Lab Sheet 2

Implement and demonstrate Multiple Linear Regression for Brain Weights Prediction using sklearn Read the training data from a HeadBrain.CSV file.

https://www.kaggle.com/datasets/jemishdonda/headbrain/data

Import Libraries

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score
```

Read Dataset

```
# Load the dataset
df = pd.read_csv('/kaggle/input/head-brain/headbrain.csv')

# Display the first few rows of the dataframe
print(df.head())
```

```
   Gender  Age Range  Head Size(cm^3)  Brain Weight(grams)
0       1          1             4512                 1530
1       1          1             3738                 1297
2       1          1             4261                 1335
3       1          1             3777                 1282
4       1          1             4177                 1590
```

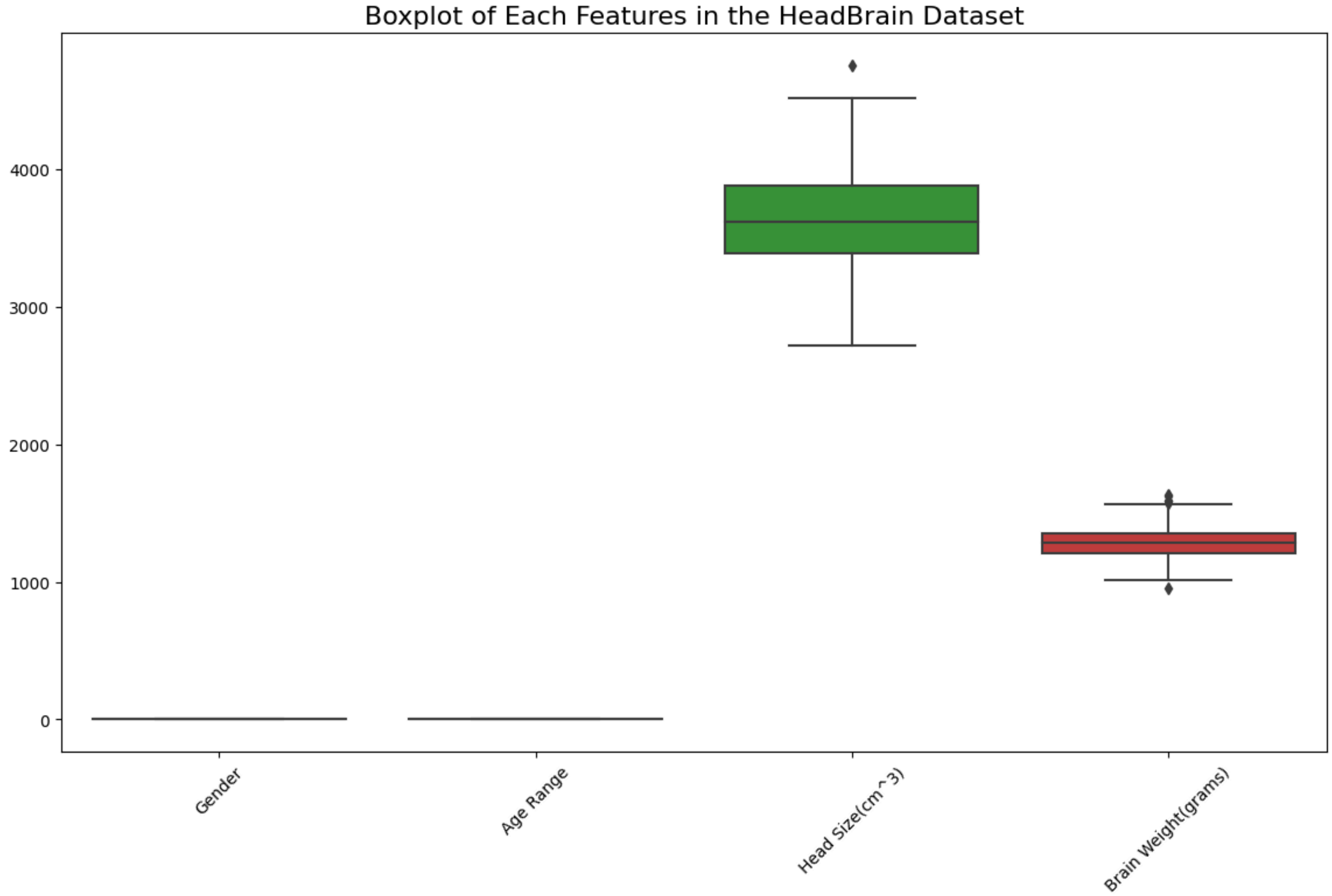
```
df.shape
```

