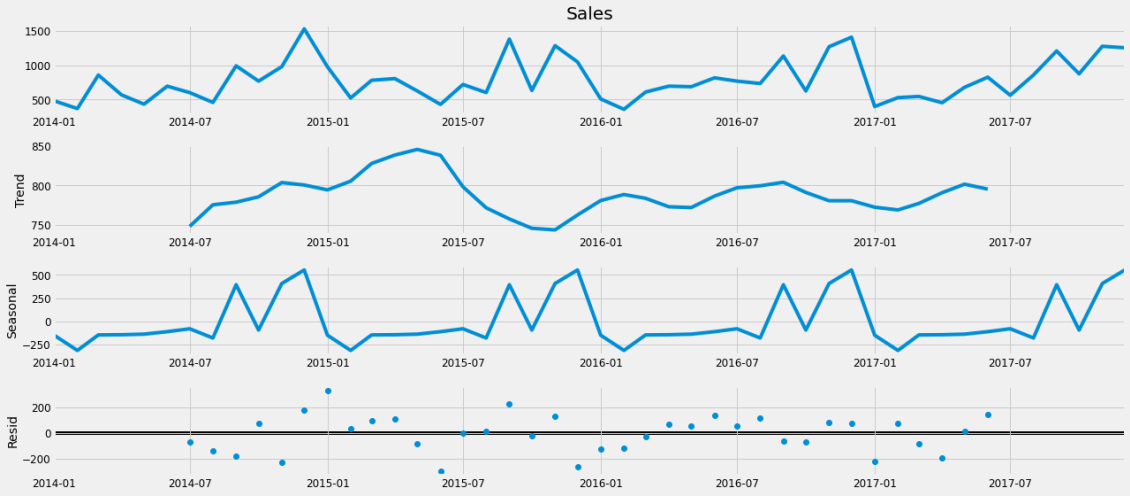
1. Trend: The trend is the long-term pattern of the time series, which shows the overall direction of the series. It represents the general tendency of the

series to increase, decrease, or remain constant over time. A trend can be either linear or nonlinear.

2. Seasonality: Seasonality refers to the pattern that repeats itself in fixed intervals of time, such as daily, weekly, monthly, or yearly. For example, sales of winter clothing tend to increase in the winter season and decrease in the summer season. Seasonality can be observed as a regular and repeated pattern within a year or a shorter period.
3. Noise: Noise refers to the random variation or irregularity in the data that cannot be explained by the trend or the seasonality. It represents the fluctuations that occur in the time series due to external factors or unexpected events. Noise can also be referred to as residuals or errors, and it is often present in most time series data.

By decomposing a time series into its trend, seasonality, and noise components, we can better understand the underlying patterns and behavior of the data. This can help us in making more accurate forecasts and identifying any anomalies or irregularities in the data.

```
from pylab import rcParams
rcParams['figure.figsize'] = 18, 8
decomposition = sm.tsa.seasonal_decompose(y, model='additive')
fig = decomposition.plot()
plt.show()
```



Time series forecasting with ARIMA

ARIMA models parameters are `ARIMA(p, d, q)`. These three parameters for seasonality, trend, and noise in data:

```
p = d = q = range(0, 2)
pdq = list(itertools.product(p, d, q))
seasonal_pdq = [(x[0], x[1], x[2], 12) for x in
list(itertools.product(p, d, q))]
print('Examples of parameter combinations for Seasonal
ARIMA...')
print('SARIMAX: {} x {}'.format(pdq[1], seasonal_pdq[1]))
print('SARIMAX: {} x {}'.format(pdq[1], seasonal_pdq[2]))
print('SARIMAX: {} x {}'.format(pdq[2], seasonal_pdq[3]))
print('SARIMAX: {} x {}'.format(pdq[2], seasonal_pdq[4]))
```

Time series forecasting with ARIMA (Autoregressive Integrated Moving Average) is a statistical method used to analyze and forecast time series data. ARIMA models are widely used for time series analysis and forecasting due to their ability to capture complex temporal patterns in the data.

ARIMA models involve three components:

Autoregression (AR): A model that uses lagged values of the variable being forecasted to predict future values. This component captures the trend of the time series.

Integrated (I): This component deals with the non-stationarity of the time series. It involves differencing the time series to make it stationary.

Moving Average (MA): A model that uses the errors (residuals) of the autoregressive model to predict future values. This component captures the noise (random fluctuations) in the time series.

