Answer 1

Assuming that x1, x2, x3 and y represents radio, tv, internet and sales respectively. The data is generated for the 3 independent variables and the response viable is calculated. The model is then built on the train data and checked against the test data. Here are the results **(Code attached in Appendix) -**
Linear Regression MSE: 74.56753771040981

Linear Regression Coefficients: [2.19957962 8.38793456 12.19228899]

After checking the predicted data of the test set against the true values, we can see that we received a mean_squared_error score of approximately 74.567.

From the coefficients we can see that all the predictor variables have a positive impact on the sales. Internet had the most effect on the sales while the radio had the least.

A coefficient of 2.19 shows 2.19 units of change in the sales dependent variable if there is a one-unit change in radio variable, holding the other variables constant.

A coefficient of 8.38 shows 8.38 units of change in the sales dependent variable if there is a one-unit change in tv variable, holding the other variables constant.

A coefficient of 12.19 shows 12.19 units of change in the sales dependent variable if there is a one-unit change in internet variable, holding the other variables constant.

Random Forest Regression MSE: 27.467311656382822

We can see that using an ensemble method like random forest works better on the unseen test set as it has a lower mean_squared_error score of 27.467.

Answer 2

True Value: 191.315

True Value (calculated using scipy library): 190.9617286539798

Estimated Value: 191.02660192359843

Confidence Interval: [187.37062903357327, 194.6825748136236]

We can clearly see our estimated value lie close to the true value calculated and also in the 95% confidence interval. Hence, our Monte Carlo algorithm, which uses a lot of points to first calculate the mean and then area of the defined space, is sufficient in

estimating the value. These values in code are subject to change as no seed is set. (**Code attached in Appendix**)

Answer 3

This variant of cross-validation that involves selecting a subset of predictors that affects response variable has its own set of limitations like overfitting and high bias.

- If we choose only a set of good predictors, it might lead to lowering the complexity of the model too much. When the model complexity goes down, the bias increases.
- The data available to us is small, which means less data points for the training set. The trained model will work great on seen/train data but might cause issues in generalizing for unseen data. This leads to overfitting as training will lead to model learning unwanted patterns or information from the limited data.
- The response generated by the model by cross validation might have a high variance. This can happen when the model trained in each iteration of the cross-validation might not have enough data in the train set to learn relevant information about the data points in test set.

Hence, this model is not feasible and more methods needs to be brought in to provide a better prediction. The true prediction error will have a high bias as a lot of data points will not be available , thus, making the model simple. To improve the results, we could include things like a better model selection or increasing data using techniques like PCA or sampling.

Answer 4

$$H=\{f(x, \theta); \theta \in \Theta\}$$

The hypothesis class – H, consists of all the pdfs in the Gamma distribution with ($\alpha > 0$) and ($\beta > 0$). α defines the shape parameter while β defines the rate.

θ represents the parameter vector (α,β). Θ represents the parameter space, containing every possible value of θ provided that both α,β are greater than 0.

$$\Theta=\{(\alpha,\beta):\alpha>0,\beta>0\}$$

Therefore, H or the hypothesis class is defined as a collection of different Gamma distributions, where every distribution is represented by f(x, θ)

$$H=\{f(x, \theta): \theta \in \Theta\}=\{f(x;\alpha,\beta):\alpha>0,\beta>0\}$$

Answer 5

From the definitions we know:

$Loss_D(g) = E_{Z\sim D} L(g,Z)$ — ① The expected loss of the classifier

$Loss_T(g) = \frac{1}{m}\sum_{i=1}^{m} L(g,z_i)$ — ② empirical loss over a given sample

Using the second definition

$$E_T\left(loss_T(g)\right) = E_T\left(\frac{1}{m}\sum_{i=1}^{m} L(g,z_i)\right)$$

$$= \frac{1}{m}\sum_{i=1}^{m} E_T L(g,z_i)$$

we know that T can be defined as the over a sample where $T = (z_1, z_2, z_3 \ldots z_m)$ and these samples will be part of the unknown distribution D

$$\Rightarrow \frac{1}{m}\sum_{i=1}^{m} E_{Z\sim D} L(g,z) \Rightarrow E_{Z\sim D}\left(\frac{1}{m}\sum_{i=1}^{m} L(g,z)\right)$$

$$\Rightarrow E_{Z\sim D} L(g,z) \Rightarrow Loss_D(g)$$

hence, $E_T Loss_T(g) = Loss_D(g)$

Answer 6

a. (**Code attached in Appendix**)

Model	β0	β1
Model 1	1.8	0
Model 2	0	0.6

b. (**Code attached in Appendix**)

Model	squared error loss	absolute error loss	L1.5 loss
Model 1	0.56	0.64	0.5849
Model 2	1.64	1.16	1.36348

c.

