In [ ]:
```
# Initialize Otter
import otter
grader = otter.Notebook("3-8216.ipynb")
```

COMPSS211 Problem Set 3 (15 points total)

Due 2024-11-04, 11.59pm California time, via bCourses

Part I

For Part I, we will be using the Heart Attack Prediction Dataset.

Download the CSV file and save it in the same directory as your Jupyter Notebook, and using this file answer the following questions.

For each of the questions in Part I, write code that answers the question and an answer in English when appropriate.

Q1 - Clean the Data (1 points)

- Load the data into a Pandas DataFrame.
- Split the Blood Pressure column into two separate columns: Systolic BP and Diastolic BP. Ensure that both new columns are of type float and remove the original Blood Pressure column from the DataFrame.
- Convert categorical variables into numerical format using OneHotEncoder. The categorical columns to encode are: Sex, Diet, Country, Continent, and Hemisphere

In [2]:
```
import kagglehub


path = kagglehub.dataset_download("iamsouravbanerjee/heart-attack-predictior

print("Path to dataset files:", path)
```

```
Downloading from https://www.kaggle.com/api/v1/datasets/download/iamsouravba
nerjee/heart-attack-prediction-dataset?dataset_version_number=2...
100%|██████████| 519k/519k [00:00<00:00, 56.5MB/s]
Extracting files...
Path to dataset files: /root/.cache/kagglehub/datasets/iamsouravbanerjee/hea
rt-attack-prediction-dataset/versions/2
```

In [69]:
```
import pandas as pd
df = pd.read_csv('heart.csv')
df.head()
```

Out[69]:

	Patient ID	Age	Sex	Cholesterol	Blood Pressure	Heart Rate	Diabetes	Family History	Smoking
0	BMW7812	67	Male	208	158/88	72	0	0	1
1	CZE1114	21	Male	389	165/93	98	1	1	1
2	BNI9906	21	Female	324	174/99	72	1	0	0
3	JLN3497	84	Male	383	163/100	73	1	1	1
4	GFO8847	66	Male	318	91/88	93	1	1	1

5 rows × 26 columns

In [86]:
```
df.columns
```

Out[86]:
```
Index(['Patient ID', 'Age', 'Sex', 'Cholesterol', 'Heart Rate', 'Diabetes',
       'Family History', 'Smoking', 'Obesity', 'Alcohol Consumption',
       'Exercise Hours Per Week', 'Diet', 'Previous Heart Problems',
       'Medication Use', 'Stress Level', 'Sedentary Hours Per Day', 'Incom
e',
       'BMI', 'Triglycerides', 'Physical Activity Days Per Week',
       'Sleep Hours Per Day', 'Country', 'Continent', 'Hemisphere',
       'Heart Attack Risk', 'Systolic BP', 'Diastolic BP'],
      dtype='object')
```

In [72]:
```
categorical_col = df.select_dtypes(include=['object', 'category']).columns

print(categorical_col)
```

```
Index(['Patient ID', 'Sex', 'Blood Pressure', 'Diet', 'Country', 'Continen
t',
       'Hemisphere'],
      dtype='object')
```

In [85]:
```
bp_split = df['Blood Pressure'].str.split('/', expand=True).astype(float)
df['Systolic BP'] = bp_split[0]
df['Diastolic BP'] = bp_split[1]
df.drop('Blood Pressure', axis=1, inplace=True)
df.head()
```

Out[85]:

	Patient ID	Age	Sex	Cholesterol	Heart Rate	Diabetes	Family History	Smoking	Obesity
0	BMW7812	67	Male	208	72	0	0	1	0
1	CZE1114	21	Male	389	98	1	1	1	1
2	BNI9906	21	Female	324	72	1	0	0	0
3	JLN3497	84	Male	383	73	1	1	1	0
4	GFO8847	66	Male	318	93	1	1	1	1

5 rows × 27 columns

In [56]:
```
df = pd.get_dummies(df, columns=['Sex', 'Diet', 'Country', 'Continent', 'Hem
df.head()
```

Out[56]:

	Patient ID	Age	Cholesterol	Heart Rate	Diabetes	Family History	Smoking	Obesity	Alc Consum
0	BMW7812	67	208	72	0	0	1	0	
1	CZE1114	21	389	98	1	1	1	1	
2	BNI9906	21	324	72	1	0	0	0	
3	JLN3497	84	383	73	1	1	1	0	
4	GFO8847	66	318	93	1	1	1	1	

5 rows × 55 columns

Q2: Train a Model (2 points)

- Create a feature `DataFrame` called `X` by removing the Patient ID and Heart Attack Risk columns.
- Create a target `Series` called `y`.
- Split the data into 80% training and 20% test using `train_test_split`.
- Train a `DecisionTreeClassifier` with default arguments (except for `random_state=42`) and fit it to the training data.
- Display the 10 most important features and their names according to the built-in model importance measure.

- Print the model accuracy on the test set

In [12]:
```
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score

X = df.drop(['Patient ID', 'Heart Attack Risk'], axis=1)
y = df['Heart Attack Risk']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, rar

clf = DecisionTreeClassifier(random_state=1)
clf.fit(X_train, y_train)

importances = clf.feature_importances_
feature_names = X.columns

feature_importances = pd.DataFrame({
    'Feature': feature_names,
    'Importance': importances
})

top_features = feature_importances.sort_values(by='Importance', ascending=Fa
print("Top 10 Feature Importances:")
print(top_features)

y_pred = clf.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
print(f"Model accuracy on the test set: {accuracy:.2f}")
```

```
Top 10 Feature Importances:
                    Feature  Importance
12  Sedentary Hours Per Day    0.084354
15            Triglycerides    0.082377
14                      BMI    0.081498
8   Exercise Hours Per Week    0.080786
18              Systolic BP    0.068880
2                Heart Rate    0.065355
13                   Income    0.062412
1               Cholesterol    0.057256
0                       Age    0.057108
19             Diastolic BP    0.052993
Model accuracy on the test set: 0.53
```

Q3: Determine the Best Hyperparameters (1 point)