By combining these three components, an ARIMA model can be used to capture the underlying structure of the time series and make predictions about future values. The process of fitting an ARIMA model involves identifying the appropriate values for the order of each component (i.e., the number of autoregressive terms, the number of differences, and the number of moving average terms) that will yield the best forecast accuracy.

grid search

In this step, we will perform parameter selection for our furniture sales ARIMA time series model. The aim is to use a "grid search" technique to identify the best combination of parameters that can enhance the performance of our model.

```
for param in pdq:
    for param_seasonal in seasonal_pdq:
        try:
            mod = sm.tsa.statespace.SARIMAX(y,
                                            order=param,

seasonal_order=param_seasonal,

enforce_stationarity=False,

enforce_invertibility=False)
results = mod.fit()
print('ARIMA{}x{}12 - AIC:{}'.format(param, param_seasonal,
results.aic))
        except:
            continue
```

After evaluating different combinations of parameters using the AIC value as a metric, we have identified SARIMAX(1, 1, 1)x(1, 1, 0, 12) as the configuration that gives us the lowest AIC value. Hence, we can consider this as the optimal option for our model.

AIC : stands for Akaike Information Criterion, which is a mathematical measure used to evaluate the goodness of fit of a statistical model. It takes into account the number of parameters in the model and the goodness of fit of the model to the data. The lower the AIC value, the better the model fits the data. AIC is commonly used in time series analysis and can be used to compare different models and select the best one for forecasting.

Fitting the best model

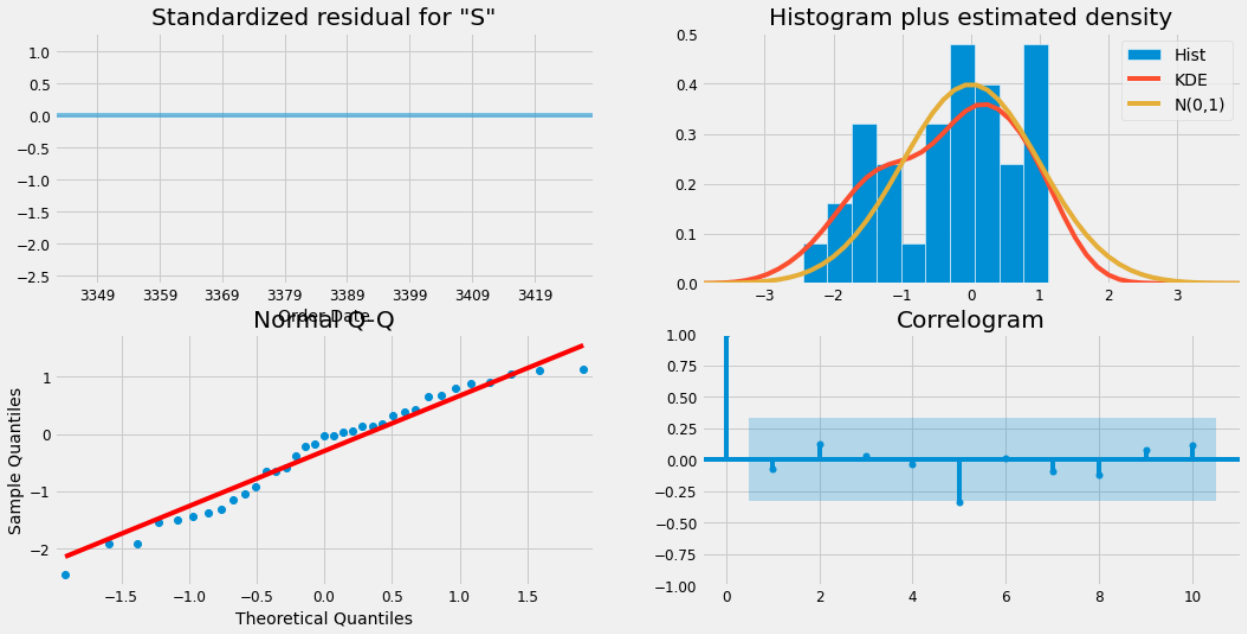
```
mod = sm.tsa.statespace.SARIMAX(y,
                                order=(1, 1, 1),
                                seasonal_order=(1, 1, 0, 12),
                                enforce_stationarity=False,
                                enforce_invertibility=False)
results = mod.fit()
print(results.summary().tables[1])
```

Invsestiget the model

To ensure that our model is working correctly, we need to perform model diagnostics and investigate any unusual behavior. By running the following code, we can generate diagnostic plots:

```
results.plot_diagnostics(figsize=(16, 8))
plt.show()
```

The figure shows the resulting diagnostic plots. Although the model is not perfect, the diagnostic plots suggest that the model residuals are nearly normally distributed, which is a good sign.



The plot_diagnostics function in the statsmodels library creates four graphs that help in diagnosing the behavior of the ARIMA model. Here is an explanation of each graph and what to focus on:

1. Histogram plus estimated density plot: This graph shows the histogram of the residuals along with the estimated probability density function of the normal distribution. The main focus of this plot is to check whether the residuals are normally distributed or not. If the residuals follow a normal distribution, then the estimated density plot will closely follow the historam.

