PRODUCT DEMAND PREDICTION WITH MACHINE LEARNINGS:

**912421104032 : RAMANATHAN A**

**PHASE-2 PROJECT SUBMISSION DOCUMENT**

**PROJECT : mechanisms for improved accuracy in predicting stock prices**.

**INTRODUCTION:**

Consider exploring more advanced deep learning techniques like CNN-LSTM or attention mechanisms for improved accuracy in predicting stock prices.

**DATA SOURCE:**

In this section we need to put design into innovation to solve the problem. Create a document around it and share the same for assessment as per the instructions mentioned.

Dataset Link: [**https://www.kaggle.com/datasets/prasoonkottarathil/microsoft-lifetime-stocks-dataset**](https://www.kaggle.com/datasets/prasoonkottarathil/microsoft-lifetime-stocks-dataset)

**EXPLANATION FOR PHASE-2 PROJECT:**

**Data Collection and Preprocessing:**

Gather historical demand data, ensuring that it is time-stamped and organized chronologically. Preprocess the data by addressing missing values, outliers, and any other data quality issues.

**Exploratory Data Analysis (EDA):**

Conduct EDA to understand the temporal patterns and characteristics of the demand data. Look for seasonality, trends, and other recurring patterns. Visualization tools and statistical tests can be helpful in this phase.

**Incorporating time series forecasting techniques:**

* **ARIMA (Auto Regressive Integrated Moving Average):**

Suitable for stationary data with autoregressive and moving average components.

* **SARIMA (Seasonal ARIMA):**

Extends ARIMA to handle seasonal patterns in data.

* **Exponential Smoothing Methods:**

These include Holt-Winters for capturing trends and seasonality.

* **Prophet:**

Developed by Facebook, Prophet is useful for data with daily observations, holidays, and seasonality.

* **Deep Learning Models (e.g., LSTM and GRU):**

Suitable for capturing complex temporal patterns, but they may require more data and computational resources.

**Model Training:**

Train the selected time series forecasting model using historical demand data. This involves estimating model parameters and seasonal components, if applicable.

**Validation and Hyperparameter Tuning:**

Assess the model's performance using validation data or cross-validation. Fine-tune hyperparameters and adjust the model structure as needed to improve forecasting accuracy.

**Forecasting:**

Once the model is trained and validated, use it to make predictions for future time periods. These forecasts will capture temporal patterns and provide insights into expected demand behavior.

**Performance Evaluation:**

Evaluate the forecasting model's performance using appropriate metrics like Mean Absolute Error (MAE), Mean Squared Error (MSE), Root Mean Squared Error (RMSE), and forecast accuracy measures.

**Continuous Monitoring and Updating:**

Implement a process for regularly updating and retraining the model as new demand data becomes available. This ensures that the model adapts to changing demand patterns over time.

**Incorporate External Factors:**

Consider adding external variables such as promotional activities, economic indicators, or weather data to your model to account for factors that influence demand fluctuations.

**PROGRAM:**

**Product Demand Prediction:**

import pandas as pd

import numpy as np

import plotly.express as px

import seaborn as sns

import matplotlib.pyplot as plt

from sklearn.model\_selection import train\_test\_split

from sklearn.tree import DecisionTreeRegressor

data=pd.read\_csv("C:\Users\mabir\AppData\Local\Microsoft\Windows\INetCache\IE\AHLGJQP8\archive[1].zip ")

data.head()

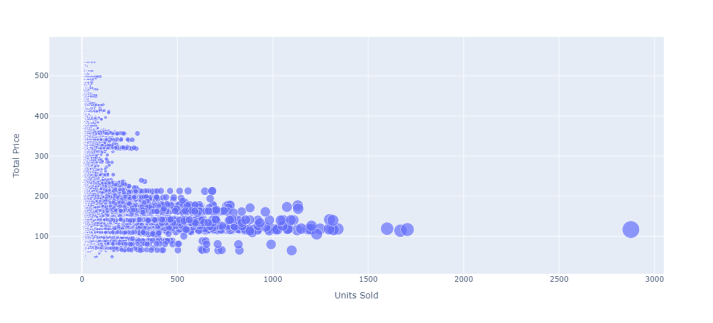
**Relationship between price and demand for the product:**

fig = px.scatter(data, x=”Units Sold”, y=”Total Price”,

size=’Units Sold’)

fig.show()