Considering the following values for each of these hypeparameters, determine the best set of values:

- 'max_depth': 3, 5, 7, 10, 12
- 'min_samples_split': 10, 30, 50, 70
- 'min_samples_leaf': 5, 10, 20, 23
- 'criterion': 'gini', 'entropy'

```
In [13]: from sklearn.model_selection import GridSearchCV

params = {
    'max_depth': [3, 5, 7, 10, 12],
    'min_samples_split': [10, 30, 50, 70],
    'min_samples_leaf': [5, 10, 20, 23],
    'criterion': ['gini', 'entropy']
}

grid_search = GridSearchCV(estimator=clf, param_grid=params, cv=5, scoring='

clf = DecisionTreeClassifier(random_state=1)


grid_search = GridSearchCV(estimator=clf, param_grid=params, cv=5, scoring='

grid_search.fit(X_train, y_train)

print(grid_search.best_params_)
print(grid_search.best_score_)
```

```
{'criterion': 'entropy', 'max_depth': 3, 'min_samples_leaf': 5, 'min_samples
_split': 10}
0.6380884450784594
```

```
/usr/local/lib/python3.10/dist-packages/numpy/ma/core.py:2820: RuntimeWarnin
g: invalid value encountered in cast
  _data = np.array(data, dtype=dtype, copy=copy,
```

Q4: Interpretation (1 point)

Which lifestyle factors appear to be the strongest predictors of heart attack risk? How might this information be used to develop preventive healthcare strategies? What are the limitations of using this model for medical decision-making?

According to the model, the strongest predictors of heart attack risk are: sedentary hours per day, triglycerides, BMI, excercise hours per week, and Systolic BP.

This information can be used to develop preventive healthcare strategies by identifying high-risk individuals and targeting them for interventions such as lifestyle changes, medication, or early detection and treatment. Also the models accuracy is of .53 so it is only slightly better than guessing, which limits the usefulness of the model.

Q5: Tabular Neural Network (2.5 points)

Using the `fastai` library, fit a tabular neural network model to solve the same problem:

- Split the dataset into training and validation sets and create DataLoaders
- Define and train the model using `fastai`
- Evaluate the model's performance using the validation set
- Describe which features are important for the model (150 words max)