2. Q-Q plot: The Q-Q plot is a graphical technique to compare the distribution of a sample of data to a normal distribution. If the residuals are normally distributed, the Q-Q plot will show a straight line. Any deviation from a straight line indicates non-normality.

3. Correlogram: This graph shows the correlation of the residuals over time, with lag on the x-axis and the correlation coefficient on the y-axis. It helps in identifying any autocorrelation in the residuals. Autocorrelation is the correlation of a signal with a delayed copy of itself. If there is significant autocorrelation in the residuals, then the model is not capturing all the information in the data.

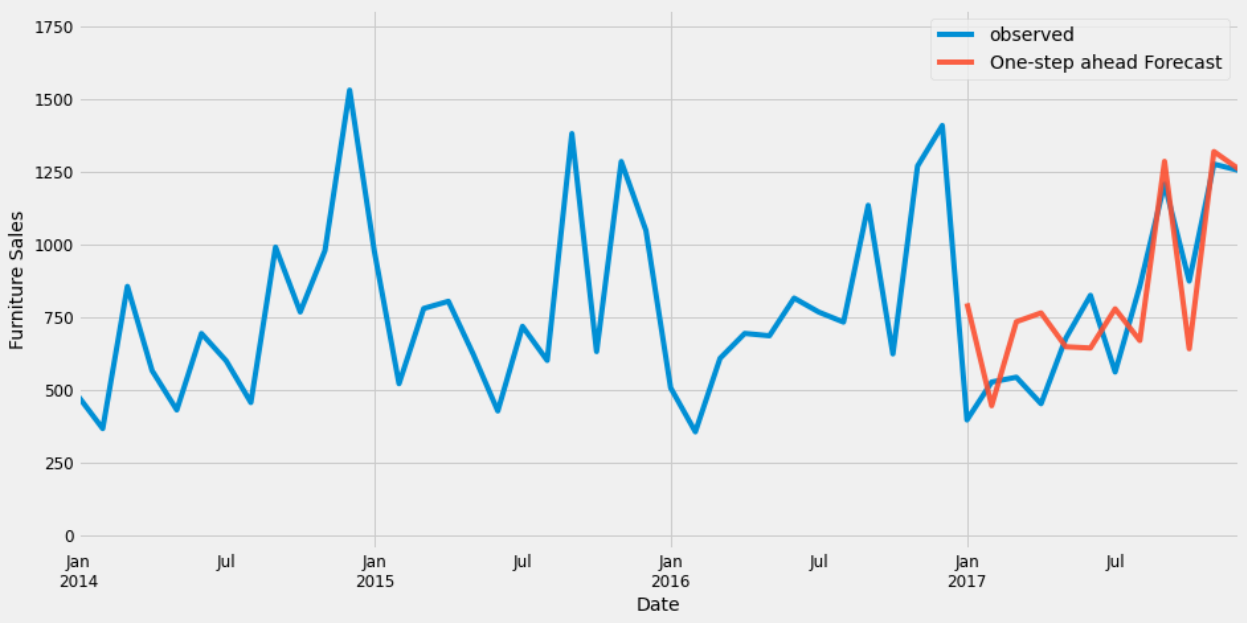
4. Partial Autocorrelation Function (PACF) plot: This plot shows the correlation between the residuals at different lags, while controlling for the correlations at shorter lags. It helps in identifying the order of the ARIMA model. If there are significant spikes beyond the confidence interval in the PACF plot, then it suggests that the residuals are not random and may require further modeling.

In summary, the main focus of the four graphs is to diagnose the behavior of the residuals. If the residuals are normally distributed and do not show any significant autocorrelation or partial autocorrelation beyond the confidence intervals, then the ARIMA model can be considered a good fit for the data. However, if the diagnostics indicate non-normality or significant autocorrelation, further modeling or modifications may be necessary.

Validating forecasts

```
pred = results.get_prediction(start=pd.to_datetime('2017-01-
01'), dynamic=False)
pred_ci = pred.conf_int()
ax = y['2014':].plot(label='observed')
pred.predicted_mean.plot(ax=ax, label='One-step ahead Forecast',
alpha=.7, figsize=(14, 7))
ax.fill_between(pred_ci.index,
                pred_ci.iloc[:, 0],
                pred_ci.iloc[:, 1], color='k', alpha=.2)
ax.set_xlabel('Date')
ax.set_ylabel('Furniture Sales')
plt.legend()
plt.show()
```

In order to evaluate the precision of our predictions, we assess the forecasted sales against the actual sales data of the time series, beginning from 2017-01-01 until the end of the dataset.



