Research Associate KTP with the University of Essex and Croud Inc

Technical Task

Ashkan Lotfipoor
School of Energy, Geoscience, Infrastructure and Society (EGIS),
Heriot-Watt University

July 26, 2023

Introduction

This report is the result of my analysis of the technical assignment for the KTP Associate with the University of Essex and Croud Inc. This report is divided into two sections. The first section presents the results of the marketing data modelling on the provided dataset. Then, section two provide results for the question in Bayesian methods section of the task.

1 Classical Marketing Data Modelling

1.1 Modelling

In this section several regression models are built to determine the effectiveness of marketing strategies of the company and also to predict the revenue. Firstly the data is divided to two dataset, the train and test set, to effectively validate the results of the model on unseen data. The 2016/05/30 date is used as the split point in the dataset.

In this section, the temporal aspect of the data is not considered in the modelling process. So, the week number and Date column are dropped from the dataframe. The remaining variables or predictors in the data are media1_S (spend on media 1), media2_S (spend on media 2), media3_S (spend on media 3), competitor_sales (competitors sales), and newsletter (number of newsletter subscription). There are various statistical methods for feature selection in a regression task, such as Chi-square Test, Variance Threshold, Dispersion Ratio, Random Forest Importance and etc. I used Correlation Coefficient to select predictors. Figure 1 presents the correlation matrix. Here, the variables with Coefficient less than 0.4 were dropped from the dataframe. In the final dataset only media1_S, competitor_sales, and newsletter are used to predict revenue.

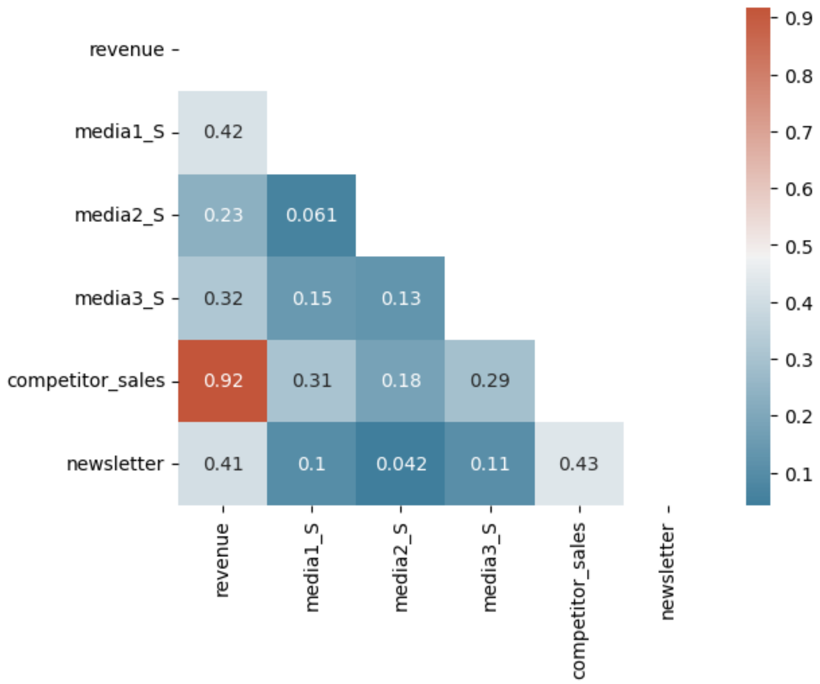

Figure 1: Correlation Matrix

For selection of the model, several commonly used regression model are used to model the marketing dataset. Using SKlearn library in Python, five algorithm are used for modelling the marketing data: linear regression, stochastic gradient descent regression, Random Forest, gradient boosting regression, Multilayer Perceptron (MLP). The results are presented in Table 1 and Figure 2.

Linear Regression and Stochastic Gradient Descent Regression models perform rather well in this task. The reported $R\hat{2}$ value for both models are 95%. Other methods does not perform well, but it is worth noting that each of these algorithms need hyperparameter tuning to perform well. Here, grid search was performed, but to reach the maximum potential of each model more analysis needs to be done. Analysing the effect of variables on

Table 1: Forecasting Error for Methods Used

Model	RMSE	MAE	R2	MAPE
Linear Regression	41441.49	34845.88	0.952	1.36
Stochastic Gradient Descent	41650.05	34783.47	0.952	1.36
Multi-layer Perceptron	123783.20	90774.79	0.576	3.55
Random Forest	139584.86	116637.73	0.461	4.79
Gradient Boosting	145422.27	136118.76	0.415	5.47

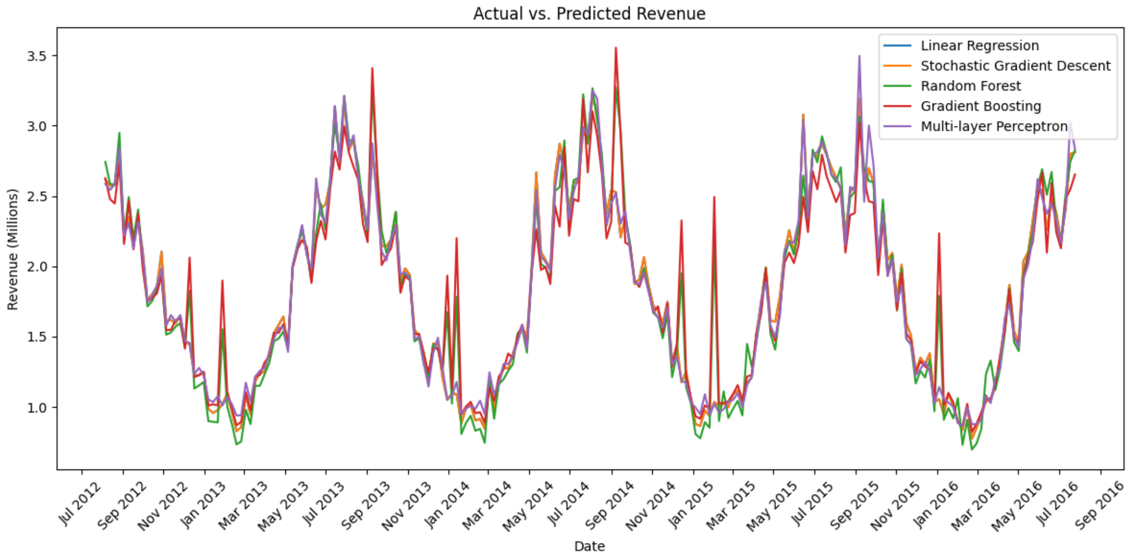

Figure 2: Forecasted Revenue

the regression results is an essential step in understanding the relationship between the independent variables (features) and the dependent variable (target) in a regression model. Somme common methods are coefficient estimates, hypothesis tests, and residual analysis which will be performed in the next section. These are the coefficient of the regression model: 112513.37, 611400.75, and 12931.45.