In [14]:
```
pip install fastai
```

```
Requirement already satisfied: fastai in /usr/local/lib/python3.10/dist-pack
ages (2.7.18)
Requirement already satisfied: pip in /usr/local/lib/python3.10/dist-package
s (from fastai) (24.1.2)
Requirement already satisfied: packaging in /usr/local/lib/python3.10/dist-p
ackages (from fastai) (24.1)
Requirement already satisfied: fastdownload<2,>=0.0.5 in /usr/local/lib/pyth
on3.10/dist-packages (from fastai) (0.0.7)
Requirement already satisfied: fastcore<1.8,>=1.5.29 in /usr/local/lib/pytho
n3.10/dist-packages (from fastai) (1.7.19)
Requirement already satisfied: torchvision>=0.11 in /usr/local/lib/python3.1
0/dist-packages (from fastai) (0.20.0+cu121)
Requirement already satisfied: matplotlib in /usr/local/lib/python3.10/dist-
packages (from fastai) (3.8.0)
Requirement already satisfied: pandas in /usr/local/lib/python3.10/dist-pack
ages (from fastai) (2.2.2)
Requirement already satisfied: requests in /usr/local/lib/python3.10/dist-pa
ckages (from fastai) (2.32.3)
Requirement already satisfied: pyyaml in /usr/local/lib/python3.10/dist-pack
ages (from fastai) (6.0.2)
Requirement already satisfied: fastprogress>=0.2.4 in /usr/local/lib/python
3.10/dist-packages (from fastai) (1.0.3)
Requirement already satisfied: pillow>=9.0.0 in /usr/local/lib/python3.10/di
st-packages (from fastai) (10.4.0)
Requirement already satisfied: scikit-learn in /usr/local/lib/python3.10/dis
t-packages (from fastai) (1.5.2)
Requirement already satisfied: scipy in /usr/local/lib/python3.10/dist-packa
ges (from fastai) (1.13.1)
Requirement already satisfied: spacy<4 in /usr/local/lib/python3.10/dist-pac
kages (from fastai) (3.7.5)
Requirement already satisfied: torch<2.6,>=1.10 in /usr/local/lib/python3.1
0/dist-packages (from fastai) (2.5.0+cu121)
Requirement already satisfied: spacy-legacy<3.1.0,>=3.0.11 in /usr/local/li
b/python3.10/dist-packages (from spacy<4->fastai) (3.0.12)
Requirement already satisfied: spacy-loggers<2.0.0,>=1.0.0 in /usr/local/li
b/python3.10/dist-packages (from spacy<4->fastai) (1.0.5)
Requirement already satisfied: murmurhash<1.1.0,>=0.28.0 in /usr/local/lib/p
ython3.10/dist-packages (from spacy<4->fastai) (1.0.10)
Requirement already satisfied: cymem<2.1.0,>=2.0.2 in /usr/local/lib/python
3.10/dist-packages (from spacy<4->fastai) (2.0.8)
Requirement already satisfied: preshed<3.1.0,>=3.0.2 in /usr/local/lib/pytho
n3.10/dist-packages (from spacy<4->fastai) (3.0.9)
Requirement already satisfied: thinc<8.3.0,>=8.2.2 in /usr/local/lib/python
3.10/dist-packages (from spacy<4->fastai) (8.2.5)
Requirement already satisfied: wasabi<1.2.0,>=0.9.1 in /usr/local/lib/python
3.10/dist-packages (from spacy<4->fastai) (1.1.3)
Requirement already satisfied: srsly<3.0.0,>=2.4.3 in /usr/local/lib/python
3.10/dist-packages (from spacy<4->fastai) (2.4.8)
Requirement already satisfied: catalogue<2.1.0,>=2.0.6 in /usr/local/lib/pyt
hon3.10/dist-packages (from spacy<4->fastai) (2.0.10)
Requirement already satisfied: weasel<0.5.0,>=0.1.0 in /usr/local/lib/python
3.10/dist-packages (from spacy<4->fastai) (0.4.1)
Requirement already satisfied: typer<1.0.0,>=0.3.0 in /usr/local/lib/python
3.10/dist-packages (from spacy<4->fastai) (0.12.5)
Requirement already satisfied: tqdm<5.0.0,>=4.38.0 in /usr/local/lib/python
3.10/dist-packages (from spacy<4->fastai) (4.66.6)
```

```
Requirement already satisfied: pydantic!=1.8,!=1.8.1,<3.0.0,>=1.7.4 in /usr/
local/lib/python3.10/dist-packages (from spacy<4->fastai) (2.9.2)
Requirement already satisfied: jinja2 in /usr/local/lib/python3.10/dist-pack
ages (from spacy<4->fastai) (3.1.4)
Requirement already satisfied: setuptools in /usr/local/lib/python3.10/dist-
packages (from spacy<4->fastai) (75.1.0)
Requirement already satisfied: langcodes<4.0.0,>=3.2.0 in /usr/local/lib/pyt
hon3.10/dist-packages (from spacy<4->fastai) (3.4.1)
Requirement already satisfied: numpy>=1.19.0 in /usr/local/lib/python3.10/di
st-packages (from spacy<4->fastai) (1.26.4)
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/py
thon3.10/dist-packages (from requests->fastai) (3.4.0)
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.10/dis
t-packages (from requests->fastai) (3.10)
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.
10/dist-packages (from requests->fastai) (2.2.3)
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.
10/dist-packages (from requests->fastai) (2024.8.30)
Requirement already satisfied: filelock in /usr/local/lib/python3.10/dist-pa
ckages (from torch<2.6,>=1.10->fastai) (3.16.1)
Requirement already satisfied: typing-extensions>=4.8.0 in /usr/local/lib/py
thon3.10/dist-packages (from torch<2.6,>=1.10->fastai) (4.12.2)
Requirement already satisfied: networkx in /usr/local/lib/python3.10/dist-pa
ckages (from torch<2.6,>=1.10->fastai) (3.4.2)
Requirement already satisfied: fsspec in /usr/local/lib/python3.10/dist-pack
ages (from torch<2.6,>=1.10->fastai) (2024.10.0)
Requirement already satisfied: sympy==1.13.1 in /usr/local/lib/python3.10/di
st-packages (from torch<2.6,>=1.10->fastai) (1.13.1)
Requirement already satisfied: mpmath<1.4,>=1.1.0 in /usr/local/lib/python3.
10/dist-packages (from sympy==1.13.1->torch<2.6,>=1.10->fastai) (1.3.0)
Requirement already satisfied: contourpy>=1.0.1 in /usr/local/lib/python3.1
0/dist-packages (from matplotlib->fastai) (1.3.0)
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.10/dis
t-packages (from matplotlib->fastai) (0.12.1)
Requirement already satisfied: fonttools>=4.22.0 in /usr/local/lib/python3.1
0/dist-packages (from matplotlib->fastai) (4.54.1)
Requirement already satisfied: kiwisolver>=1.0.1 in /usr/local/lib/python3.1
0/dist-packages (from matplotlib->fastai) (1.4.7)
Requirement already satisfied: pyparsing>=2.3.1 in /usr/local/lib/python3.1
0/dist-packages (from matplotlib->fastai) (3.2.0)
Requirement already satisfied: python-dateutil>=2.7 in /usr/local/lib/python
3.10/dist-packages (from matplotlib->fastai) (2.8.2)
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.10/dis
t-packages (from pandas->fastai) (2024.2)
Requirement already satisfied: tzdata>=2022.7 in /usr/local/lib/python3.10/d
ist-packages (from pandas->fastai) (2024.2)
Requirement already satisfied: joblib>=1.2.0 in /usr/local/lib/python3.10/di
st-packages (from scikit-learn->fastai) (1.4.2)
Requirement already satisfied: threadpoolctl>=3.1.0 in /usr/local/lib/python
3.10/dist-packages (from scikit-learn->fastai) (3.5.0)
Requirement already satisfied: language-data>=1.2 in /usr/local/lib/python3.
10/dist-packages (from langcodes<4.0.0,>=3.2.0->spacy<4->fastai) (1.2.0)
Requirement already satisfied: annotated-types>=0.6.0 in /usr/local/lib/pyth
on3.10/dist-packages (from pydantic!=1.8,!=1.8.1,<3.0.0,>=1.7.4->spacy<4->fa
stai) (0.7.0)
Requirement already satisfied: pydantic-core==2.23.4 in /usr/local/lib/pytho
```

```
n3.10/dist-packages (from pydantic!=1.8,!=1.8.1,<3.0.0,>=1.7.4->spacy<4->fas
tai) (2.23.4)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.10/dist-pa
ckages (from python-dateutil>=2.7->matplotlib->fastai) (1.16.0)
Requirement already satisfied: blis<0.8.0,>=0.7.8 in /usr/local/lib/python3.
10/dist-packages (from thinc<8.3.0,>=8.2.2->spacy<4->fastai) (0.7.11)
Requirement already satisfied: confection<1.0.0,>=0.0.1 in /usr/local/lib/py
thon3.10/dist-packages (from thinc<8.3.0,>=8.2.2->spacy<4->fastai) (0.1.5)
Requirement already satisfied: click>=8.0.0 in /usr/local/lib/python3.10/dis
t-packages (from typer<1.0.0,>=0.3.0->spacy<4->fastai) (8.1.7)
Requirement already satisfied: shellingham>=1.3.0 in /usr/local/lib/python3.
10/dist-packages (from typer<1.0.0,>=0.3.0->spacy<4->fastai) (1.5.4)
Requirement already satisfied: rich>=10.11.0 in /usr/local/lib/python3.10/di
st-packages (from typer<1.0.0,>=0.3.0->spacy<4->fastai) (13.9.3)
Requirement already satisfied: cloudpathlib<1.0.0,>=0.7.0 in /usr/local/lib/
python3.10/dist-packages (from weasel<0.5.0,>=0.1.0->spacy<4->fastai) (0.20.
0)
Requirement already satisfied: smart-open<8.0.0,>=5.2.1 in /usr/local/lib/py
thon3.10/dist-packages (from weasel<0.5.0,>=0.1.0->spacy<4->fastai) (7.0.5)
Requirement already satisfied: MarkupSafe>=2.0 in /usr/local/lib/python3.10/
dist-packages (from jinja2->spacy<4->fastai) (3.0.2)
Requirement already satisfied: marisa-trie>=0.7.7 in /usr/local/lib/python3.
10/dist-packages (from language-data>=1.2->langcodes<4.0.0,>=3.2.0->spacy<4-
>fastai) (1.2.1)
Requirement already satisfied: markdown-it-py>=2.2.0 in /usr/local/lib/pytho
n3.10/dist-packages (from rich>=10.11.0->typer<1.0.0,>=0.3.0->spacy<4->fasta
i) (3.0.0)
Requirement already satisfied: pygments<3.0.0,>=2.13.0 in /usr/local/lib/pyt
hon3.10/dist-packages (from rich>=10.11.0->typer<1.0.0,>=0.3.0->spacy<4->fas
tai) (2.18.0)
Requirement already satisfied: wrapt in /usr/local/lib/python3.10/dist-packa
ges (from smart-open<8.0.0,>=5.2.1->weasel<0.5.0,>=0.1.0->spacy<4->fastai)
(1.16.0)
Requirement already satisfied: mdurl~=0.1 in /usr/local/lib/python3.10/dist-
packages (from markdown-it-py>=2.2.0->rich>=10.11.0->typer<1.0.0,>=0.3.0->sp
acy<4->fastai) (0.1.2)
```