**output:**



**Correlation between the features of the dataset:**

print(data.corr())

**Output:**

**ID Store ID Total Price Base Price Units Sold**

**ID 1.000000 0.007464 0.008473 0.018932 -0.010616**

**Store ID 0.007464 1.000000 -0.038315 -0.038848 -0.004372**

**Total Price 0.008473 -0.038315 1.000000 0.958885 -0.235625**

**Base Price 0.018932 -0.038848 0.958885 1.000000 -0.140032**

**Units Sold -0.010616 -0.004372 -0.235625 -0.140032 1.000000**

1

correlations = data.corr(method='pearson')

2

plt.figure(figsize=(15, 12))

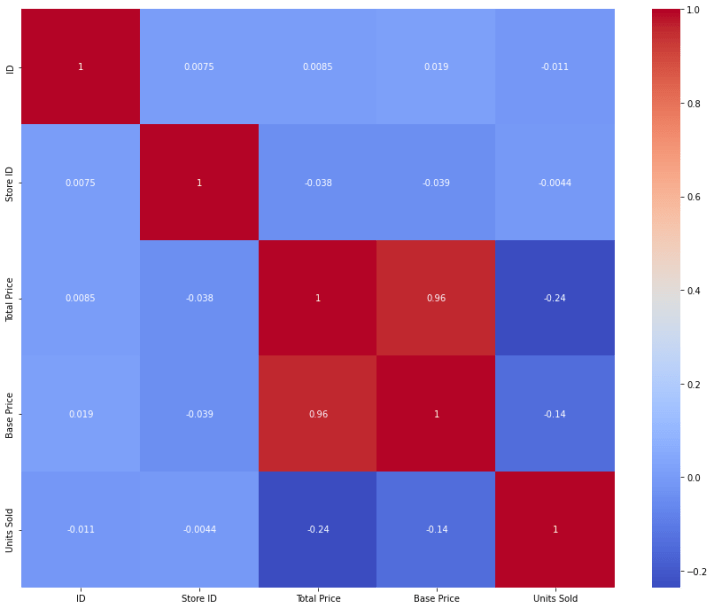
3

sns.heatmap(correlations, cmap="coolwarm", annot=True)

4

plt.show()

**Output:**



# fit an ARIMA model and plot residual errors

from pandas import datetime

from pandas import read\_csv

from pandas import DataFrame

from statsmodels.tsa.arima.model import ARIMA

from matplotlib import pyplot

# load dataset

def parser(x):

return datetime.strptime('190'+x, '%Y-%m')

series = read\_csv('shampoo-sales.csv', header=0, index\_col=0, parse\_dates=True, squeeze=True, date\_parser=parser)

series.index = series.index.to\_period('M')

# fit model

model = ARIMA(series, order=(5,1,0))

model\_fit = model.fit()

# summary of fit model

print(model\_fit.summary())

# line plot of residuals

residuals = DataFrame(model\_fit.resid)

residuals.plot()

pyplot.show()

# density plot of residuals

residuals.plot(kind='kde')

pyplot.show()

# summary stats of residuals

print(residuals.describe())

**Output:**

SARIMAX Results

==============================================================================

Dep. Variable: Sales No. Observations: 36

Model: ARIMA(5, 1, 0) Log Likelihood -198.485

Date: Thu, 10 Dec 2020 AIC 408.969

Time: 09:15:01 BIC 418.301

Sample: 01-31-1901 HQIC 412.191

- 12-31-1903

Covariance Type: opg

==============================================================================

coef std err z P>|z| [0.025 0.975]