From the above table we can see that the mean squared error loss, absolute error loss, and L1.5 loss of model1 is smaller than Model2. Hence, Model1 generalizes better than Model2. Therefore, Model1 must be selected.

Answer 7

a) The data was read and 2 different data frames were created. Here we checked the number of unique values in x2 column and decided to use one-hot encoding to divide and create additional columns x2_1, x2_2, x2_3 for creating a non-ordinal set as more than 2 unique values were present. The new columns like x2_1 contain values like TRUE/FALSE to simulate whether in the particular row the value for x2 column was either 1 or not. The ordinal set contains data like 1,2,3 in the x2 column.

b) (**Code attached in Appendix**)

Ordered mean squared error: 1.2685717076212815

Un-Ordered mean squared error: 3.757360926169872e-30

From the mean squared error scores, we can clearly see that after 10-Fold Cross-Validation, the MSE of un-ordered process is a lot smaller and closer to 0. This means that the column x2 can be considered as unordered categorical column as it performs better in generalizing the data.

Importing Libraries

In [1]:

```
import random
import pandas as pd
import numpy as np
from scipy.integrate import quad
from sklearn.linear_model import LinearRegression
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_squared_error
from sklearn.metrics import mean_absolute_error
from sklearn.model_selection import cross_val_score, KFold
```

Question 1

In [2]:

```
np.random.seed(1)
x1_train = np.random.gamma(shape=1, scale=1, size=1000)
x2_train = np.random.gamma(shape=1, scale=1, size=1000)
x3_train = np.random.gamma(shape=1, scale=1, size=1000)
w_train = np.random.normal(loc=0, scale=2, size=1000)
y_train = (0.5*x1_train) + (3*x2_train) + (5*x3_train) + (5*x2_train*x3_train) + (2*x1_t
rain*x2_train*x3_train) + w_train

np.random.seed(2)
x1_test = np.random.gamma(shape=1, scale=1, size=1000)
x2_test = np.random.gamma(shape=1, scale=1, size=1000)
x3_test = np.random.gamma(shape=1, scale=1, size=1000)
w_test = np.random.normal(loc=0, scale=2, size=1000)
y_test = (0.5*x1_test) + (3*x2_test) + (5*x3_test) + (5*x2_test*x3_test) + (2*x1_test*x2
_test*x3_test) + w_test
```

In [3]:

```
train_df = pd.DataFrame({"x1": x1_train, "x2": x2_train, "x3": x3_train, "y": y_train})
x_train = train_df[["x1", "x2", "x3"]]
y_train = train_df["y"]

test_df = pd.DataFrame({"x1": x1_test, "x2": x2_test, "x3": x3_test, "y": y_test})
x_test = test_df[["x1", "x2", "x3"]]
y_test = test_df["y"]
```

In [4]:

```
lr_model = LinearRegression()
lr_model.fit(x_train, y_train)
y_lr_pred = lr_model.predict(x_test)
lr_mse = mean_squared_error(y_test, y_lr_pred)
print("Linear Regression MSE:", lr_mse)
print("Linear Regression Coefficients:", lr_model.coef_)
```

```
Linear Regression MSE: 74.56753771040981
Linear Regression Coefficients: [ 2.19957962  8.38793456 12.19228899]
```

In [5]:

```
rf_model = RandomForestRegressor(n_estimators = 500)
rf_model.fit(x_train, y_train)
y_rf_pred = rf_model.predict(x_test)
rf_mse = mean_squared_error(y_test, y_rf_pred)
print("Random Forest Regression MSE:", rf_mse)
```

```
Random Forest Regression MSE: 27.467311656382822
```

Question 2

In [6]:

```
def function(x):
    return 3 + (x**2) - (2*np.sin(x))

a, b = 1, 8
true_value, error = quad(function, a, b)
print("True Value:", true_value, u"\u00B1", error)
```

```
True Value: 189.9617286539798 ± 2.1089988494924473e-12
```