The biggest limitation of the model used in the context of media marketing is the lack of model interpretability. Advanced regression models, like neural networks, can be complex and lack interpretability, making it challenging for marketers to understand the model's decision-making process. That's why the methods such as Bayesian model estimation can be usefull in this field.

1.2 Prediction

This section overlaps with the previous section, the data was split into two datasets and appropriate variables were used for developing the prediction model. Figure 3 presents the forecasted revenue for the test period.

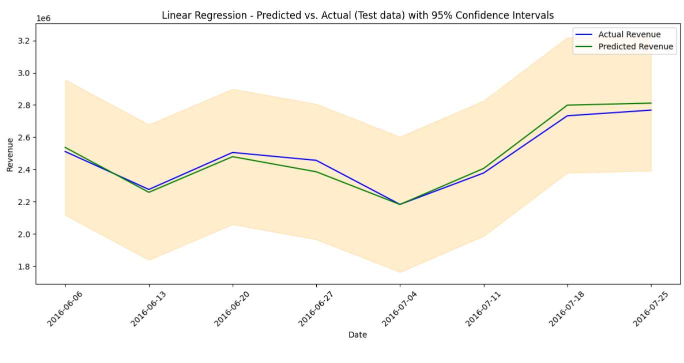

Figure 3: Forecast Results for Test Set

The performance of the model was evaluated using various metrics, RMSE, MAE, $R\hat{2}$ and MAPE. The results for the test set are presented in Table 1.

1.3 Temporal effects

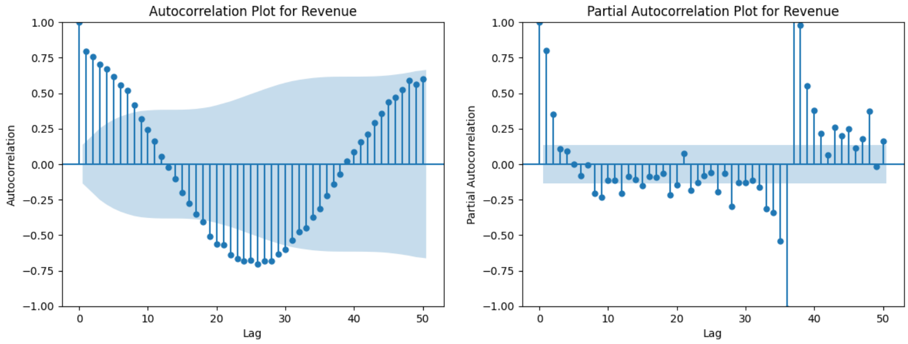

Figure 4: ACF and PACF Plots

As it is suggested by a colleague in the task, there are a clear affects of time seen in the marketing dataset. This can be seen by plotting the revenue against time. However, to be more precise, we can plot the Autocorrelation Function (ACF) of the revenue to determine whether trends and seasonal patterns are present. Figure 4 presents the ACF plot. As the autocorrelation is significant for number of lags, we can conclude that there is a temporal effect in the dataset. From the plot is can also be concluded that there is a clear trend and seasonality in the dataset. We can investigate this further by decomposition of the dataset. Figure 5 shows the result of the time series decomposition of the revenue. This enables us to identify patterns and trends that might impact the revenue data.

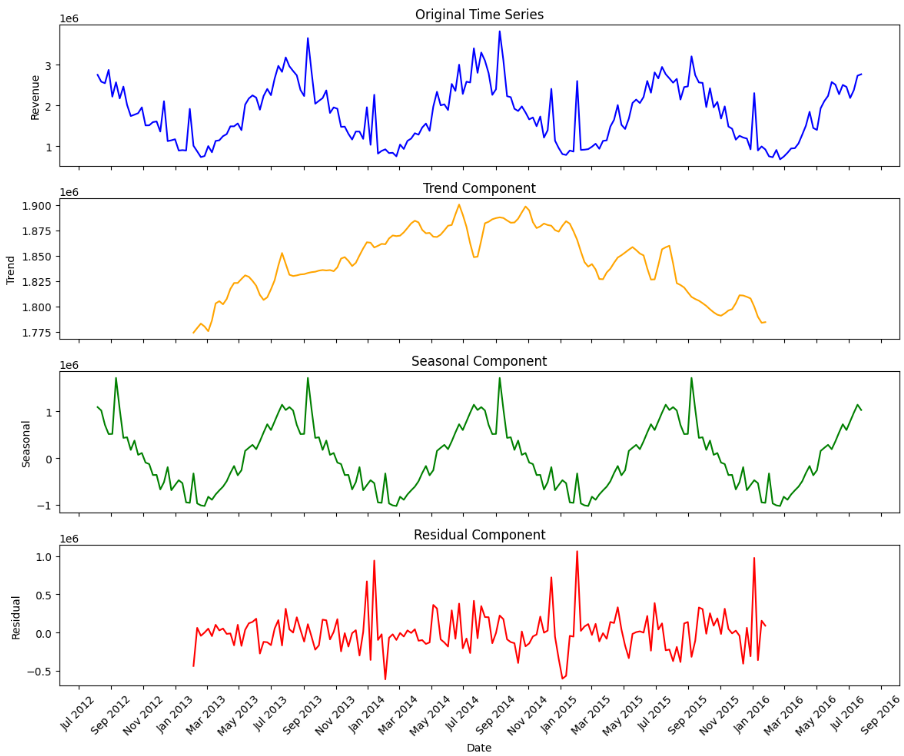

Figure 5: Time Series Decomposition of the Revenue

Next, we are going to model the dataset with the temporal components of the data. For this I used ARIMA to forecast the revenue. For this, we need to first check to see if the data is stationary, as this is one the requirements of the ARIMA algorithm. If the data is not stationary, we need to perform transformation such as differencing on the revenue. For selecting the autoregressive term or the number of lag observations in