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

ar.L1 -0.9014 0.247 -3.647 0.000 -1.386 -0.417

ar.L2 -0.2284 0.268 -0.851 0.395 -0.754 0.298

ar.L3 0.0747 0.291 0.256 0.798 -0.497 0.646

ar.L4 0.2519 0.340 0.742 0.458 -0.414 0.918

ar.L5 0.3344 0.210 1.593 0.111 -0.077 0.746

sigma2 4728.9608 1316.021 3.593 0.000 2149.607 7308.314

===================================================================================

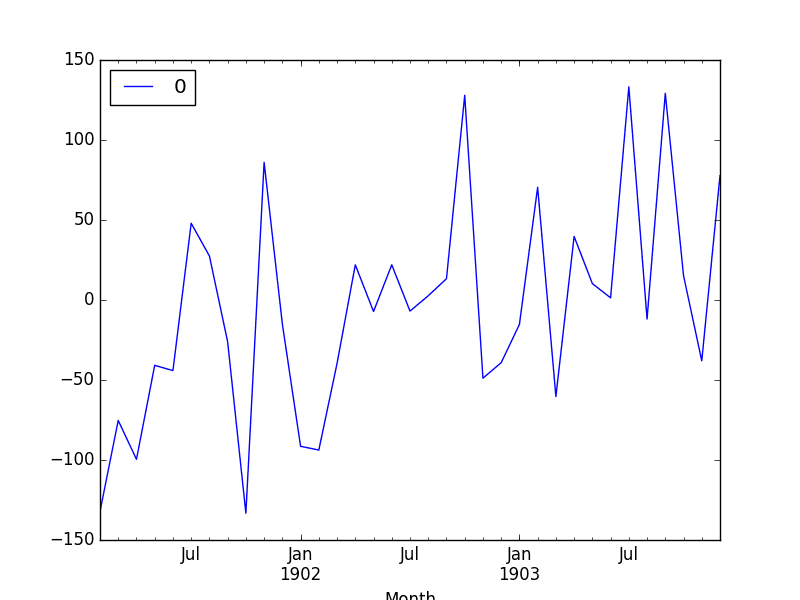
Ljung-Box (L1) (Q): 0.61 Jarque-Bera (JB): 0.96

Prob(Q): 0.44 Prob(JB): 0.62

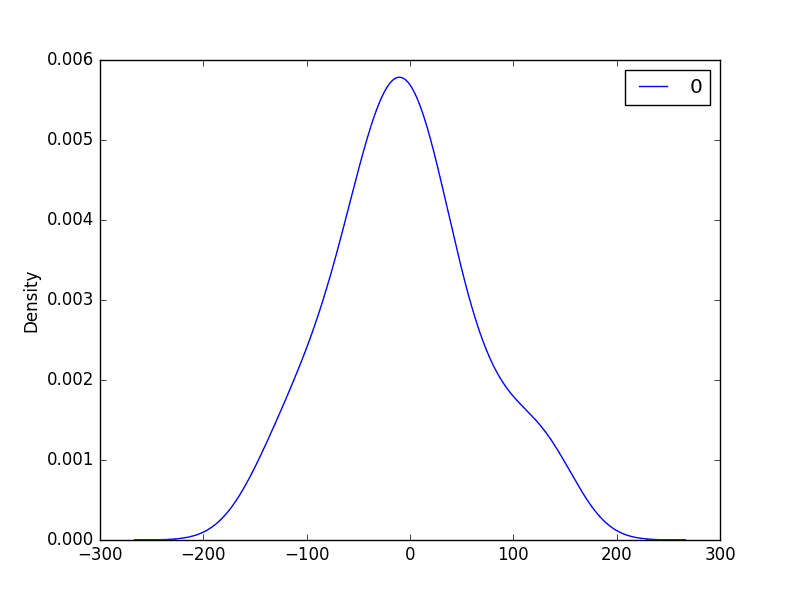
Heteroskedasticity (H): 1.07 Skew: 0.28

Prob(H) (two-sided): 0.90 Kurtosis: 2.41

First, we get a line plot of the residual errors, suggesting that there may still be some trend information not captured by the model.