In [15]:

```
samples = np.random.uniform(low=a, high=b, size=10000)
function_samples = [function(sample) for sample in samples]

estimated_value = np.mean(function_samples) * (b-a)
print("Estimated Value: ", estimated_value)
```

```
Estimated Value:  190.97185353936558
```

In [8]:

```
sd = 0
sd = sum([(result - estimated_value)**2 for result in samples])
sd = (sd/(10000 - 1))**0.5

lower = estimated_value - 1.96*sd/(10000**0.5)
upper = estimated_value + 1.96*sd/(10000**0.5)
confidence_interval = [lower, upper]
print("Confidence Interval: ", confidence_interval)
```

```
Confidence Interval:  [184.94448784143083, 192.16113042048005]
```

Question 6

In [9]:

```
df = pd.DataFrame({"x1": [0, 1, 2, 3, 4], "y": [1, 2, 3, 2, 1]})
```

In [10]:

```
lr_model_1 = LinearRegression()
lr_model_1.fit(df[["x1"]], df["y"])
y_pred_1 = lr_model_1.predict(df[["x1"]])
print("model 1 intercept:", lr_model_1.intercept_)

lr_model_2 = LinearRegression(fit_intercept=False)
lr_model_2.fit(df[["x1"]], df["y"])
y_pred_2 = lr_model_2.predict(df[["x1"]])
print("model 2 coefficients: ", lr_model_2.coef_)
```

```
model 1 intercept: 1.8
model 2 coefficients:  [0.6]
```

In [11]:

```
squared_error_1 = ((y_pred_1 - df["y"])**2).mean()
squared_error_2 = ((y_pred_2 - df["y"])**2).mean()
print("Average Squared Error for model_1:", squared_error_1, "model_2:", squared_error_2)

absolute_error_1 = (np.abs(y_pred_1 - df["y"])).mean()
absolute_error_2 = (np.abs(y_pred_2 - df["y"])).mean()
print("Average Absolute Error for model_1:", absolute_error_1, "model_2:", absolute_error
_2)
```

```
l_15_model_1 = (np.abs(y_pred_1 - df["y"])**1.5).mean()
l_15_model_2 = (np.abs(y_pred_2 - df["y"])**1.5).mean()
print("L1.5 loss for model_1:", l_15_model_1, "model_2:", l_15_model_2)
```

```
Average Squared Error for model_1: 0.5600000000000002 model_2: 1.64
Average Absolute Error for model_1: 0.6400000000000001 model_2: 1.1600000000000001
L1.5 loss for model_1: 0.5849006163624495 model_2: 1.36348016266711
```

Question 7

In [12]:

```
#Reading Data
df = pd.read_csv(r"C:\Users\jaske\Desktop\studies\OneDrive\Statistical Methods for Data Science\Assignments\Assignment 1\data.csv")
df["x2"] = df["x2"].astype(int)
df["x2"].unique()
```

Out[12]:

```
array([1, 2, 3])
```

In [13]:

```
#Ordinal Data
ordinal_df = df.copy()
ordinal_model = LinearRegression()
kf = KFold(n_splits=10, shuffle=True, random_state=42)
mse_list = -cross_val_score(ordinal_model, ordinal_df.drop(columns=["y"]), ordinal_df["y"], cv=kf, scoring="neg_mean_squared_error")
mse = (mse_list).mean()
print("Ordered mean squared error:", mse)
```

```
Ordered mean squared error: 1.2685717076212815
```

In [14]:

```
#Non Ordinal Data
non_ordinal_df = pd.get_dummies(df, columns=["x2"])
non_ordinal_model = LinearRegression()
kf = KFold(n_splits=10, shuffle=True, random_state=42)
mse_list = -cross_val_score(non_ordinal_model, non_ordinal_df.drop(columns=["y"]), non_ordinal_df["y"], cv=kf, scoring="neg_mean_squared_error")
mse = np.mean(mse_list)
print("UnOrdered mean squared error:", mse)
```

```
UnOrdered mean squared error: 3.757360926169872e-30
```

References

Monte Carlo integration in Python. (2022, March 10). GeeksforGeeks. https://www.geeksforgeeks.org/monte-carlo-integration-in-python/ An Easy Guide to K-Fold Cross-Validation. (2020. November 4). Statology. https://www.statology.org/k-fold-cross-validation/