Code:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

Code:

```
df = pd.read_csv('/content/sample_data/advertising.csv')
df.head()
```

Code:

```
df.info()
```

Output:

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1000 entries, 0 to 999
Data columns (total 10 columns):
 #   Column                    Non-Null Count  Dtype  
---  ------                    --------------  -----  
 0   Daily Time Spent on Site  1000 non-null   float64
 1   Age                       1000 non-null   int64  
 2   Area Income               1000 non-null   float64
 3   Daily Internet Usage      1000 non-null   float64
 4   Ad Topic Line             1000 non-null   object 
 5   City                      1000 non-null   object 
 6   Male                      1000 non-null   int64  
 7   Country                   1000 non-null   object 
 8   Timestamp                 1000 non-null   object 
 9   Clicked on Ad             1000 non-null   int64  
dtypes: float64(3), int64(3), object(4)
memory usage: 78.3+ KB
```

Code:

```
df.sample(12)
```

Code:

```
a = df.shape
print(f'Rows: {a[0]}')
print(f'Columns: {a[1]}')
```

Output:

```
Rows: 1000
Columns: 10
```

Code:

```
df.isnull().sum().sum()
```

Code:

```
df.duplicated().sum()
```

Code:

```
import plotly.express as px
```

Code:

```
colList = df.columns
colList
```

Here "CLICKED ON AD" is the target column we need for training the model

Code:

```
for x in colList:
  if df[x].dtype != 'object':
    fig = px.box(df[x],title =f'Box Plot of {x} column')
    fig.show()
```

Code:

```
outCols = ['Area Income']
```

Code:

```
for x in outCols:
  Q1 = df[x].quantile(0.25)
  Q3 = df[x].quantile(0.75)
  IQR = Q3 - Q1

  LF = Q1 - (1.5*IQR)
  UF = Q3 + (1.5*IQR)

  # Keep the values in between UF and LF
  df = df[(df[x]>=LF) & (df[x]<=UF)]
```

Code:

```
for x in colList:
  if df[x].dtype != 'object':
    fig = px.box(df[x],title =f'Box Plot of {x} column')
    fig.show()
```

Code:

```
df.shape
```

male ratio of ads clicked

Code:

```
# checking male female click to ad ratio
sns.barplot(data=df, x='Male', y='Clicked on Ad')
```

Code:

```
# print co-relation
plt.figure(figsize=(10,7))
sns.heatmap(df.corr(numeric_only=True), annot=True, cmap='coolwarm', fmt='0.2f', linewidths=0.5)
plt.title('Correlation Heatmap')
plt.tight_layout()
plt.show()
```

encoding and model building

Code:

```
catCol = []
for x in df.columns:
  if df[x].dtype =='object' and x !='Clicked on Ad':
    catCol.append(x)
```

Code:

```
catCol
```

now encoding the object data type into numerical data type

Code:

```
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
labelMap = {}
for x in catCol:
  df[x] = le.fit_transform(df[x])
  labelMap[x] = dict(zip(le.classes_, le.transform(le.classes_)))
  print(f'Mapped for {x}: {labelMap[x]}')
```

Output:

```
Mapped for Ad Topic Line: {'Adaptive 24hour Graphic Interface': np.int64(0), 'Adaptive asynchronous a
Mapped for City: {'Adamsbury': np.int64(0), 'Adamside': np.int64(1), 'Adamsstad': np.int64(2), 'Alanv
Mapped for Country: {'Afghanistan': np.int64(0), 'Albania': np.int64(1), 'Algeria': np.int64(2), 'Amo
Mapped for Timestamp: {'2016-01-01 02:52:10': np.int64(0), '2016-01-01 03:35:35': np.int64(1), '2016
```

Code:

```
df
```

Splitting the Data Into Dependent and Independent Columns

Code:

```
#we have excluded Ad Topic Line, city, country and timestamp from the model
x = df[['Daily Time Spent on Site', 'Age', 'Area Income', 'Daily Internet Usage', 'Male']]
y = df['Clicked on Ad']
x
```

Code:

```
y
```

Train Test and Splitting the model

Code:

```
from sklearn.model_selection import train_test_split
```

Code:

```
x_train , x_test, y_train, y_test = train_test_split(x,y,train_size=0.7,random_state = 42)
```

Code:

```
x_train
```

Code:

```
x_test
```

implementing the model

Code:

```
from sklearn.linear_model import LogisticRegression
```

Code:

```
model = LogisticRegression()
```

Code:

```
model.fit(x_train,y_train)
```

Output:

```
/usr/local/lib/python3.12/dist-packages/sklearn/linear_model/_logistic.py:465: ConvergenceWarning:

lbfgs failed to converge (status=1):
STOP: TOTAL NO. OF ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
```

Code:

```
y_pred = model.predict(x_test)
```

Code:

```
y_pred
```

Code:

```
y_test
```

Evaluation of model

Code:

```
from sklearn.metrics import confusion_matrix, classification_report, accuracy_score
```

Code:

```
from sklearn.metrics import precision_score, recall_score, f1_score
```

Code:

```
accuracy_score(y_test,y_pred)
#the accuracy is 92% of model
```

Code:

```
confusion_matrix(y_test,y_pred)
```

Code:

```
precision_score(y_test,y_pred)
```

Code:

```
recall_score(y_test,y_pred)
```

Code:

```
f1_score(y_test,y_pred)
```

Code:

```
print(classification_report(y_test,y_pred))
```

Output:

```
                   precision    recall  f1-score   support

               0       0.89      0.95      0.92       154
               1       0.95      0.88      0.91       144

        accuracy                           0.92       298
       macro avg       0.92      0.91      0.92       298
    weighted avg       0.92      0.92      0.92       298
```

sigmoid

Code:

```
def sigmoid(x):
  result = 1/(1+np.exp(-x))
  return result
```

Code:

```
sigmoid(-3)
```

Code:

```
y_score = model.predict_proba(x_test)[:,1]
```

Code:

```
y_score
```

Code:

```
len(y_score)
```

Code:

```
sorInd = np.argmax(y_score)
sorInd
```

Code:

```
y_score[sorInd]
```

Code:

```
sortLabel = y_test.iloc[sorInd]
sortLabel
```

Code:

```
sortScore = y_score[sorInd]
sortScore
```

Code:

```
x_values = np.linspace(-10,10,100)
y_sigmoid = sigmoid(x_values)
```

Code:

```
y_sigmoid
```

Code:

```
y_score
```

Code:

```
ind = np.where(y_test)
```

Code:

```
ind
```

Code:

```
sortLabel = y_test.iloc[:]
sortLabel
```

Code:

```
plt.plot(x_values,y_sigmoid,color='red')
plt.axhline(y=0.5, color ='black',linestyle='--')
plt.xticks([0, 1], ['No', 'Yes'])
plt.scatter(sortLabel,y_score)
plt.show()
```