Next, we get a density plot of the residual error values, suggesting the errors are Gaussian, but may not be centered on zero.



**Rolling Forecast ARIMA Model:**

# evaluate an ARIMA model using a walk-forward validation

from pandas import read\_csv

from pandas import datetime

from matplotlib import pyplot

from statsmodels.tsa.arima.model import ARIMA

from sklearn.metrics import mean\_squared\_error

from math import sqrt

# load dataset

def parser(x):

return datetime.strptime('190'+x, '%Y-%m')

series = read\_csv('shampoo-sales.csv', header=0, index\_col=0, parse\_dates=True, squeeze=True, date\_parser=parser)

series.index = series.index.to\_period('M')

# split into train and test sets

X = series.values

size = int(len(X) \* 0.66)

train, test = X[0:size], X[size:len(X)]

history = [x for x in train]

predictions = list()

# walk-forward validation

for t in range(len(test)):

model = ARIMA(history, order=(5,1,0))

model\_fit = model.fit()

output = model\_fit.forecast()

yhat = output[0]

predictions.append(yhat)

obs = test[t]

history.append(obs)

print('predicted=%f, expected=%f' % (yhat, obs))

# evaluate forecasts

rmse = sqrt(mean\_squared\_error(test, predictions))

print('Test RMSE: %.3f' % rmse)

# plot forecasts against actual outcomes

pyplot.plot(test)

pyplot.plot(predictions, color='red')

pyplot.show()

Running the example prints the prediction and expected value each iteration.

We can also calculate a final root mean squared error score (RMSE) for the predictions, providing a point of comparison for other ARIMA configurations.

predicted=343.272180, expected=342.300000

predicted=293.329674, expected=339.700000

predicted=368.668956, expected=440.400000

predicted=335.044741, expected=315.900000

predicted=363.220221, expected=439.300000

predicted=357.645324, expected=401.300000

predicted=443.047835, expected=437.400000

predicted=378.365674, expected=575.500000

predicted=459.415021, expected=407.600000

predicted=526.890876, expected=682.000000

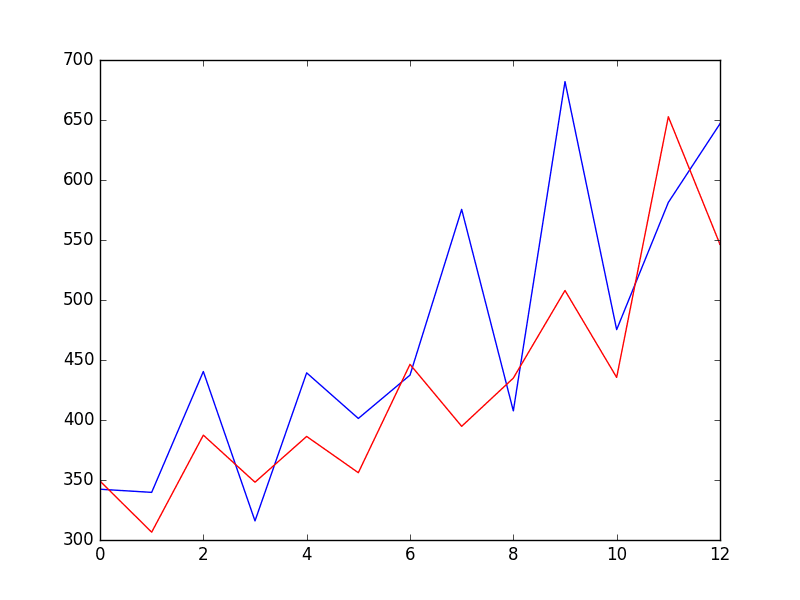
predicted=457.231275, expected=475.300000

predicted=672.914944, expected=581.300000

predicted=531.541449, expected=646.900000

Test RMSE: 89.021

A line plot is created showing the expected values (blue) compared to the rolling forecast predictions (red). We can see the values show some trend and are in the correct scale.



