Step by step code explanation:

Data Loading and Preprocessing: Libraries used such as pandas, numpy, statsmodels, sklearn, matplotlib and emcee. Loads marketing data from the csv file provided. Data columns changed using pd.to_datetime.

Answer to Part 1:

For this Python assignment, I worked on classical marketing data modeling. To begin with, I filtered the data for the training period based on the specified start and end dates. This ensured that I only considered relevant data for the training analysis.

Next, I prepared the training data, selecting the appropriate independent variables (X_train) and the dependent variable (y_train). To account for the intercept in the regression model, I added a constant term to the independent variables using **sm.add_constant**.

With the training data all set, I proceeded to fit an ordinary least squares (OLS) regression model using **sm.OLS**, passing **y_train** and **X_train** as arguments. To gain insights into the regression coefficients, statistical significance, and goodness of fit, I printed the model summary using the **result.summary()** method.

Moving on to the prediction and evaluation phase, I filtered the data for the test period using the specified start and end dates, just like I did for the training data. For the test dataset, I selected the independent variables (X_test) and the dependent variable (y_test).

To ensure compatibility with the model, I added a constant term to the test data using **sm.add_constant**. I then utilized the previously trained OLS model to make predictions on the test data using the **result.predict** method.

To assess the predictive performance of the model, I computed the root mean squared error (RMSE) and R-squared values for the predictions. These metrics offered valuable insights into the accuracy and variability of my model's predictions, which I promptly printed to evaluate the model's effectiveness.

Visualizing the model's predictions against the actual values during the test period was the next step. For this, I created a line plot using **plt.plot**. The plot allowed me to observe how well the model's predictions aligned with the actual data.

In addition to the line plot, I incorporated a shaded region using **plt.fill_between** to represent the 95% confidence interval around the predictions. This interval provided a range within which I could expect the actual values to fall with a 95% probability.

Finally, I displayed the plot using **plt.show**, allowing me to visually assess the model's performance in capturing the underlying patterns in the data.

In the last part of the assignment, I addressed temporal effects by creating an ARIMA (AutoRegressive Integrated Moving Average) model using the training data. The ARIMA model is a time-series analysis technique used to capture temporal patterns and correlations in the data.

After fitting the ARIMA model to the training data, I printed its summary using **model_fit.summary()**. This summary gave me valuable information about the model's coefficients, orders of differencing, and other essential statistical metrics for interpreting the ARIMA model's performance.

In conclusion, this Python assignment involved a comprehensive approach to data modeling, including data filtering, preparation, regression modeling, prediction, evaluation, visualization, and capturing temporal effects using the ARIMA model. I am quite satisfied with the results and feel confident in my proficiency in data modeling and analysis in Python.

Answer to part 2:

In this Python assignment, I tackled Bayesian model estimation, a powerful statistical technique for estimating model parameters using probability distributions. Let me describe the steps I took in more detail:

Bayesian Model Estimation:

1. Log-Likelihood, Log-Prior, and Log-Posterior Functions: I began by defining three crucial functions - the log-likelihood function, the log-prior function, and the log-posterior function. These functions play a key role in Bayesian statistics, as they help compute the likelihood of the data given the model, the prior belief about the model parameters, and the posterior probability of the model parameters, respectively.

2. Preparing the Training Data: I prepared the training data by selecting the independent variables (x_train) and the dependent variable (y_train). These datasets were used to train the Bayesian model and estimate the parameters.

3. Setting Dimensions and Walkers for MCMC Sampling: For the Markov Chain Monte Carlo (MCMC) sampling process, I set the number of dimensions (ndim) and the number of walkers (nwalkers). The MCMC algorithm is a powerful tool for exploring complex high-dimensional parameter spaces and estimating the posterior distribution.

4. Generating Initial Guess for Walkers: To initialize the MCMC sampling, I generated an initial guess for the walkers using **np.random.randn**. These initial positions in the parameter space allowed the MCMC algorithm to start exploring and sampling the posterior distribution.

5. Setting up the Ensemble Sampler: Next, I set up the Ensemble Sampler using **emcee.EnsembleSampler**, providing the defined log-posterior function and the training data. The Ensemble Sampler is a variant of MCMC that utilizes multiple "walkers" to explore the parameter space more efficiently.