```
(237, 4)
```

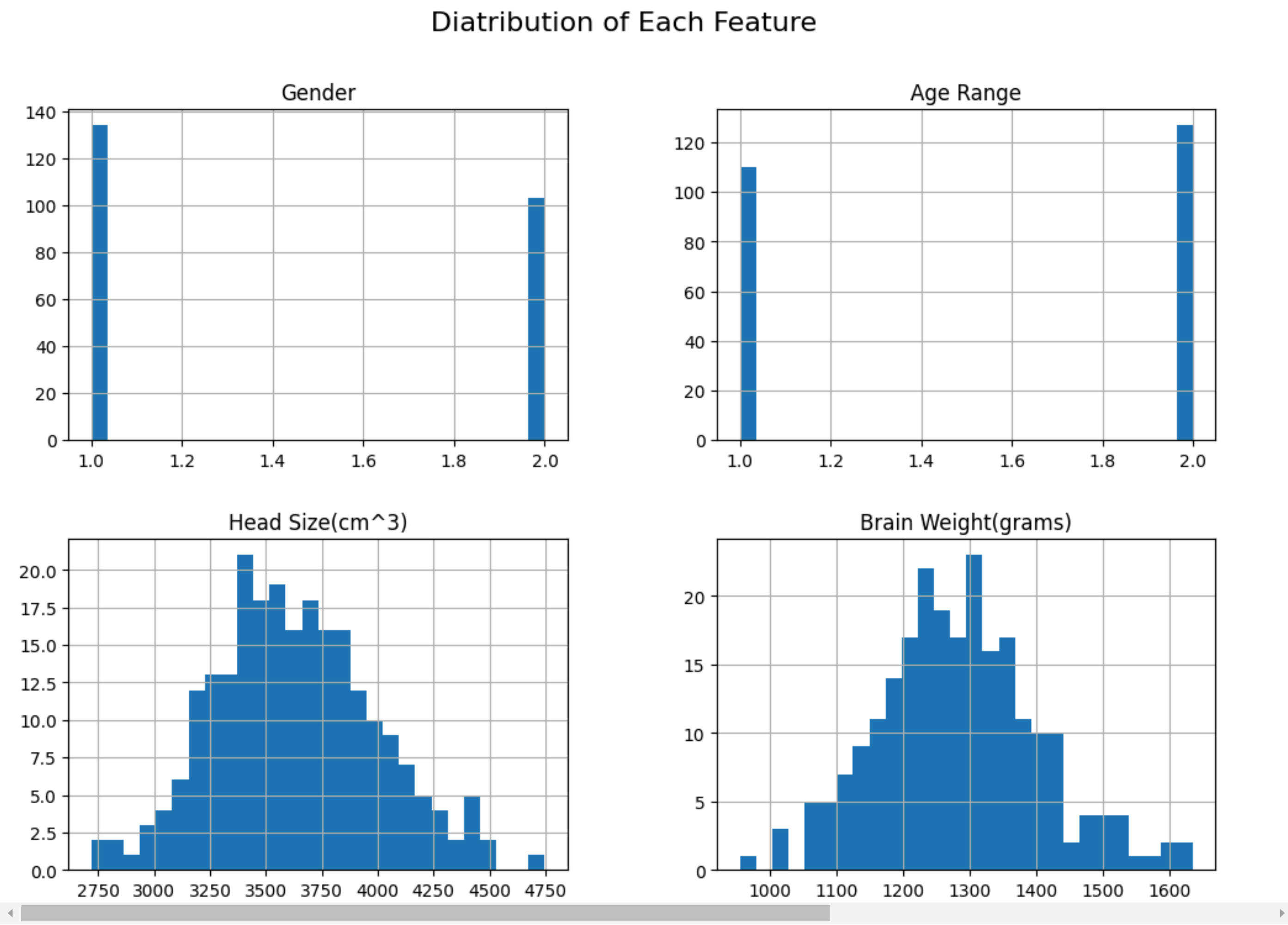
```
df.tail()
```

	Gender	Age Range	Head Size(cm^3)	Brain Weight(grams)
232	2	2	3214	1110
233	2	2	3394	1215
234	2	2	3233	1104
235	2	2	3352	1170
236	2	2	3391	1120