In [87]:

```
print(df.columns)
for column in df.columns:
  print(f"\nUnique value counts for {column}:")
  print(df[column].value_counts())
```

```
Index(['Patient ID', 'Age', 'Sex', 'Cholesterol', 'Heart Rate', 'Diabetes',
       'Family History', 'Smoking', 'Obesity', 'Alcohol Consumption',
       'Exercise Hours Per Week', 'Diet', 'Previous Heart Problems',
       'Medication Use', 'Stress Level', 'Sedentary Hours Per Day', 'Incom
e',
       'BMI', 'Triglycerides', 'Physical Activity Days Per Week',
       'Sleep Hours Per Day', 'Country', 'Continent', 'Hemisphere',
       'Heart Attack Risk', 'Systolic BP', 'Diastolic BP'],
      dtype='object')

Unique value counts for Patient ID:
Patient ID
BMW7812    1
DCD4966    1
ETF7967    1
WPM0379    1
MLL3192    1
          ..
NRV3150    1
EZF9124    1
EOI3054    1
MFA4348    1
ZWN9666    1
Name: count, Length: 8763, dtype: int64

Unique value counts for Age:
Age
90    152
42    150
33    147
59    147
29    137
     ...
75    102
72    101
39    100
47     99
51     82
Name: count, Length: 73, dtype: int64

Unique value counts for Sex:
Sex
Male      6111
Female    2652
Name: count, dtype: int64

Unique value counts for Cholesterol:
Cholesterol
235    52
360    47
149    46
218    46
251    45
       ..
248    20
186    20
```

```
328    20
398    20
397    19
Name: count, Length: 281, dtype: int64

Unique value counts for Heart Rate:
Heart Rate
94     157
97     146
57     143
52     140
104    139
      ... 
70     107
48     107
79     105
96      97
73      93
Name: count, Length: 71, dtype: int64

Unique value counts for Diabetes:
Diabetes
1    5716
0    3047
Name: count, dtype: int64

Unique value counts for Family History:
Family History
0    4443
1    4320
Name: count, dtype: int64

Unique value counts for Smoking:
Smoking
1    7859
0     904
Name: count, dtype: int64

Unique value counts for Obesity:
Obesity
1    4394
0    4369
Name: count, dtype: int64

Unique value counts for Alcohol Consumption:
Alcohol Consumption
1    5241
0    3522
Name: count, dtype: int64

Unique value counts for Exercise Hours Per Week:
Exercise Hours Per Week
4.168189     1
18.477430    1
11.883523    1
19.353157    1
```

```
19.365546    1
            ..
9.884039     1
12.644947    1
1.089868     1
10.500477    1
18.081748    1
Name: count, Length: 8763, dtype: int64

Unique value counts for Diet:
Diet
Healthy      2960
Average      2912
Unhealthy    2891
Name: count, dtype: int64

Unique value counts for Previous Heart Problems:
Previous Heart Problems
0    4418
1    4345
Name: count, dtype: int64

Unique value counts for Medication Use:
Medication Use
0    4396
1    4367
Name: count, dtype: int64

Unique value counts for Stress Level:
Stress Level
2     913
4     910
7     903
9     887
8     879
3     868
1     865
5     860
6     855
10    823
Name: count, dtype: int64

Unique value counts for Sedentary Hours Per Day:
Sedentary Hours Per Day
6.615001     1
0.772688     1
0.723868     1
10.125510    1
2.054331     1
            ..
11.921800    1
0.087028     1
9.198925     1
3.383760     1
9.005234     1
Name: count, Length: 8763, dtype: int64
```

```
Unique value counts for Income:
Income
225278    4
194461    3
195282    3
220507    2
139451    2
         ..
44744     1
85563     1
20443     1
258704    1
247338    1
Name: count, Length: 8615, dtype: int64

Unique value counts for BMI:
BMI
31.251233    1
39.385227    1
36.280438    1
18.218558    1
23.885840    1
            ..
28.358868    1
22.539845    1
34.721372    1
18.881817    1
32.914151    1
Name: count, Length: 8763, dtype: int64

Unique value counts for Triglycerides:
Triglycerides
799    25
507    22
121    22
593    22
469    22
       ..
120     3
213     3
185     3
295     3
130     2
Name: count, Length: 771, dtype: int64

Unique value counts for Physical Activity Days Per Week:
Physical Activity Days Per Week
3    1143
1    1121
2    1109
7    1095
5    1079
4    1077
6    1074
0    1065
```

```
Name: count, dtype: int64

Unique value counts for Sleep Hours Per Day:
Sleep Hours Per Day
10    1293
8     1288
6     1276
7     1270
5     1263
9     1192
4     1181
Name: count, dtype: int64

Unique value counts for Country:
Country
Germany           477
Argentina         471
Brazil            462
United Kingdom    457
Australia         449
Nigeria           448
France            446
Canada            440
China             436
New Zealand       435
Japan             433
Italy             431
Spain             430
Colombia          429
Thailand          428
South Africa      425
Vietnam           425
United States     420
India             412
South Korea       409
Name: count, dtype: int64

Unique value counts for Continent:
Continent
Asia             2543
Europe           2241
South America    1362
Australia         884
Africa            873
North America     860
Name: count, dtype: int64

Unique value counts for Hemisphere:
Hemisphere
Northern Hemisphere    5660
Southern Hemisphere    3103
Name: count, dtype: int64

Unique value counts for Heart Attack Risk:
Heart Attack Risk
0    5624
```

```
1    3139
Name: count, dtype: int64

Unique value counts for Systolic BP:
Systolic BP
102.0    123
142.0    117
101.0    115
132.0    112
140.0    112
        ...
127.0     77
112.0     75
179.0     75
122.0     73
137.0     67
Name: count, Length: 91, dtype: int64

Unique value counts for Diastolic BP:
Diastolic BP
83.0     198
103.0    193
96.0     191
78.0     190
89.0     190
72.0     189
98.0     189
105.0    189
93.0     188
63.0     185
76.0     184
102.0    183
104.0    183
94.0     179
95.0     178
82.0     178
108.0    177
107.0    177
69.0     177
90.0     176
64.0     176
106.0    175
97.0     174
60.0     173
99.0     173
91.0     172
73.0     171
66.0     171
81.0     170
77.0     170
88.0     169
100.0    168
75.0     168
65.0     168
62.0     167
87.0     166
```

```
74.0     165
67.0     164
109.0    161
80.0     161
68.0     160
79.0     160
86.0     159
71.0     157
70.0     157
92.0     154
61.0     152
84.0     151
85.0     149
110.0    145
101.0    143
Name: count, dtype: int64
```