## Seasonal ARIMA (SARIMA):

SARIMAXmodel = SARIMAX(y, order = (5, 4, 2), seasonal\_order=(2,2,2,12))

SARIMAXmodel = SARIMAXmodel.fit()

y\_pred = SARIMAXmodel.get\_forecast(len(test.index))

y\_pred\_df = y\_pred.conf\_int(alpha = 0.05)

y\_pred\_df["Predictions"] = SARIMAXmodel.predict(start = y\_pred\_df.index[0], end = y\_pred\_df.index[-1])

y\_pred\_df.index = test.index

y\_pred\_out = y\_pred\_df["Predictions"]

plt.plot(y\_pred\_out, color='Blue', label = 'SARIMA Predictions')

plt.legend()

## Output:

## time-series-forecasting-python

**Prophet:**

# make an in-sample forecast

from pandas import read\_csv

from pandas import to\_datetime

from pandas import DataFrame

from fbprophet import Prophet

from matplotlib import pyplot

# load data

path = 'https://raw.githubusercontent.com/jbrownlee/Datasets/master/monthly-car-sales.csv'

df = read\_csv(path, header=0)

# prepare expected column names

df.columns = ['ds', 'y']

df['ds']= to\_datetime(df['ds'])

# define the model

model = Prophet()

# fit the model

model.fit(df)

# define the period for which we want a prediction

future = list()

for i in range(1, 13):

date = '1968-%02d' % i

future.append([date])

future = DataFrame(future)

future.columns = ['ds']

future['ds']= to\_datetime(future['ds'])

# use the model to make a forecast

forecast = model.predict(future)

# summarize the forecast

print(forecast[['ds', 'yhat', 'yhat\_lower', 'yhat\_upper']].head())

# plot forecast

model.plot(forecast)

pyplot.show()

Running the example forecasts the last 12 months of the dataset.

The first five months of the prediction are reported and we can see that values are not too different from the actual sales values in the dataset(output).