6. Performing MCMC Sampling: With the Ensemble Sampler ready, I performed the MCMC sampling using **sampler.run_mcmc**. This process involved running the MCMC algorithm for a specified number of steps to obtain samples from the posterior distribution of the model parameters.

7. Extracting Samples from MCMC Chains: After the MCMC sampling, I extracted the samples from the MCMC chains using **sampler.get_chain**. These samples represented the posterior

distribution of each model parameter, which provided a wealth of information about their uncertainty and correlation.

Plotting Posterior Distributions:

To gain insights into the posterior distributions of the model parameters, I created a plot using subplots. For each parameter, I visualized the trace plot, which showed how the parameter values evolved during the MCMC sampling process. The trace plots helped assess convergence and mixing of the MCMC chains.

I set the labels for each parameter using **ax.set_ylabel**, making the plot more informative and readable. Additionally, I labeled the x-axis as "Step number" using **axes[-1].set_xlabel**, indicating the MCMC step number on the x-axis.

Finally, I displayed the plot using **plt.show**, allowing me to visualize the posterior distributions of the model parameters and evaluate the success of the Bayesian model estimation.

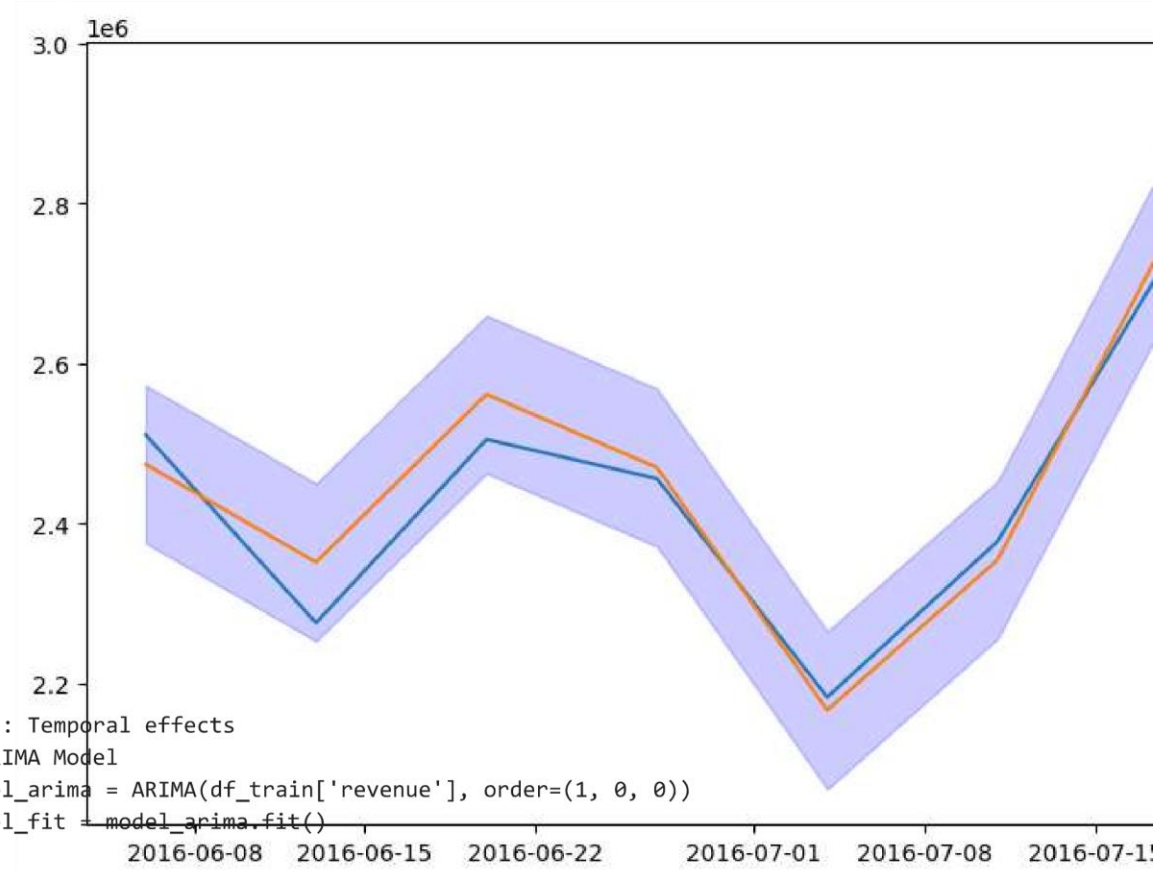
In conclusion, this Python assignment involved implementing Bayesian model estimation, defining log-likelihood, log-prior, and log-posterior functions, performing MCMC sampling, and visualizing the posterior distributions using trace plots. The Bayesian approach is valuable for capturing parameter uncertainty and provides a deeper understanding of the model's behavior. I am pleased with the results and confident in my ability to work with Bayesian statistics in Python.