```
# Plot boxplot for each feature in the dataset
plt.figure(figsize=(14, 8))
sns.boxplot(data=df)
plt.title('Boxplot of Each Features in the HeadBrain Dataset', fontsize=16)
plt.xticks(rotation=45)
plt.show()
```

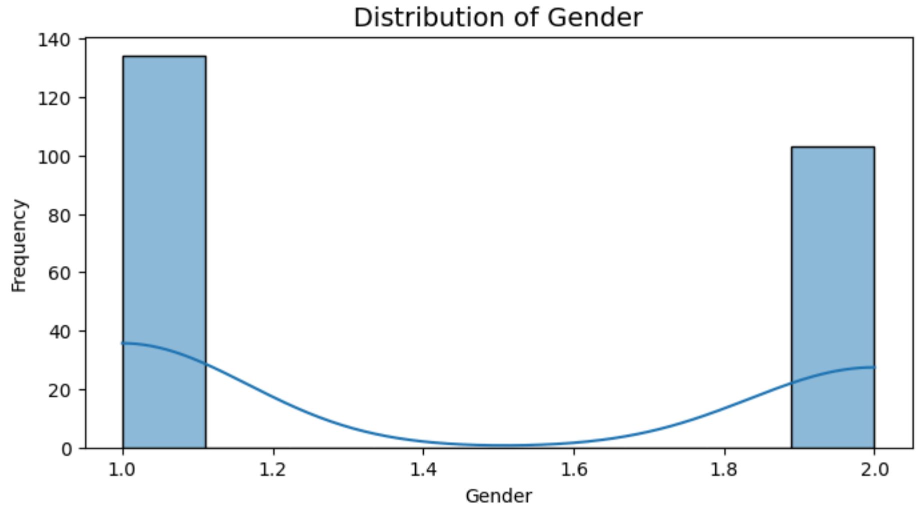



```
# Plot histograms for each features
df.hist(figsize=(12, 8), bins=28)
plt.suptitle('Diatribution of Each Feature', fontsize=16)
plt.show()
```

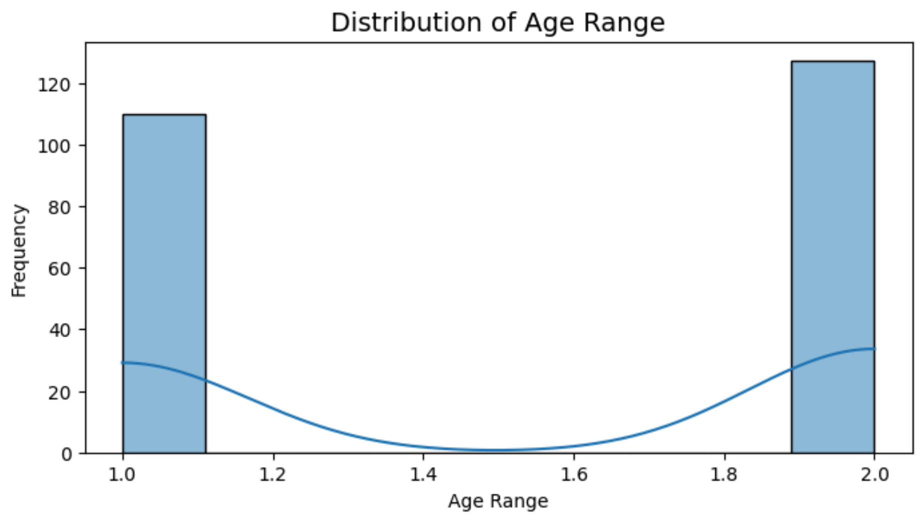