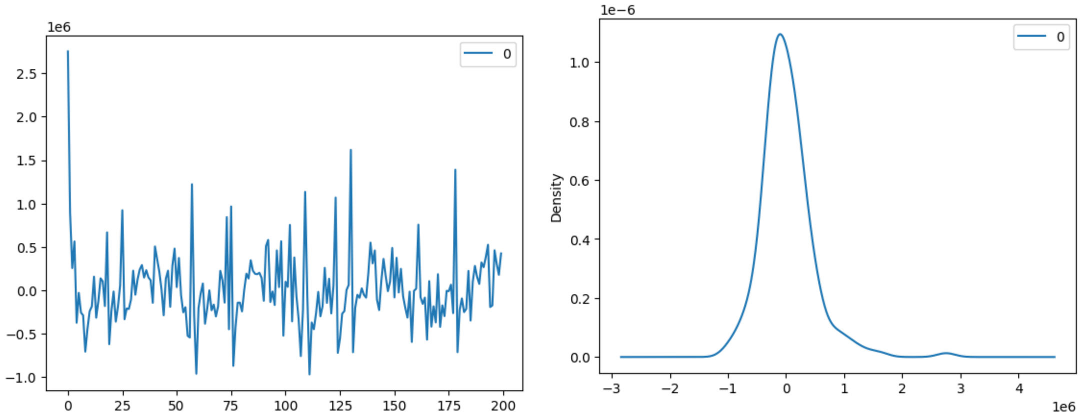

Figure 6: Line Plot and Denesity Plot of Residual Error

the ARIMA we need to plot the Partial Autocorrelation function (PACF) which is presented in Figure 4. By analysing the ACF and PACF the number of lag was selected as 12. This value was also selected during the Grid Search process for hyperparameter optimisation. Details of this are presented in the code. The best ARIMA order (p, d, q) is selected as following: 12, 1, 2. For this the RMSE value was reported as 217787.24.

Aside from the tipycal metrics that are used for evaluating the regression error, we can also plot the residuals as a line graph and also plot the density of residuals to check if the model has captured all of patterns in the data. This is presented in Figure 6.

Using ARIMA, we considered the temporal aspect of the data, but we discard the other features in the data. We can use other algorithms to predict revenue using all of the available features along side the temporal features. This can be done by using feature engineering, we can add new features to the data, like number of week, month, and year to do this. However, using windowing method we can transform the data and keep the temporal aspect of the data too. I wrote a function in python to do this. After this we can use any algorithm to perform prediction.

In the code submitted with this report a deep learning model is used to predict the revenue. The RMSE error is 213412.58, but please bear in mind that deep learning needs a large quantity of data for the algorithm to perform. Here we only used 208 data points. Also, by spending more time on the model and its architecture we can improve the result further. Figure 7 shows the result of the modelling for test set.

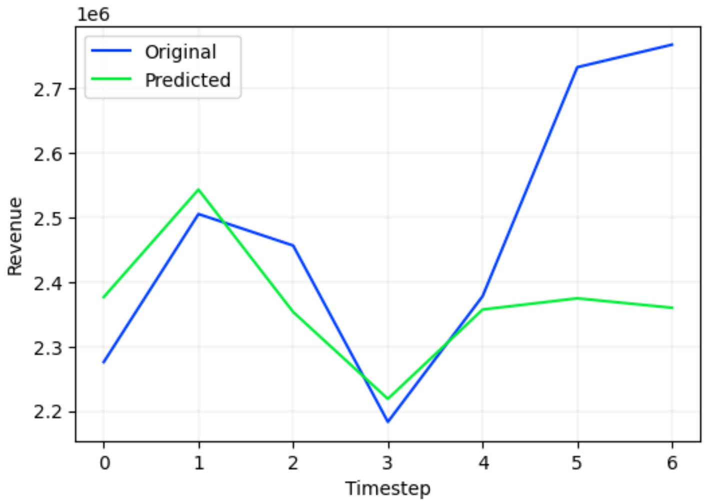

Figure 7: Deep Learning Results

2 Bayesian Methods

In this section I fit a simple neural network in a Bayesian framework using non-informative priors. In this example, the priors will be non-informative, which means they will have minimal impact on the final results and will let the data speak for itself. Non-informative priors are often used when we have little or no prior knowledge about the parameters we're trying to estimate, which is a good starting point in our Bayesian analysis.

Common noninformative priors include $unif(-1000, 1000)$ and $N(0, 10, 000)$. In this task I used a normal distribution for the priors, specifically, for weight and biases of the model, I used a normal distribution with mean zero and standard deviation of 10.

A prior distribution is said to be conjugate to a likelihood function if the resulting posterior distribution is in the same family of probability distributions as the prior. For example, in the case of a normal likelihood function with unknown mean and known variance, the conjugate prior is another normal distribution. If the likelihood is a binomial distribution, the conjugate prior is a beta distribution. If the likelihood is a Poisson distribution, the conjugate prior is a gamma distribution. Looking at the posterior distribution I can say the priors are conjugate. For MCMC I used the default value in PyMC, which is NUTS, a sampler for continuous variables based on Hamiltonian mechanics. It allows us to sample from the posterior distribution and obtain estimates for the model parameters.

Comparing the results from the first section and the result of the Bayesian framework produced similar outputs. The model has the RMSE value of 94420.70 which is on par with the performance of the previous model used. Figure **??** shows the result of the prediction on the test set.