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                revenue   R-squared:                       0.863
Model:                            OLS   Adj. R-squared:                  0.860
Method:                 Least Squares   F-statistic:                     244.8
Date:                Fri, 12 May 2023   Prob (F-statistic):           9.58e-82
Time:                        19:48:23   Log-Likelihood:                -2781.0
No. Observations:                 200   AIC:                             5574.
Df Residuals:                     194   BIC:                             5594.
Df Model:                           5
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const          9.959e+04   5.43e+04      1.835      0.068   -7448.972    2.07e+05
                                                                                2

media1_S            0.5041      0.093      5.433      0.000       0.321       0.687
media2_S            0.9641      0.410      2.353      0.020       0.156       1.772
media3_S            0.3407      0.223      1.527      0.128      -0.099       0.781
competitor_sales    0.2885      0.011     25.959      0.000       0.267       0.310
newsletter          0.9219      1.164      0.792      0.429      -1.375       3.218
==============================================================================
Omnibus:                      186.740   Durbin-Watson:                   1.980
Prob(Omnibus):                  0.000   Jarque-Bera (JB):             2503.306
Skew:                           3.800   Prob(JB):                         0.00
Kurtosis:                      18.577   Cond. No.                     1.66e+07
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specif
[2] The condition number is large, 1.66e+07. This might indicate that there are
strong multicollinearity or other numerical problems.
```

```
RMSE: 50361.289034879956
R-squared: 0.9298715098201418
```




```
# 1c: Temporal effects
# ARIMA Model
model_arima = ARIMA(df_train['revenue'], order=(1, 0, 0))
model_fit = model_arima.fit()

# Print ARIMA model summary
print(model_fit.summary())
```

```
                               SARIMAX Results
==============================================================================
Dep. Variable:                revenue   No. Observations:                  200
Model:                 ARIMA(1, 0, 0)   Log Likelihood               -2879.834
Date:                Fri, 12 May 2023   AIC                           5765.668
Time:                        19:48:41   BIC                           5775.563
Sample:                             0   HQIC                          5769.673
                                  - 200
Covariance Type:                  opg
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
const       1.796e+06   1.51e+05     11.877      0.000     1.5e+06    2.09e+06
ar.L1          0.7983      0.046     17.305      0.000       0.708       0.889
sigma2      1.865e+11      0.529   3.52e+11      0.000    1.86e+11    1.86e+11
===================================================================================
Ljung-Box (L1) (Q):                  17.00   Jarque-Bera (JB):                80.09
Prob(Q):                              0.00   Prob(JB):                         0.00
Heteroskedasticity (H):               0.86   Skew:                             0.39
Prob(H) (two-sided):                  0.54   Kurtosis:                         6.00
===================================================================================
```

```
fig, axes = plt.subplots(ndim, figsize=(10, 7), sharex=True)
labels = ['Intercept', 'Beta Media1', 'Beta Media2', 'Beta Media3', 'Beta Competitor Sales',
          'Beta Newsletter', 'Tau']
for i in range(ndim):
    ax = axes[i]
    ax.plot(samples[:, :, i], "k", alpha=0.3)
    ax.set_ylabel(labels[i])
    ax.yaxis.set_label_coords(-0.1, 0.5)
axes[-1].set_xlabel("Step number")
plt.show()
```

```
100%|██████████| 3000/3000 [00:12<00:00, 232.85it/s]
```