```
# Plot distribution plots with KDE for each feature
for column in df.columns:
    plt.figure(figsize=(8,4))
    sns.histplot(df[column], kde=True)
    plt.title(f'Distribution of {column}', fontsize=14)
    plt.xlabel(column)
    plt.ylabel('Frequency')
    plt.show()
```

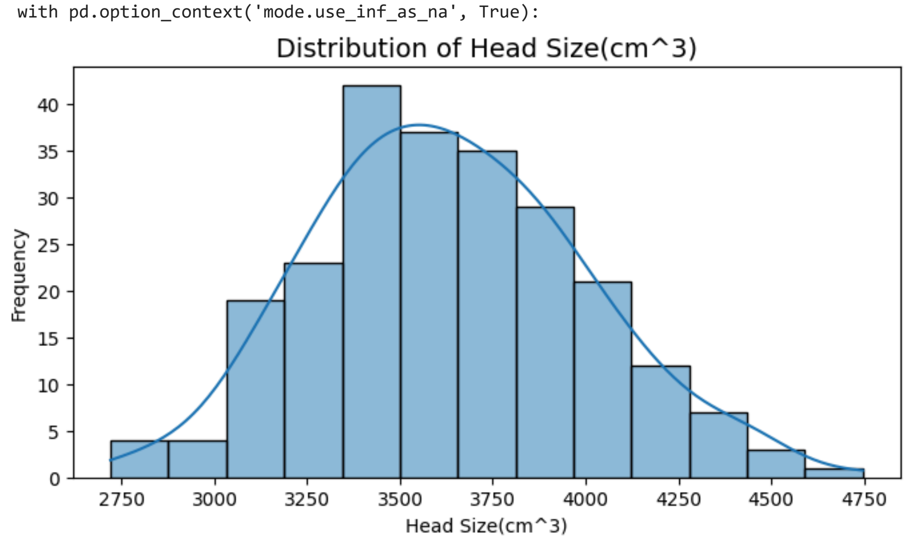
```
/opt/conda/lib/python3.10/site-packages/seaborn/_oldcore.py:1119: FutureWarning: use_inf_as_na option is deprecated and will be remove
  with pd.option_context('mode.use_inf_as_na', True):
```



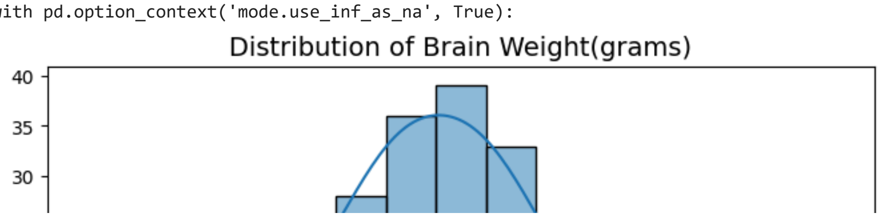

```
/opt/conda/lib/python3.10/site-packages/seaborn/_oldcore.py:1119: FutureWarning: use_inf_as_na option is deprecated and will be remove
  with pd.option_context('mode.use_inf_as_na', True):
```

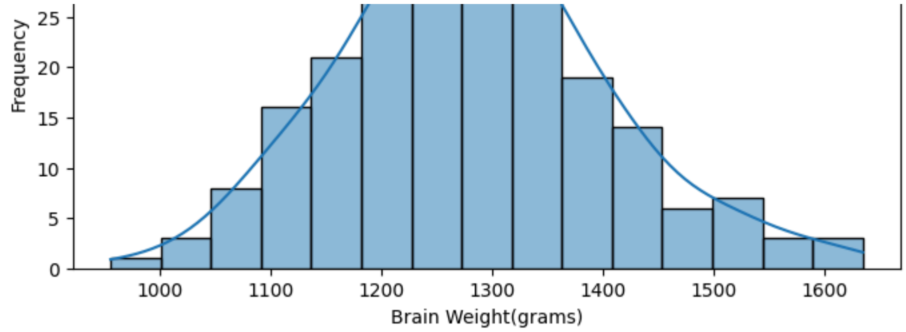



```
/opt/conda/lib/python3.10/site-packages/seaborn/_oldcore.py:1119: FutureWarning: use_inf_as_na option is deprecated and will be remove
  with pd.option_context('mode.use_inf_as_na', True):
```



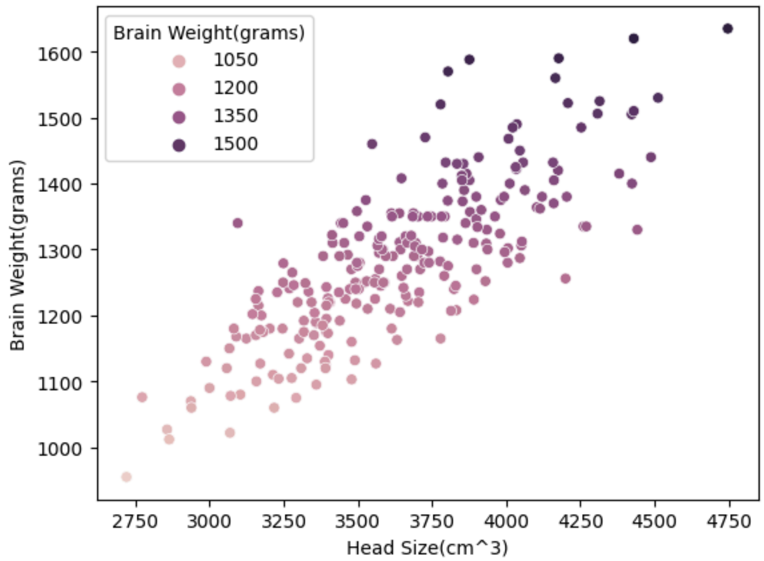