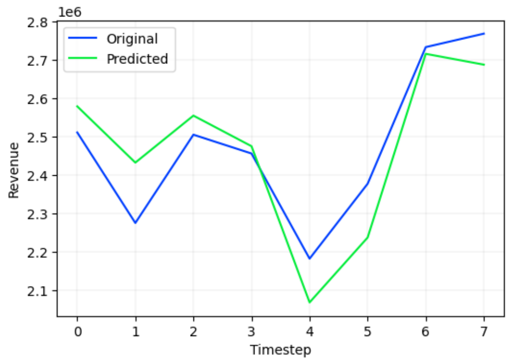

Figure 8: Bayesian Neural Network Result for Test Set

2.1 Prior choice

To modify the prior specification to account for the expert's belief that "Media 3 has no impact in generating marketing revenues," we can update the prior for the corresponding coefficient to reflect this belief. We want to incorporate the expert's opinion while still allowing the data to influence the final estimates. We can do this by using a prior distribution for the coefficient of "Media 3" with a high variance, indicating a wide range of possible values. For example, we can use a Normal distribution for the coefficient of Media 3 with a mean of 0 and a large standard deviation, such as 10. This would express the expert's belief that the coefficient is likely to be around 0, but they are not certain and allow the data to adjust this belief.

However, I already use a normal distribution with mean 0 and standard deviation of 10 to accommodate non informative priors in the model. So to answer the question raised in the task, I must say based on the estimated parameters from the model, the expert is correct and Media 3 has no to little impact on the revenue.

Additionally, to incorporate the opinions of the other expert, we can set informative priors for the coefficients of "newsletter," "media1_S," "media2_S," and "competitor_sales" based on the expert's beliefs. For example:

"newsletter": We can use a Normal distribution with a mean of 1 and a small standard deviation, such as 0.1, to express the belief that there is a strictly positive relationship between "newsletter" and marketing revenues. For "media1_S" we can use a Normal distribution with a mean of 2 and a small standard deviation, such as 0.1, to express the belief that one unit invested in "Media 1" will result in two units of marketing revenues returned. For "media2_S" we can use a Normal distribution with a mean of 8 (four times the mean for "media1_S") and a small standard deviation, such as 0.1, to express the belief that the impact of "Media

2" is four times that of "Media 1."

For "competitor_sales" we can use a Normal distribution with a mean of 0.15 and a small standard deviation, such as 0.05, to express the belief that for every unit increase in "competitor_sales," the change on revenues returned must range between 0 and 0.3.

Each of these hypothesis need to be implemented in the model separately and the results and posterior distribution needs to be carefully study. Unfortunately, I did not had enough time to do this analysis. The code needed for doing this is provided in the submitted code. Given more time, I could have make the necessary analysis to come up with answers.

3 Closing Remarks

I would want to convey my gratitude for the opportunity to demonstrate my qualifications for this post at the University of Essex and Croud Inc. This is an intriguing role for me, and I would be pleased to go to the next stage of the hiring procedure.

The Python codes can be found on the following pages. You will also receive a link to a colab notebook along with this report.
Click here to open in Google Colab