In [88]:

```
categorical_columns = [
    'Patient ID',
    'Sex',
    'Diabetes',
    'Family History',
    'Smoking',
    'Obesity',
    'Alcohol Consumption',
    'Diet',
    'Previous Heart Problems',
    'Medication Use',
    'Continent',
    'Hemisphere',
    'Heart Attack Risk',
    'Country'
]

non_categorical_columns = [
    'Age',
    'Cholesterol',
    'Heart Rate',
    'Exercise Hours Per Week',
    'Stress Level',
    'Sedentary Hours Per Day',
    'Income',
    'BMI',
    'Triglycerides',
    'Physical Activity Days Per Week',
    'Sleep Hours Per Day',
    'Systolic BP',
    'Diastolic BP'
]
```

In [101…

```
from fastai.tabular.all import *
import pandas as pd
import torch


categorical_columns = [
```

```
    'Sex',
    'Diabetes',
    'Family History',
    'Smoking',
    'Obesity',
    'Alcohol Consumption',
    'Diet',
    'Previous Heart Problems',
    'Medication Use',
    'Continent',
    'Hemisphere',
    'Country'
]

non_categorical_columns = [
    'Age',
    'Cholesterol',
    'Heart Rate',
    'Exercise Hours Per Week',
    'Stress Level',
    'Sedentary Hours Per Day',
    'Income',
    'BMI',
    'Triglycerides',
    'Physical Activity Days Per Week',
    'Sleep Hours Per Day',
    'Systolic BP',
    'Diastolic BP'
]


df['Heart Attack Risk'] = df['Heart Attack Risk'].astype('category')


splits = RandomSplitter(valid_pct=0.2, seed=42)(range_of(df))


procs = [Categorify, FillMissing, Normalize]

to = TabularPandas(
    df,
    procs=procs,
    cat_names=categorical_columns,
    cont_names=non_categorical_columns,
    y_names='Heart Attack Risk',
    splits=splits,
    y_block=CategoryBlock()
)


dls = to.dataloaders(bs=64)

learn = tabular_learner(
    dls,
    layers=[200, 100],
    metrics=accuracy,
```

```
    loss_func=CrossEntropyLossFlat(),
    wd=1e-2
)

learn.fit_one_cycle(20, lr_max=1e-3)

learn.show_results()
```

epoch	train_loss	valid_loss	accuracy	time
0	0.706870	0.699202	0.544521	00:03
1	0.673657	0.677681	0.587329	00:03
2	0.647293	0.675363	0.623288	00:03
3	0.646245	0.666270	0.611872	00:01
4	0.637875	0.677108	0.614155	00:01
5	0.618973	0.679499	0.617580	00:01
6	0.612994	0.685443	0.598173	00:01
7	0.600794	0.691948	0.610160	00:02
8	0.578011	0.700306	0.616438	00:02
9	0.560857	0.739223	0.594178	00:01
10	0.552998	0.709125	0.597032	00:01
11	0.523796	0.733370	0.599886	00:01
12	0.502140	0.742686	0.608447	00:01
13	0.485652	0.749867	0.593607	00:01
14	0.475445	0.760929	0.588470	00:01
15	0.454695	0.759443	0.603311	00:02
16	0.449254	0.769534	0.594749	00:02
17	0.444112	0.776768	0.600457	00:01
18	0.435123	0.778073	0.598173	00:01
19	0.424241	0.775784	0.594749	00:01