```
/opt/conda/lib/python3.10/site-packages/seaborn/_oldcore.py:1119: FutureWarning: use_inf_as_na option is deprecated and will be remove
  with pd.option_context('mode.use_inf_as_na', True):
```






```
sns.scatterplot(x='Head Size(cm^3)', y='Brain Weight(grams)', hue='Brain Weight(grams)', data=df)
```

<Axes: xlabel='Head Size(cm^3)', ylabel='Brain Weight(grams)'>



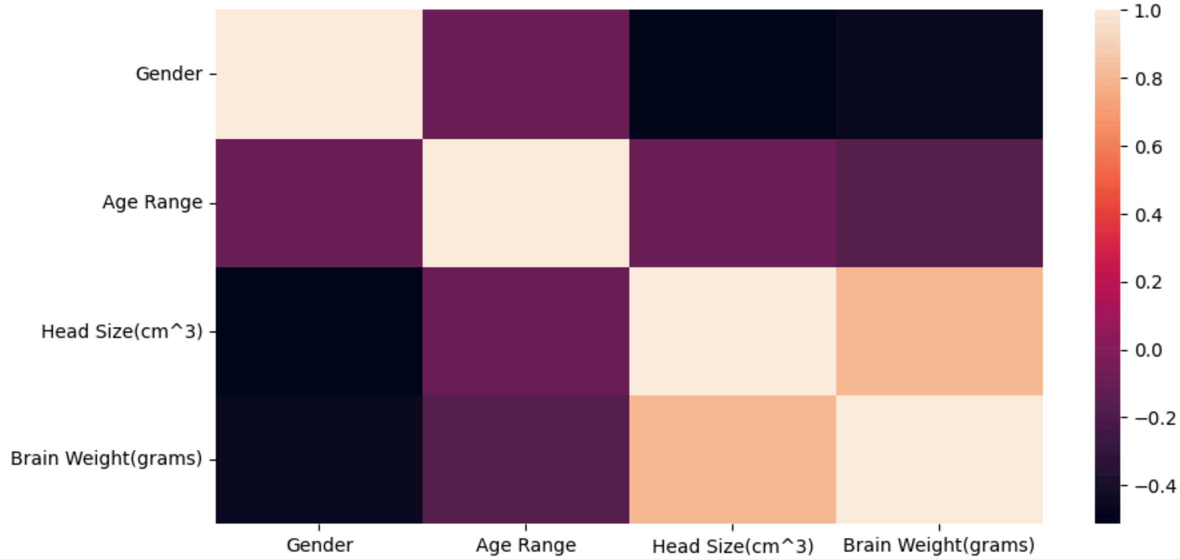

```
#Calculate the correlation matrix
##df.corr(): Computes the correlation matrix for the DataFrame, showing the correlation coefficients between variables.
tc=df.corr()
```