Python Code

```
# -*- coding: utf-8 -*-
"""alotfipoor_ktp_associate_essex.ipynb

Automatically generated by Colaboratory.

Original file is located at
    https://colab.research.google.com/drive/16cX08SFB-SyHqLJpUx33vgg6iqRHmVel

# Setup
"""

# Commented out IPython magic to ensure Python compatibility.
# from datetime import datetime
import os, glob, re
import numpy as np
import pandas as pd
import warnings
from math import sqrt

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

# import seaborn as sns
import matplotlib as mpl
import matplotlib.pyplot as plt
# %matplotlib inline
import seaborn as sns
from matplotlib.ticker import FuncFormatter
import matplotlib.dates as mdates

from IPython.core.display import display, HTML
display(HTML("<style>.container { width:100% !important; }</style>"))

# plotly
import plotly.express as px
import plotly.graph_objs as go
import plotly.io as pio
pio.renderers.default = 'colab'

from statsmodels.tsa.stattools import adfuller
from statsmodels.tsa.arima.model import ARIMA
from statsmodels.graphics.tsaplots import plot_pacf
from statsmodels.graphics.tsaplots import plot_acf
from statsmodels.tsa.seasonal import seasonal_decompose

import pymc as pm
import arviz as az

# import optuna

# from scipy.stats import uniform, randint
# from sklearn.model_selection import TimeSeriesSplit, cross_val_score, GridSearchCV, RandomizedSearchCV

from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler, MinMaxScaler
from sklearn.linear_model import LinearRegression, SGDRegressor
from sklearn.svm import SVR
from sklearn.gaussian_process import GaussianProcessRegressor
from sklearn.gaussian_process.kernels import WhiteKernel, DotProduct
from sklearn.tree import DecisionTreeRegressor
```

```python
from sklearn.ensemble import RandomForestRegressor, GradientBoostingRegressor
from sklearn.neural_network import MLPRegressor
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score, mean_squared_log_error

from keras.models import Sequential
from keras.layers import Dense, LSTM, Dropout, TimeDistributed, Flatten, BatchNormalization
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.callbacks import LearningRateScheduler, EarlyStopping, ReduceLROnPlateau, ModelCheckpoint
# import tensorflow as tf

# data Table Display
# %load_ext google.colab.data_table

# to make this notebook's output stable across runs
np.random.seed(42)

# mount google colab
from google.colab import drive
drive.mount('/content/drive')

# load  dataset
df = pd.read_csv('/content/drive/MyDrive/[04] Colab Notebooks/data/weekly_media_sample.csv')
df['DATE'] = pd.to_datetime(df['DATE'])

df.head(5)

fig = go.Figure()
fig.add_trace(go.Scatter(x=df.DATE, y=df.revenue, name='Revenue', line=dict(color='blue', width=1)))
fig.add_trace(go.Scatter(x=df.DATE, y=df.competitor_sales, name='Competitor Sales', line=dict(color='red', width=.8)))
fig.add_trace(go.Scatter(x=df.DATE, y=df.newsletter, name='Newsletter Subscription', line=dict(color='green', width=.8)))
fig.show()

"""# Part 1: Classical Marketing Data Modelling"""

# Creating a subset of dataframe for modelling
# df2 = df[['DATE','revenue','media1_S','media2_S','media3_S','competitor_sales','newsletter']].copy()
df2 = df[['DATE','revenue','media1_S','competitor_sales','newsletter']].copy()

# Calculate correlation coefficients with the target variable 'revenue'
correlation_matrix = df2.corr()
correlation_with_revenue = correlation_matrix['revenue'].abs().sort_values(ascending=False)

# Print the correlation coefficients
print(correlation_with_revenue)

mask = np.triu(np.ones_like(correlation_matrix, dtype=bool))
cmap = sns.diverging_palette(230, 20, as_cmap=True)
sns.heatmap(correlation_matrix,mask = mask, cmap=cmap, annot=True)

# Define the split date based on the given threshold (e.g., '2016-05-30' in this case)
split_date = '2016-05-30'

# Split the data into train and test sets
train = df2[df2.DATE <= split_date]
test = df2[df2.DATE > split_date]

# Extract the target variable 'revenue' from the training and test data
y_train = train['revenue']
y_test = test['revenue']

# Extract the input features from the training and test data
```

```python
X_train = train.drop(['revenue', 'DATE'], axis=1)
X_test = test.drop(['revenue', 'DATE'], axis=1)

# Scale the data
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

# Create instances of regression models with hyperparameters
linear_reg = LinearRegression()
sgd_reg = SGDRegressor()
# svm_reg = SVR(C=10, degree=3, gamma='auto', kernel='rbf')
# gaussian_process_reg = GaussianProcessRegressor(kernel=DotProduct() + WhiteKernel())
random_forest_reg = RandomForestRegressor(n_estimators=200, max_depth=20, random_state=42)
gradient_boosting_reg = GradientBoostingRegressor(n_estimators=200, learning_rate=0.01, max_depth=20,
↪  random_state=42)
mlp_reg = MLPRegressor(hidden_layer_sizes=(500, 300, 200), batch_size=64, learning_rate='adaptive', alpha=0.1,
↪  max_iter=5000, random_state=42)

# Create DataFrames to store predicted values
train_predictions_df = pd.DataFrame(data=y_train.values, columns=['Actual Revenue'])
test_predictions_df = pd.DataFrame(data=y_test.values, columns=['Actual Revenue'])

results_df = pd.DataFrame(columns=['Model', 'RMSE', 'MAE', 'R2', 'MAPE'])

# Mean Absolute Percentage Error (MAPE) function
def mean_absolute_percentage_error(y_true, y_pred):
    return np.mean(np.abs((y_true - y_pred) / y_true)) * 100

# Function to evaluate and append results to the DataFrame
def evaluate_model(model_name, model, X_train, y_train, X_test, y_test):
    model.fit(X_train, y_train)
    y_train_pred = model.predict(X_train)
    y_test_pred = model.predict(X_test)
    train_predictions_df[model_name] = y_train_pred
    test_predictions_df[model_name] = y_test_pred
    rmse = mean_squared_error(y_test, y_test_pred, squared=False)
    mae = mean_absolute_error(y_test, y_test_pred)
    r2 = r2_score(y_test, y_test_pred)
    mape = mean_absolute_percentage_error(y_test, y_test_pred)
    results_df.loc[len(results_df)] = [model_name, rmse, mae, r2, mape]

# Evaluate each model and store the results
evaluate_model('Linear Regression', linear_reg, X_train, y_train, X_test, y_test)
evaluate_model('Stochastic Gradient Descent', sgd_reg, X_train, y_train, X_test, y_test)
# evaluate_model('SVM', svm_reg, X_train, y_train, X_test, y_test)
# evaluate_model('Gaussian Processes', gaussian_process_reg, X_train, y_train, X_test, y_test)
evaluate_model('Random Forest', random_forest_reg, X_train, y_train, X_test, y_test)
evaluate_model('Gradient Boosting', gradient_boosting_reg, X_train, y_train, X_test, y_test)
evaluate_model('Multi-layer Perceptron', mlp_reg, X_train, y_train, X_test, y_test)

results_df.head()

# Print the coefficient of the linear regression
linear_reg.coef_

# Merge train_predictions_df and test_predictions_df while keeping 'DATE' column
merged_predictions_df = pd.concat([train_predictions_df, test_predictions_df])

# Add the 'DATE' column from df2 to merged_predictions_df
merged_predictions_df['DATE'] = np.sort(df2['DATE'])

# Create a trace for each model's predicted values against the actual 'revenue'
```

```python
traces = []
for column in merged_predictions_df.columns[:-1]:  # Exclude the first and last columns ('Actual Revenue' and 'DATE')
    trace = go.Scatter(
        x=merged_predictions_df['DATE'],
        y=merged_predictions_df[column],
        mode='lines',
        name=column
    )
    traces.append(trace)

# Create the layout for the plot
layout = go.Layout(
    title='Actual vs. Predicted Revenue',
    xaxis=dict(title='Date'),
    yaxis=dict(title='Revenue'),
)

# Create the figure and plot
fig = go.Figure(data=traces, layout=layout)
fig.show()

# Convert 'DATE' column to datetime object
# merged_predictions_df['DATE'] = pd.to_datetime(merged_predictions_df['DATE'])

# Create a line chart for each model's predicted values against the actual 'revenue' over time
plt.figure(figsize=(12, 6))
for column in merged_predictions_df.columns[1:-1]:  # Exclude the first and last columns ('Actual Revenue' and 'DATE')
    plt.plot(merged_predictions_df['DATE'], merged_predictions_df[column], label=column)

# Set the x-axis label to 'Date' and y-axis label to 'Revenue'
plt.xlabel('Date')
plt.ylabel('Revenue (Millions)')

# Set the title of the plot
plt.title('Actual vs. Predicted Revenue')

# Show the legend to identify each model's line
plt.legend()

# Format x-axis date labels to show only the month and year
date_format = mdates.DateFormatter("%b %Y")
plt.gca().xaxis.set_major_formatter(date_format)

# Set the x-axis tick interval to show labels every n months (adjust n as needed)
n_months = 2
plt.gca().xaxis.set_major_locator(mdates.MonthLocator(interval=n_months))

# Rotate the x-axis labels for better visibility
plt.xticks(rotation=45)

# Format y-axis tick labels in millions
def millions_formatter(x, pos):
    return f'{x / 1e6:.1f}'

formatter = FuncFormatter(millions_formatter)
plt.gca().yaxis.set_major_formatter(formatter)

# Show the plot
plt.tight_layout()
plt.show()