	Sex	Diabetes	Family History	Smoking	Obesity	Alcohol Consumption	Diet	Previous Heart Problems	Medicatior Use
0	2.0	2.0	1.0	2.0	1.0	2.0	3.0	2.0	2.(
1	2.0	2.0	1.0	2.0	1.0	2.0	1.0	2.0	2.(
2	2.0	1.0	1.0	2.0	2.0	1.0	3.0	1.0	1.(
3	2.0	2.0	1.0	2.0	2.0	2.0	2.0	2.0	2.(
4	2.0	2.0	1.0	2.0	1.0	2.0	1.0	2.0	1.(
5	1.0	1.0	1.0	2.0	2.0	2.0	1.0	1.0	2.(
6	1.0	2.0	1.0	2.0	2.0	2.0	2.0	1.0	1.(
7	2.0	2.0	1.0	2.0	2.0	1.0	3.0	2.0	1.(
8	1.0	2.0	2.0	2.0	1.0	2.0	2.0	1.0	1.(

In [104…

```
import numpy as np

def permute_feature_importance(learner, df, features, metric=accuracy, n_rep

    baseline = learner.validate()[1]


    feature_importances = {}


    for feature in features:
        metric_decrease = 0.0

        for _ in range(n_repeats):
            df_permuted = df.copy()
            df_permuted[feature] = np.random.permutation(df_permuted[feature

            to_permuted = TabularPandas(
                df_permuted,
                procs=procs,
                cat_names=categorical_columns,
                cont_names=non_categorical_columns,
                y_names='Heart Attack Risk',
                splits=splits,
                y_block=CategoryBlock()
            )
            dls_permuted = to_permuted.dataloaders(bs=64)

            learner.dls = dls_permuted
            permuted_accuracy = learner.validate()[1]

            metric_decrease += baseline - permuted_accuracy

        feature_importances[feature] = metric_decrease / n_repeats
```

```
    learner.dls = dls

    return feature_importances

features = categorical_columns + non_categorical_columns
feature_importance_scores = permute_feature_importance(learn, df, features)

sorted_importance = sorted(feature_importance_scores.items(), key=lambda x:
for feature, importance in sorted_importance:
    print(f"{feature}: {importance:.4f}")
```

```
BMI: 0.0185
Country: 0.0154
Age: 0.0151
Diastolic BP: 0.0143
Income: 0.0126
Physical Activity Days Per Week: 0.0120
Cholesterol: 0.0092
Systolic BP: 0.0076
Stress Level: 0.0075
Triglycerides: 0.0070
Sleep Hours Per Day: 0.0064
Heart Rate: 0.0062
Sedentary Hours Per Day: 0.0045
Smoking: 0.0037
Diet: 0.0030
Obesity: 0.0029
Alcohol Consumption: 0.0025
Medication Use: 0.0016
Hemisphere: 0.0016
Diabetes: 0.0014
Exercise Hours Per Week: 0.0008
Previous Heart Problems: 0.0007
Continent: -0.0008
Sex: -0.0010
Family History: -0.0016
```

To ensure the model generalized well and avoided overfitting, I simplified the architecture by reducing the number of layers and neurons and incorporated weight decay as a regularization use. After training the model, I conducted a permutation feature importance analysis to identify which features most significantly influence the predictions.

The analysis revealed that BMI, Country, and Age are the most important factors in predicting heart attack risk, with BMI having the highest impact.

Part II

In this part of the assignment, you'll have an opportunity to pursue a more open-ended analysis of a dataset. The questions guide you through the basic steps of the problem, but you're expected to make judicious decisions about each of them. Throughout this question, apply the best practices we've covered in class. If it's necessary to perform a good analysis, take steps and answer questions which are beyond those explicitly required.

For each question, write code to answer it, and provide a justification and explanation for your choices in English (max 300 words per question).

The overall task is to predict whether a passenger will accept a coupon in the In-Vehicle Coupon Recommendation dataset.

Q1: Loading and Preparing Data (2 point)

- Load the data, perform exporatory data analysis. Does the dataset have any characteristics which need to be accounted for in using it to train a model?
- Prepare this data for use in model training. Explain what you're doing.

In [ ]:
```
df1= pd.read_csv('vehicle.csv')
df1.head()
```

Out[ ]:

	destination	passanger	weather	temperature	time	coupon	expiration	g
0	No Urgent Place	Alone	Sunny	55	2PM	Restaurant(<20)	1d	F
1	No Urgent Place	Friend(s)	Sunny	80	10AM	Coffee House	2h	F
2	No Urgent Place	Friend(s)	Sunny	80	10AM	Carry out & Take away	2h	F
3	No Urgent Place	Friend(s)	Sunny	80	2PM	Coffee House	2h	F
4	No Urgent Place	Friend(s)	Sunny	80	2PM	Coffee House	1d	F

5 rows × 26 columns

In [ ]:
```
missing_values = df1.isnull().sum()
print(missing_values)
```

```
destination                 0
passanger                   0
weather                     0
temperature                 0
time                        0
coupon                      0
expiration                  0
gender                      0
age                         0
maritalStatus               0
has_children                0
education                   0
occupation                  0
income                      0
car                     12576
Bar                       107
CoffeeHouse               217
CarryAway                 151
RestaurantLessThan20      130
Restaurant20To50          189
toCoupon_GEQ5min            0
toCoupon_GEQ15min           0
toCoupon_GEQ25min           0
direction_same              0
direction_opp               0
Y                           0
dtype: int64
```

In [ ]:

```
for column in ['Bar', 'CoffeeHouse', 'CarryAway', 'RestaurantLessThan20', 'R
    df1[column].fillna(df1[column].mode()[0], inplace=True)

df1['car'].fillna('Unknown', inplace=True)
df1.head()
```

```
<ipython-input-14-bbee44ca115b>:2: FutureWarning: A value is trying to be se
t on a copy of a DataFrame or Series through chained assignment using an inp
lace method.
The behavior will change in pandas 3.0. This inplace method will never work
because the intermediate object on which we are setting values always behave
s as a copy.

For example, when doing 'df[col].method(value, inplace=True)', try using 'd
f.method({col: value}, inplace=True)' or df[col] = df[col].method(value) ins
tead, to perform the operation inplace on the original object.


  df1[column].fillna(df1[column].mode()[0], inplace=True)
<ipython-input-14-bbee44ca115b>:4: FutureWarning: A value is trying to be se
t on a copy of a DataFrame or Series through chained assignment using an inp
lace method.
The behavior will change in pandas 3.0. This inplace method will never work
because the intermediate object on which we are setting values always behave
s as a copy.

For example, when doing 'df[col].method(value, inplace=True)', try using 'd
f.method({col: value}, inplace=True)' or df[col] = df[col].method(value) ins
tead, to perform the operation inplace on the original object.


  df1['car'].fillna('Unknown', inplace=True)
```