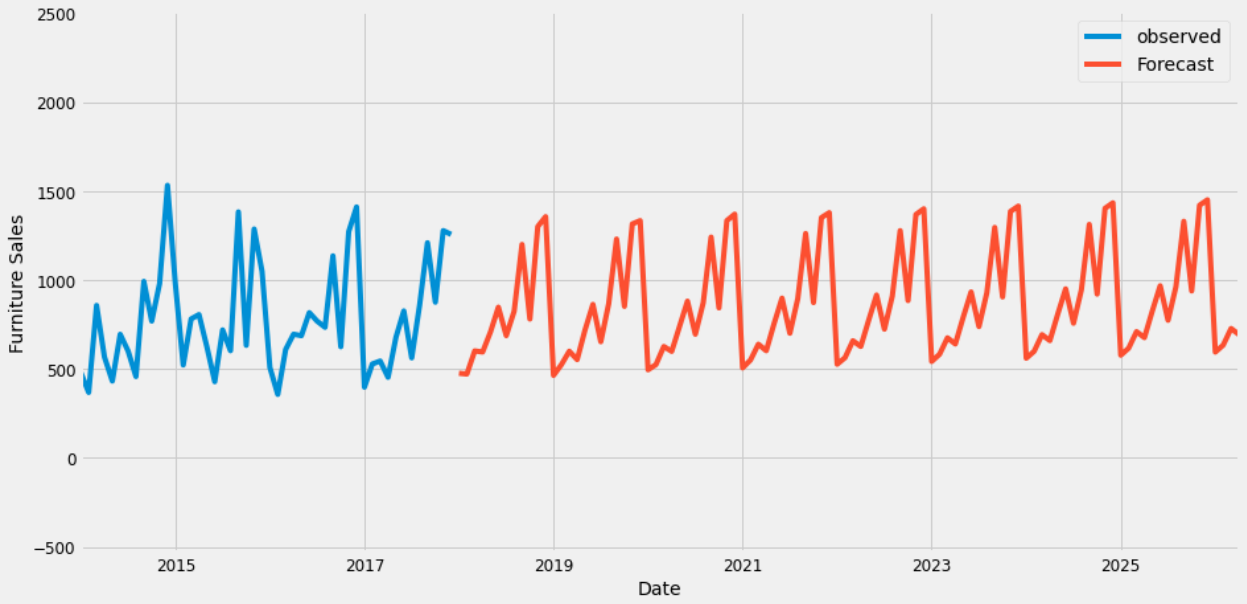
The plotted line graph displays a comparison between the observed values and the rolling forecast predictions. Upon inspection, our forecasts closely match the actual values, exhibiting a positive trend that commences at the start of the year and effectively capturing the seasonality towards the end of the year.

We can evaluate our model by getting MSE and RMSE where the mean squared error (MSE) of an estimator is a metric that gauges the average of the squared differences between estimated values and the actual values. The MSE serves as an indicator of the accuracy of an estimator, with lower MSE values indicating greater proximity to the best-fit line.

Furthermore, the Root Mean Square Error (RMSE) of our model suggests that it has the capability to forecast the average daily furniture sales in the test set with a margin of error of 151.64 in relation to the actual sales. Since our furniture sales range from around 400 to over 1200, this is deemed as a fairly impressive model.

We can as well apply the forcsstiong for as mean number of month in the future as we want , here we applied for 100 steps (month)

```
pred_uc = results.get_forecast(steps=100)
pred_ci = pred_uc.conf_int()
ax = y.plot(label='observed', figsize=(14, 7))
pred_uc.predicted_mean.plot(ax=ax, label='Forecast')
ax.fill_between(pred_ci.index,
                pred_ci.iloc[:, 0],
                pred_ci.iloc[:, 1], color='k', alpha=.25)
ax.set_xlabel('Date')
ax.set_ylabel('Furniture Sales')
plt.legend()
plt.show()
```



Our analysis indicates that our model effectively captured the seasonality of furniture sales. However, as we extend our forecast into the future, it is expected that our level of confidence in the projected values will decrease. This is demonstrated by the increasing width of the confidence intervals generated by our model as we move further into the future.

The insights gained from our examination of furniture sales piques my interest regarding the performance of other categories over time. Hence, we will proceed to compare the time series of furniture and office supplies.

Future work

In this model we used SARIMA method for forcsesting which give us a good result however there are other model that may yield more accurate result such as.

1. Exponential Smoothing (ES): This method is based on the assumption that future values of a time series are a function of its past values. It involves generating a weighted average of past observations to predict future values. **It is usually less accurate compared to SARIMA**

2. Prophet: This is a time series forecasting tool developed by Facebook that uses a decomposable model consisting of trend, seasonality, and holiday components.

3. Vector Autoregression (VAR): This is a multivariate time series forecasting technique that models multiple time series variables simultaneously, taking into account the interdependencies between them.

4. Deep Learning : This is a family of machine learning techniques that involves training deep neural networks on time series data to make predictions. Methods such as LSTM (Long Short-Term Memory) and CNN (Convolutional Neural Networks) are commonly used for time series forecasting.