```

```
# Fit the Linear Regression model
linear_reg.fit(X_train, y_train)

# Predict on the test set
Y_pred_linear = linear_reg.predict(X_test)

# Calculate the 95% confidence intervals for the predictions
conf_interval_linear = 1.96 * np.std(Y_pred_linear)  # 1.96 is the z-score for 95% confidence

# # Create a DataFrame with predictions and confidence intervals for the last 20 rows
# df_pred_linear = pd.DataFrame({'DATE': df2['DATE'].tail(8).values,
#                                'Actual Revenue': y_test[-8:].values.flatten(),
#                                'Predicted Revenue': Y_pred_linear[-8:].flatten(),
#                                'Lower CI': (Y_pred_linear[-8:] - conf_interval_linear).flatten(),
#                                'Upper CI': (Y_pred_linear[-8:] + conf_interval_linear).flatten()})

# Create a DataFrame with predictions and confidence intervals for the last 20 rows
df_pred_linear = pd.DataFrame({'DATE': df2['DATE'].tail(8).values,
                               'Actual Revenue': y_test.values.flatten(),
                               'Predicted Revenue': Y_pred_linear.flatten(),
                               'Lower CI': (Y_pred_linear - conf_interval_linear).flatten(),
                               'Upper CI': (Y_pred_linear + conf_interval_linear).flatten()})

# Plot the last 20 rows of predictions, actual observations, and confidence intervals
plt.figure(figsize=(12, 6))
plt.plot(df2['DATE'].tail(8), df2['revenue'].tail(8), label='Actual Revenue', color='blue')
plt.plot(df_pred_linear['DATE'], df_pred_linear['Predicted Revenue'], label='Predicted Revenue', color='green')
plt.fill_between(df_pred_linear['DATE'], df_pred_linear['Lower CI'], df_pred_linear['Upper CI'], alpha=0.2,
↪  color='orange')
plt.xlabel("Date")
plt.ylabel("Revenue")
plt.title("Linear Regression - Predicted vs. Actual (Test data) with 95% Confidence Intervals")
plt.legend()
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()

"""## ARIMA"""

sns_c = sns.color_palette(palette="deep")

sns.pairplot(
    data=df2, kind="scatter", height=2, plot_kws={"color": sns_c[1]}, diag_kws={"color": sns_c[2]}
);

plt.figure(figsize=(12, 6))
plot_acf(df2['revenue'], lags=50)
plt.xlabel('Lag')
plt.ylabel('Autocorrelation')
plt.title('Autocorrelation Plot for Revenue')
plt.show()

plt.figure(figsize=(12, 6))
plot_pacf(df2['revenue'], lags=50)
plt.xlabel('Lag')
plt.ylabel('Partial Autocorrelation')
plt.title('Partial Autocorrelation Plot for Revenue')
plt.show()

# Performing seasonal decomposition
result = seasonal_decompose(df2['revenue'], model='additive', period=52)  # Assuming weekly seasonality
↪  (period=52)

```

```python
# Creating a subplot layout
fig, axes = plt.subplots(4, 1, figsize=(12, 10), sharex=True)

# Plotting the original time series
axes[0].plot(df2['DATE'], df2['revenue'], label='Original', color='blue')
axes[0].set_ylabel('Revenue')
axes[0].set_title('Original Time Series')

# Plotting the trend component
axes[1].plot(df2['DATE'], result.trend, label='Trend', color='orange')
axes[1].set_ylabel('Trend')
axes[1].set_title('Trend Component')

# Plotting the seasonal component
axes[2].plot(df2['DATE'], result.seasonal, label='Seasonal', color='green')
axes[2].set_ylabel('Seasonal')
axes[2].set_title('Seasonal Component')

# Plotting the residual component
axes[3].plot(df2['DATE'], result.resid, label='Residual', color='red')
axes[3].set_xlabel('Date')
axes[3].set_ylabel('Residual')
axes[3].set_title('Residual Component')

# Format x-axis date labels to show only the month and year
date_format = mdates.DateFormatter("%b %Y")
axes[-1].xaxis.set_major_formatter(date_format)

# Set the x-axis tick interval to show labels every n months (adjust n as needed)
n_months = 2
axes[-1].xaxis.set_major_locator(mdates.MonthLocator(interval=n_months))

# Rotate the x-axis labels for better visibility
plt.xticks(rotation=45)

# Adjusting the layout
plt.tight_layout()

# Show the plot
plt.show()

# Perform the ADF test on the 'revenue' data
result = adfuller(df2['revenue'])

# Extract the test statistic and p-value
test_statistic = result[0]
p_value = result[1]

print("ADF Test Statistic:", test_statistic)
print("p-value:", p_value)

if p_value <= 0.05:
    print("The data is stationary.")
else:
    print("The data is not stationary.")

# Find the optimal ARIMA hyperparameters using grid search
best_aic = float("inf")
best_order = None

# Define the ranges for p, d, and q
p_range = range(0, 15)  # Example: p can be 0, 1, or 2
d_range = range(0, 2)  # Example: d can be 0 or 1
```

```
q_range = range(0, 3)  # Example: q can be 0, 1, or 2

for p in p_range:
    for d in d_range:
        for q in q_range:
            try:
                arima_model = ARIMA(train['revenue'], order=(p, d, q))
                arima_fit = arima_model.fit()

                # Calculate AIC score for the current ARIMA model
                aic = arima_fit.aic

                if aic < best_aic:
                    best_aic = aic
                    best_order = (p, d, q)

            except:
                continue

# Fit the ARIMA model with the best parameters to the training data
arima_model = ARIMA(train['revenue'], order=best_order)
arima_fit = arima_model.fit()

# Make predictions on the test set
y_pred_arima = arima_fit.forecast(steps=len(test))

# Evaluate the performance using Mean Squared Error (MSE)
mse_arima = mean_squared_error(test['revenue'], y_pred_arima)
rmse_arima = np.sqrt(mse_arima)

print("Best ARIMA Order (p, d, q):", best_order)
print("ARIMA RMSE:", rmse_arima)

# Plot the actual revenue and ARIMA predictions
plt.figure(figsize=(10, 6))
plt.plot(df2.index, df2['revenue'], label='Actual Revenue', color='blue')
plt.plot(test.index, y_pred_arima, label='ARIMA Predictions', color='red')
plt.xlabel('Date')
plt.ylabel('Revenue')
plt.title('Actual Revenue vs. ARIMA Predictions')
plt.legend()
plt.show()

# summary of fit model
print(arima_fit.summary())

# line plot of residuals
residuals = pd.DataFrame(arima_fit.resid)
residuals.plot()
plt.show()

# density plot of residuals
residuals.plot(kind='kde')
plt.show()
# summary stats of residuals
print(residuals.describe())

"""## Deep learning"""

def create_time_series_dataset_with_lags(df, threshold_date, lag_observation=5, forecast_horizon=1):
    """
    Create X and Y matrices for time series modeling with lag observations and split into train and test sets.