Out[ ]:

	destination	passanger	weather	temperature	time	coupon	expiration	g
0	No Urgent Place	Alone	Sunny	55	2PM	Restaurant(<20)	1d	F
1	No Urgent Place	Friend(s)	Sunny	80	10AM	Coffee House	2h	F
2	No Urgent Place	Friend(s)	Sunny	80	10AM	Carry out & Take away	2h	F
3	No Urgent Place	Friend(s)	Sunny	80	2PM	Coffee House	2h	F
4	No Urgent Place	Friend(s)	Sunny	80	2PM	Coffee House	1d	F

5 rows × 26 columns

In [ ]:

```
categorical_cols = df1.select_dtypes(include=['object', 'category']).columns

print(categorical_cols)
```

```
Index(['destination', 'passanger', 'weather', 'time', 'coupon', 'expiratio
n',
       'gender', 'age', 'maritalStatus', 'education', 'occupation', 'incom
e',
       'car', 'Bar', 'CoffeeHouse', 'CarryAway', 'RestaurantLessThan20',
       'Restaurant20To50'],
      dtype='object')
```

In [ ]:
```
df1 = pd.get_dummies(df1, columns=categorical_cols, drop_first=True)
df1.head()
```

Out[ ]:

	temperature	has_children	toCoupon_GEQ5min	toCoupon_GEQ15min	toCoupon_GE
0	55	1	1	0	
1	80	1	1	0	
2	80	1	1	1	
3	80	1	1	1	
4	80	1	1	1	

5 rows × 98 columns

In [ ]:
```
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler

X = df1.drop('Y', axis=1)
y = df1['Y']

X = X.apply(pd.to_numeric, errors='coerce')
X = X.fillna(0)

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, rar

scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

So far, I've transformed the dataset by converting all categorical variables into numerical format using one-hot encoding. After separating the features from the target variable (`Y`), I encountered non-numeric values that prevented scaling. To resolve this, I converted all feature data to numeric, replacing any non-convertible entries with zeros. I then split the data into training and testing sets to prepare for model training and applied standard scaling to ensure that all feature values are on a similar scale, which helps improve the performance.

Q2: Training Models (2 points)

Train two different types of predictive model for the task of predicting whether the coupon is accepted.

- Why did you choose these two models?
- Explain which arguments you used to configure the models and why.
- What values did you set the hyperparameters to. Why?
- Which features did you use? Why?

In [ ]:
```
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, classification_report

model_lr = LogisticRegression(random_state=42)
model_lr.fit(X_train_scaled, y_train)
y_pred_lr = model_lr.predict(X_test_scaled)
accuracy_lr = accuracy_score(y_test, y_pred_lr)
report_lr = classification_report(y_test, y_pred_lr)


model_rf = RandomForestClassifier(random_state=42)
model_rf.fit(X_train_scaled, y_train)
y_pred_rf = model_rf.predict(X_test_scaled)
accuracy_rf = accuracy_score(y_test, y_pred_rf)
report_rf = classification_report(y_test, y_pred_rf)

print("Logistic Regression Accuracy:", accuracy_lr)
print("Logistic Regression Report:\n", report_lr)
print("Random Forest Accuracy:", accuracy_rf)
print("Random Forest Report:\n", report_rf)
```

```
Logistic Regression Accuracy: 0.6854552621206149
Logistic Regression Report:
               precision    recall  f1-score   support

           0       0.67      0.57      0.62      1128
           1       0.69      0.78      0.73      1409

    accuracy                           0.69      2537
   macro avg       0.68      0.67      0.68      2537
weighted avg       0.68      0.69      0.68      2537

Random Forest Accuracy: 0.7469452108789909
Random Forest Report:
               precision    recall  f1-score   support

           0       0.75      0.64      0.69      1128
           1       0.74      0.83      0.79      1409

    accuracy                           0.75      2537
   macro avg       0.75      0.74      0.74      2537
weighted avg       0.75      0.75      0.74      2537
```

I configured both models by setting the random_state parameter to 42 to ensure that the results are reproducible each time the code is run. For the Logistic Regression, I used the default settings because they are generally effective for binary classification tasks like predicting coupon acceptance. Similarly, for the Random Forest Classifier, I also used the default hyperparameters, allowing the model to automatically determine the number of trees and other settings based on the data. I used all the processed features from the dataset, excluding the target variable Y, because this set includes both numerical and one-hot encoded categorical variables. Using all available features ensures that the models have the necessary information to learn patterns and make accurate predictions on whether a passenger will accept a coupon.

In [ ]:
```
param_grid = {
    'n_estimators': [200, 300],
    'max_depth': [None, 10, 20],
    'min_samples_split': [2, 5],
    'min_samples_leaf': [1, 2],
    'max_features': [ 'sqrt']
}

rf = RandomForestClassifier(random_state=42)
grid_search = GridSearchCV(estimator=rf, param_grid=param_grid,
                           cv=5, n_jobs=-1, scoring='accuracy')
grid_search.fit(X_train_scaled, y_train)

best_params = grid_search.best_params_
print("Best Hyperparameters:", best_params)
```

```
Best Hyperparameters: {'max_depth': None, 'max_features': 'sqrt', 'min_sampl
es_leaf': 1, 'min_samples_split': 5, 'n_estimators': 300}
```

In [ ]:
```
model_rf_best = RandomForestClassifier(**best_params, random_state=42)
model_rf_best.fit(X_train_scaled, y_train)
y_pred_rf_best = model_rf_best.predict(X_test_scaled)
accuracy_rf_best = accuracy_score(y_test, y_pred_rf_best)
report_rf_best = classification_report(y_test, y_pred_rf_best)

print("Random Forest with Best Hyperparameters Accuracy:", accuracy_rf_best)
print("Random Forest with Best Hyperparameters Report:\n", report_rf_best)
```