```
tc
```

	Gender	Age Range	Head Size(cm^3)	Brain Weight(grams)
Gender	1.000000	-0.088652	-0.514050	-0.465266
Age Range	-0.088652	1.000000	-0.105428	-0.169438
Head Size(cm^3)	-0.514050	-0.105428	1.000000	0.799570
Brain Weight(grams)	-0.465266	-0.169438	0.799570	1.000000

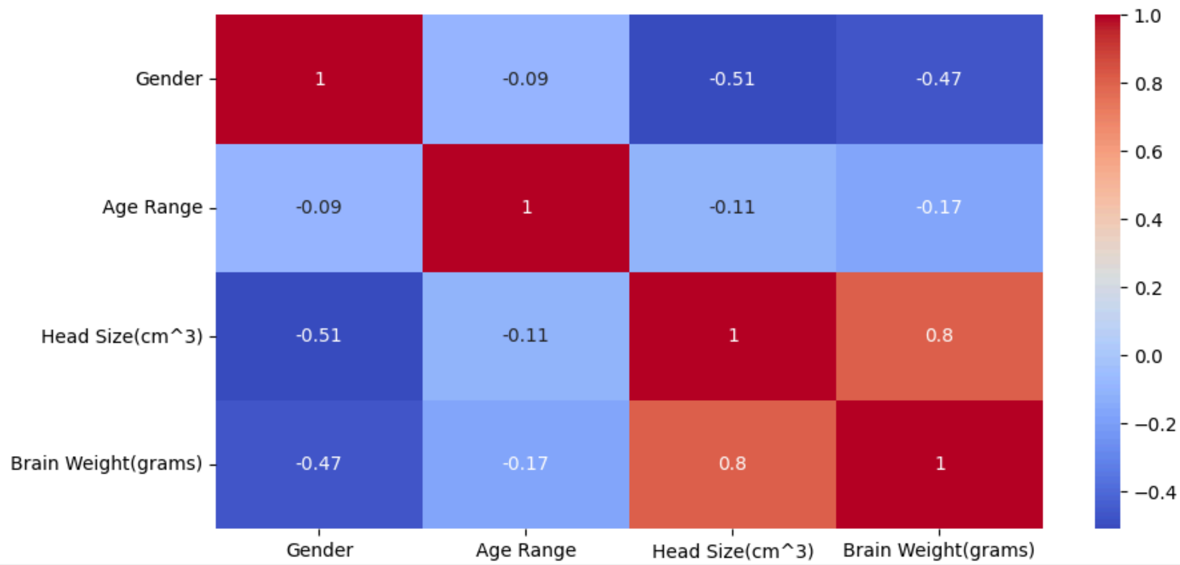
```
plt.figure(figsize=(10,5))
sns.heatmap(tc)
```

<Axes: >




```
#Visualize the correlation matrix using a heatmap
##sns.heatmap(): Creates a heatmap to visualize the correlation matrix.
##annot=True: Displays the correlation coefficient values in the heatmap.
##cmap='coolwarm': Sets the color map for the heatmap.
plt.figure(figsize=(10,5))
tc=df.corr().round(2)
sns.heatmap(tc,annot=True,cmap='coolwarm')
```

<Axes: >




```
# Identify and list the features with the highest positive and negative correlation with the target variable
target_correlation = tc['Brain Weight(grams)'].sort_values(ascending=False)

top_positive_correlations = target_correlation.head()

top_negative_correlations = target_correlation.tail()

print(" Positive Correlations:\n", top_positive_correlations)
print("\n Negative Correlations:\n", top_negative_correlations)
```

```
 Positive Correlations:
 Brain Weight(grams)    1.00
Head Size(cm^3)        0.80
Age Range             -0.17
Gender                -0.47
Name: Brain Weight(grams), dtype: float64

 Negative Correlations:
 Brain Weight(grams)    1.00
```

```
Head Size(cm^3)        0.80
Age Range             -0.17
Gender                -0.47
Name: Brain Weight(grams), dtype: float64
```

Prepare the data for training the Linear Regression model

```
#Select the features and the target variable
#df.drop(): Removes the target variable 'Brain Weight' from the features DataFrame X.
#df['Brain Weight(grams)']: Selects the target variable y.

# Select the features and the target variable
X=df.drop('Brain Weight(grams)',axis=1)
y=df['Brain Weight(grams)']
```

df.drop(): Removes the target variable 'MEDV' from the features DataFrame X.

df['Brain Weight(grams)']: Selects the target variable y.

```
#Split the dataset into training and testing sets

##train_test_split(): Splits the data into training and testing sets.
##test_size=0.2: Allocates 20% of the data for testing and 80% for training.
##random_state=42: Ensures reproducibility of the split.
from sklearn.model_selection import train_test_split

# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

train_test_split(): Splits the data into training and testing sets. **test_size=0.2:** Allocates 20% of the data for testing and 80% for training. **random_state=42:** Ensures reproducibility of the split.

Standardize the feature variables

```
from sklearn.preprocessing import StandardScaler

# Standardize the feature variables
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

StandardScaler(): Standardizes features by removing the mean and scaling to unit variance. **fit_transform():** Fits the scaler to the training data and transforms it. **transform():* *Transforms the testing data using the already fitted scaler.

Train a Linear Regression model using the training data

```
from sklearn.linear_model import LinearRegression
model=LinearRegression()
model.fit(X_train,y_train)
```

```
▾ LinearRegression
LinearRegression()
```

In Training,62% Score

```
model.score(X_train,y_train)*100
```

```
62.64565800251868
```

```
# Display the model's coefficients and intercep
print("Coefficients:", model.coef_)
```