```

```
    Parameters:
        df (pd.DataFrame): DataFrame containing the time series data with columns: revenue, competitor_sales,
                           newsletter, Total_Media_Spend, and DATE.
        threshold_date (str): The date to split the data into train and test sets (format: 'YYYY-MM-DD').
        lag_observation (int): The number of lag observations to use. Default is 5.
        forecast_horizon (int): The number of steps ahead to forecast revenue. Default is 1.

    Returns:
        X_train (np.ndarray): 3-dimensional array containing input features with lag observations for the train set.
        Y_train (np.ndarray): 1-dimensional array containing target values (revenue) for the train set.
        X_test (np.ndarray): 3-dimensional array containing input features with lag observations for the test set.
        y_test (np.ndarray): 1-dimensional array containing target values (revenue) for the test set.
    """
    # Convert threshold_date to Timestamp object
    threshold_date = pd.to_datetime(threshold_date)


    # Get the 'revenue' time series and other features
    revenue = df['revenue'].values
    competitor_sales = df['competitor_sales'].values
    newsletter = df['newsletter'].values
    media1_S = df['media1_S'].values

    X_train, Y_train, X_test, Y_test = [], [], [], []

    for i in range(len(df) - lag_observation - forecast_horizon + 1):
        # Input sequence with lag observations for all features
        x_sequence = np.column_stack((
            revenue[i:i + lag_observation],
            competitor_sales[i:i + lag_observation],
            newsletter[i:i + lag_observation],
            media1_S[i:i + lag_observation]
        ))

        if df['DATE'][i + lag_observation - 1] <= threshold_date:
            # Add to train set
            X_train.append(x_sequence)
            Y_train.append(revenue[i + lag_observation + forecast_horizon - 1])
        else:
            # Add to test set
            X_test.append(x_sequence)
            Y_test.append(revenue[i + lag_observation + forecast_horizon - 1])

    X_train = np.array(X_train)
    Y_train = np.array(Y_train)
    X_test = np.array(X_test)
    Y_test = np.array(Y_test)

    return X_train, Y_train, X_test, Y_test

threshold_date = '2016-05-30'
lag_observation =3
forecast_horizon = 1

X_train, Y_train, X_test, Y_test = create_time_series_dataset_with_lags(df2, threshold_date, lag_observation, forecast_horizon)

# Normalize the input data
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train.reshape(-1, X_train.shape[-1])).reshape(X_train.shape)
X_test_scaled = scaler.transform(X_test.reshape(-1, X_test.shape[-1])).reshape(X_test.shape)
```

```python

# Define the model
model = Sequential()
model.add(LSTM(128, activation='relu', return_sequences=True, input_shape=(X_train.shape[1], X_train.shape[2])))
model.add(LSTM(64, activation='relu', return_sequences=True))
model.add(Dropout(0.2))
model.add(Flatten())
model.add(Dense(128))
model.add(Dense(64))
model.add(Dense(64))
model.add(Dense(1))

model.compile(optimizer='adam', loss='mean_squared_error')

# Train the model
history = model.fit(X_train_scaled, Y_train, epochs=200, batch_size=16, verbose=0, shuffle=False)

# Make predictions
Y_pred = model.predict(X_test_scaled)

rmse = mean_squared_error(Y_test, Y_pred, squared = False)
print('RMSE for test dataset = %.5f' % rmse)

df_predicted = pd.DataFrame({'Actual Revenue': Y_test.flatten(), 'Predicted Revenue': Y_pred.flatten()})
print(df_predicted)

with plt.style.context('seaborn-bright', after_reset=True):
  plt.figure(figsize=(6,4))
  plt.plot(Y_test, label='Original')
  plt.plot(Y_pred, label='Predicted')
  plt.legend()
  plt.xlabel('Timestep')
  plt.ylabel('Revenue')
  plt.grid(color='gray', linestyle='-', linewidth=0.1)
  plt.show()

"""# Part 2: Bayesian methods"""

# Define the split date based on the given threshold (e.g., '2016-05-30' in this case)
split_date = '2016-05-30'

# Split the data into train and test sets
train = df[df.DATE <= split_date]
test = df[df.DATE > split_date]

# Extract the target variable 'revenue' from the training and test data
y_train = train['revenue']
y_test = test['revenue']

# Extract the input features from the training and test data
x_train = train.drop(['revenue', 'DATE', 'X'], axis=1)
x_test = test.drop(['revenue', 'DATE', 'X'], axis=1)

# Scale the data
scaler = StandardScaler()
X_train = scaler.fit_transform(x_train)
X_test = scaler.transform(x_test)

# Define different priors for each feature's weights
priors = {
    'media1_S': {
        'mu': 0,
        'sd': 10
```

```python
    },
    'media2_S': {
        'mu': 0,
        'sd': 10
    },
    'media3_S': {
        'mu': 0,
        'sd': 10
    },
    'competitor_sales': {
        'mu': 0,