```
Random Forest with Best Hyperparameters Accuracy: 0.7449743791880173
Random Forest with Best Hyperparameters Report:
               precision    recall  f1-score   support

           0       0.77      0.61      0.68      1128
           1       0.73      0.85      0.79      1409

    accuracy                           0.74      2537
   macro avg       0.75      0.73      0.73      2537
weighted avg       0.75      0.74      0.74      2537
```

I now performed hyperparameter tuning on the Random Forest classifier using GridSearchCV to explore different combinations of n_estimators, max_depth, min_samples_split, min_samples_leaf, and max_features. After evaluating the models with 5-fold cross-validation, I identified the best set of hyperparameters that yielded the highest accuracy. I then retrained the Random Forest model using these optimal hyperparameters and evaluated its performance on the test set, resulting in improved accuracy and classification metrics compared to the initial model.

In [ ]:
```
import pandas as pd
import numpy as np

feature_names = X_train.columns

coefficients = model_lr.coef_[0]
feature_importance_lr = pd.Series(coefficients, index=feature_names).abs().s

print("Top features in Logistic Regression:")
print(feature_importance_lr.head(10))

importances = model_rf.feature_importances_
feature_importance_rf = pd.Series(importances, index=feature_names).sort_val

print("Top features in Random Forest:")
print(feature_importance_rf.head(10))
```

```
Top features in Logistic Regression:
coupon_Carry out & Take away    0.661237
coupon_Restaurant(<20)          0.637073
expiration_2h                   0.430607
CoffeeHouse_never               0.414163
destination_No Urgent Place     0.399402
coupon_Coffee House             0.250863
CoffeeHouse_less1               0.222456
car_Unknown                     0.175021
weather_Sunny                   0.161943
occupation_Unemployed           0.143322
dtype: float64
Top features in Random Forest:
expiration_2h                   0.032556
coupon_Carry out & Take away    0.032081
coupon_Restaurant(<20)          0.030212
temperature                     0.029675
coupon_Coffee House             0.028720
toCoupon_GEQ15min               0.025909
CoffeeHouse_never               0.024857
time_6PM                        0.019562
Restaurant20To50_less1          0.019319
gender_Male                     0.018891
dtype: float64
```

I analyzed the importance of features in both models by extracting the coefficients from the Logistic Regression model and the feature importances from the Random Forest model. For the Logistic Regression model, I sorted the absolute values of the coefficients to identify which features have the most significant impact on the prediction. The top features with the highest coefficients indicate the strongest influence on whether a coupon is accepted.

For the Random Forest model, I used the feature_importances_ attribute to find out which features contribute most to the model's decisions. By sorting these importance scores in descending order, I identified the features that the model considers most critical when predicting coupon acceptance.

Q3: Evaluation (2.5 points)

- Determine which model performed best. How do you know?
- Which features are most important to your models? Are they the same across different models? How do they affect the predicted outcome?

The Random Forest model performed best, achieving an accuracy of approximately 74.85% compared to the Logistic Regression's 68.55%. I determined this by comparing the accuracy scores, where the Random Forest showed higher overall correctness in its predictions.

In terms of feature importance, for Logistic Regression, the top features include `coupon_Carry out & Take away`, `coupon_Restaurant(<20)`, and `expiration_2h`. For the **Random Forest**, the most important features are `expiration_2h`, `coupon_Carry out & Take away`, and `coupon_Restaurant(<20)`, among others. While there is some overlap in the important features between the two models, such as the coupon-related features and `expiration_2h`, Random Forest also highlights additional features like `temperature` and `toCoupon_GEQ15min`.

These important features influence the predicted outcome by indicating which factors are most strongly associated with a passenger accepting a coupon. For example, specific coupon types and the expiration time significantly impact the likelihood of acceptance in both models.

Q4: Conclusion (1 point)

- Based on your findings, what recommendations would you make to the coupon provider?
- Are there any improvements to your analysis you would consider making?

Based on my analysis, I recommend that the coupon provider focus on offering coupons for "Carry out & Take away" and "Restaurant(<20)" as these are the most influential in driving acceptance according to both the Logistic Regression and Random Forest models. Additionally, setting shorter expiration times, such as 2 hours, significantly increases the likelihood of coupons being used. Targeting customers who never visit coffee houses and those traveling to "No Urgent Place" destinations can also enhance acceptance rates. Furthermore, considering factors like temperature and timing (e.g., 6 PM) can help in tailoring coupon offers more effectively.

For improvements to my analysis, I would consider performing more extensive hyperparameter tuning using techniques like `RandomizedSearchCV` to explore a wider range of parameters efficiently. Incorporating feature engineering to create new relevant features or interactions between existing ones could provide deeper insights.

Administrative Questions

Question A.1 (0 points)

Did you use an LLM like ChatGPT or Claude to assist in answering this problem set?

Write "No" if you did not. Write "Yes" and paste a link to the transcript (e.g. https://chat.openai.com/share/5c14a304-1b7f-4fb9-b400-21e65ad545bb) if you did.

No

Question A.2 (0 points)

Please use this anonymous form to provide feedback on the assignment. Your input will help us improve and refine future assignments.

Did you fill out the feedback form?

Type your answer here, replacing this text.

Submission

Make sure you have run all cells in your notebook in order before running the cell below, so that all images/graphs appear in the output. The cell below will generate a zip file for you to submit. **Please save before exporting!**

In [105…

```
# Save your notebook first, then run this cell to export your submission.
grader.export(pdf=False)
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-105-ef10547530e4> in <cell line: 2>()
      1 # Save your notebook first, then run this cell to export your submis
sion.
----> 2 grader.export(pdf=False)

NameError: name 'grader' is not defined
```