ds yhat yhat\_lower yhat\_upper

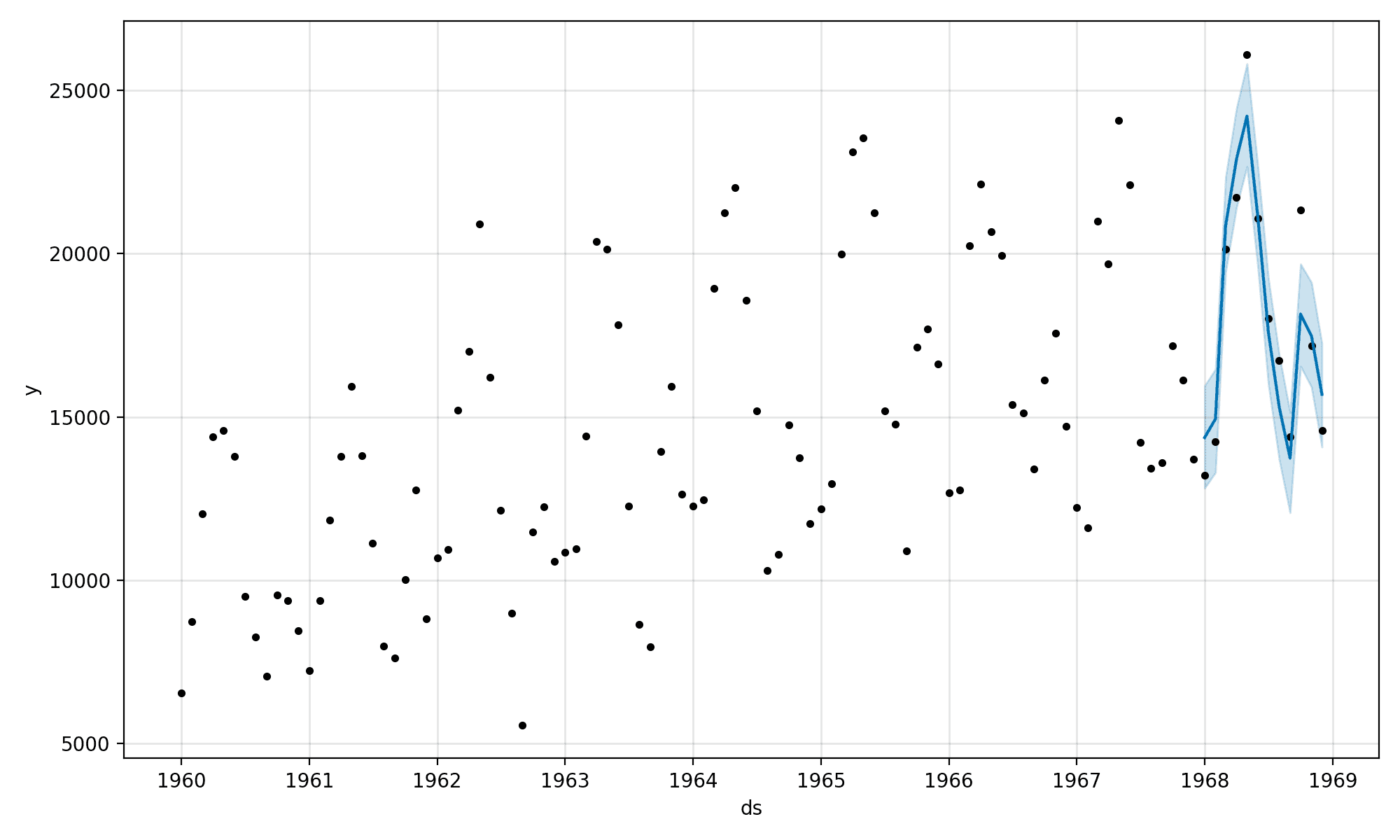
0 1968-01-01 14364.866157 12816.266184 15956.555409

1 1968-02-01 14940.687225 13299.473640 16463.811658

2 1968-03-01 20858.282598 19439.403787 22345.747821

3 1968-04-01 22893.610396 21417.399440 24454.642588

4 1968-05-01 24212.079727 22667.146433 25816.191457



Tying this together, the example below demonstrates how to evaluate a Prophet model on a hold-out dataset.

# evaluate prophet time series forecasting model on hold out dataset

from pandas import read\_csv

from pandas import to\_datetime

from pandas import DataFrame

from fbprophet import Prophet

from sklearn.metrics import mean\_absolute\_error

from matplotlib import pyplot

# load data

path = 'https://raw.githubusercontent.com/jbrownlee/Datasets/master/monthly-car-sales.csv'

df = read\_csv(path, header=0)

# prepare expected column names

df.columns = ['ds', 'y']

df['ds']= to\_datetime(df['ds'])

# create test dataset, remove last 12 months

train = df.drop(df.index[-12:])

print(train.tail())

# define the model

model = Prophet()

# fit the model

model.fit(train)

# define the period for which we want a prediction

future = list()

for i in range(1, 13):

date = '1968-%02d' % i

future.append([date])

future = DataFrame(future)

future.columns = ['ds']

future['ds'] = to\_datetime(future['ds'])

# use the model to make a forecast

forecast = model.predict(future)

# calculate MAE between expected and predicted values for december

y\_true = df['y'][-12:].values

y\_pred = forecast['yhat'].values

mae = mean\_absolute\_error(y\_true, y\_pred)

print('MAE: %.3f' % mae)

# plot expected vs actual

pyplot.plot(y\_true, label='Actual')

pyplot.plot(y\_pred, label='Predicted')

pyplot.legend()

pyplot.show()

**Output:**