        'sd': 10
    },
    'newsletter': {
        'mu': 0,
        'sd': 10
    }
}

def contruct_bnn(x_input, y_input):
    # Using the context manager to build the model
    with pm.Model() as bnn:
        # Data input
        x_data = pm.MutableData('x_data', x_input)
        y_data = pm.MutableData('y_data', y_input)

        # Model structure
        num_hidden = 5

        # Priors for the weights
        weights = {}

        for feature in x_train.columns:
            weights[feature] = pm.Normal(f'w_{feature}', mu=priors[feature]['mu'], sigma=priors[feature]['sd'], shape=num_hidden)


        b_1 = pm.Normal('b_1', mu=0, sigma=10, shape=num_hidden)
        w_out = pm.Normal('w_out', mu=0, sigma=10, shape=num_hidden)
        b_out = pm.Normal('b_out', mu=0, sigma=10, shape=1)

        # Neural network architecture
        act_1 = pm.math.tanh(pm.math.dot(x_data, pm.math.stack([weights[feature] for feature in x_train.columns])) + b_1)

        act_out = pm.Deterministic('act_out', pm.math.dot(act_1, w_out) + b_out)

        # Likelihood of the data (observed)
        sigma = pm.HalfNormal('sigma', sigma=1)
        likelihood = pm.Normal('Y_obs', mu=act_out, sigma=1, observed=y_data, total_size=y_input.shape[0])

        return bnn

bnn = contruct_bnn(X_train, y_train)
# pm.model_to_graphviz(bnn)

# Specifying the MCMC algorithm (NUTS)
with bnn:
    step = pm.NUTS()
    trace = pm.sample(1000, tune=1000, step=step, chains=4, random_seed=42)

pm.plot_trace(trace, figsize=(14,10), legend=True, compact=False)
plt.show()
```

```python

pm.set_data({"x_data": X_test, "y_data": y_test}, model=bnn)

# Generate posterior samples.
ppc_test = pm.sample_posterior_predictive(trace, model=bnn)

# Compute the point prediction by taking the mean
y_test_pred = ppc_test.posterior_predictive['Y_obs'].mean(dim=['chain', 'draw'])

# y_mean = y_test_pred.mean(axis=0)
rmse = mean_squared_error(y_test, y_test_pred, squared = False)
print('RMSE for test dataset = %.5f' % rmse)

with plt.style.context('seaborn-bright', after_reset=True):
  plt.figure(figsize=(6,4))
  plt.plot(y_test.values, label='Original')
  plt.plot(y_test_pred.values, label='Predicted')
  plt.legend()
  plt.xlabel('Timestep')
  plt.ylabel('Revenue')
  plt.grid(color='gray', linestyle='-', linewidth=0.1)
  plt.show()

pm.plot_trace(trace, figsize=(14,10), legend=True, compact=False)
plt.show()

pm.summary(trace, round_to=2)

# Plotting the ACF plot for the MCMC chains
pm.plot_autocorr(trace)
plt.show()

az.plot_posterior(trace)

"""## Prior Choice"""

# Define different priors for each feature's weights
priors = {
    'media1_S': {
        'mu': 2,
        'sd': 0.1
    },
    'media2_S': {
        'mu': 8,
        'sd': 0.1
    },
    'media3_S': {
        'mu': 0,
        'sd': 10
    },
    'competitor_sales': {
        'mu': 0.15,
        'sd': 0.05
    },
    'newsletter': {
        'mu': 1,
        'sd': 0.1
    }
}

def contruct_bnn(x_input, y_input):
    # Using the context manager to build the model
    with pm.Model() as bnn:
```

```python
        # Data input
        x_data = pm.MutableData('x_data', x_input)
        y_data = pm.MutableData('y_data', y_input)

        # Model structure
        num_hidden = 1

        # Priors for the weights
        weights = {}

        for feature in x_train.columns:
            weights[feature] = pm.Normal(f'w_{feature}', mu=priors[feature]['mu'], sigma=priors[feature]['sd'],
            ↪ shape=num_hidden)


        b_1 = pm.Normal('b_1', mu=0, sigma=10, shape=num_hidden)
        w_out = pm.Normal('w_out', mu=0, sigma=10, shape=num_hidden)
        b_out = pm.Normal('b_out', mu=0, sigma=10, shape=1)

        # Neural network architecture
        act_1 = pm.math.tanh(pm.math.dot(x_data, pm.math.stack([weights[feature] for feature in
        ↪ x_train.columns])) + b_1)

        act_out = pm.Deterministic('act_out', pm.math.dot(act_1, w_out) + b_out)

        # Likelihood of the data (observed)
        sigma = pm.HalfNormal('sigma', sigma=1)
        likelihood = pm.Normal('Y_obs', mu=act_out, sigma=1, observed=y_data, total_size=y_input.shape[0])

        return bnn

bnn = contruct_bnn(X_train, y_train)
# pm.model_to_graphviz(bnn)

# Specifying the MCMC algorithm (NUTS)
with bnn:
    step = pm.NUTS()
    trace = pm.sample(1000, tune=1000, step=step, chains=4, random_seed=42)

pm.set_data({"x_data": X_test, "y_data": y_test}, model=bnn)

# Generate posterior samples.
ppc_test = pm.sample_posterior_predictive(trace, model=bnn)

# Compute the point prediction by taking the mean
y_test_pred = ppc_test.posterior_predictive['Y_obs'].mean(dim=['chain', 'draw'])

# y_mean = y_test_pred.mean(axis=0)
rmse = mean_squared_error(y_test, y_test_pred, squared = False)
print('RMSE for test dataset = %.5f' % rmse)
```