Chapter 8 - Tree Based Methods

September 14, 2023

[97]:
```
!pip install ISLP
```

```
Requirement already satisfied: ISLP in
/Users/barnana/anaconda3/lib/python3.10/site-packages (0.3.16)
Requirement already satisfied: pandas>=0.0 in
/Users/barnana/anaconda3/lib/python3.10/site-packages (from ISLP) (1.5.3)
Requirement already satisfied: numpy>=0.0 in
/Users/barnana/anaconda3/lib/python3.10/site-packages (from ISLP) (1.25.1)
Requirement already satisfied: pygam>=0.0 in
/Users/barnana/anaconda3/lib/python3.10/site-packages (from ISLP) (0.9.0)
Requirement already satisfied: scipy>=0.9 in
/Users/barnana/anaconda3/lib/python3.10/site-packages (from ISLP) (1.11.1)
Requirement already satisfied: jupyter>=0.0 in
/Users/barnana/anaconda3/lib/python3.10/site-packages (from ISLP) (1.0.0)
Requirement already satisfied: statsmodels>=0.13 in
/Users/barnana/anaconda3/lib/python3.10/site-packages (from ISLP) (0.13.5)
Requirement already satisfied: lifelines>=0.0 in
/Users/barnana/anaconda3/lib/python3.10/site-packages (from ISLP) (0.27.7)
Requirement already satisfied: scikit-learn>=1.2 in
/Users/barnana/anaconda3/lib/python3.10/site-packages (from ISLP) (1.3.0)
Requirement already satisfied: joblib>=0.0 in
/Users/barnana/anaconda3/lib/python3.10/site-packages (from ISLP) (1.1.1)
Requirement already satisfied: matplotlib>=3.3.3 in
/Users/barnana/anaconda3/lib/python3.10/site-packages (from ISLP) (3.7.0)
Requirement already satisfied: lxml>=0.0 in
/Users/barnana/anaconda3/lib/python3.10/site-packages (from ISLP) (4.9.1)
Requirement already satisfied: ipykernel in
/Users/barnana/anaconda3/lib/python3.10/site-packages (from jupyter>=0.0->ISLP)
(6.19.2)
Requirement already satisfied: nbconvert in
/Users/barnana/anaconda3/lib/python3.10/site-packages (from jupyter>=0.0->ISLP)
(6.5.4)
Requirement already satisfied: qtconsole in
/Users/barnana/anaconda3/lib/python3.10/site-packages (from jupyter>=0.0->ISLP)
(5.4.0)
Requirement already satisfied: jupyter-console in
/Users/barnana/anaconda3/lib/python3.10/site-packages (from jupyter>=0.0->ISLP)
(6.6.3)
```

```
Requirement already satisfied: ipywidgets in
/Users/barnana/anaconda3/lib/python3.10/site-packages (from jupyter>=0.0->ISLP)
(8.0.7)
Requirement already satisfied: notebook in
/Users/barnana/anaconda3/lib/python3.10/site-packages (from jupyter>=0.0->ISLP)
(6.5.2)
Requirement already satisfied: autograd>=1.5 in
/Users/barnana/anaconda3/lib/python3.10/site-packages (from
lifelines>=0.0->ISLP) (1.6.2)
Requirement already satisfied: formulaic>=0.2.2 in
/Users/barnana/anaconda3/lib/python3.10/site-packages (from
lifelines>=0.0->ISLP) (0.6.4)
Requirement already satisfied: autograd-gamma>=0.3 in
/Users/barnana/anaconda3/lib/python3.10/site-packages (from
lifelines>=0.0->ISLP) (0.5.0)
Requirement already satisfied: python-dateutil>=2.7 in
/Users/barnana/anaconda3/lib/python3.10/site-packages (from
matplotlib>=3.3.3->ISLP) (2.8.2)
Requirement already satisfied: cycler>=0.10 in
/Users/barnana/anaconda3/lib/python3.10/site-packages (from
matplotlib>=3.3.3->ISLP) (0.11.0)
Requirement already satisfied: pillow>=6.2.0 in
/Users/barnana/anaconda3/lib/python3.10/site-packages (from
matplotlib>=3.3.3->ISLP) (9.4.0)
Requirement already satisfied: packaging>=20.0 in
/Users/barnana/anaconda3/lib/python3.10/site-packages (from
matplotlib>=3.3.3->ISLP) (22.0)
Requirement already satisfied: contourpy>=1.0.1 in
/Users/barnana/anaconda3/lib/python3.10/site-packages (from
matplotlib>=3.3.3->ISLP) (1.0.5)
Requirement already satisfied: kiwisolver>=1.0.1 in
/Users/barnana/anaconda3/lib/python3.10/site-packages (from
matplotlib>=3.3.3->ISLP) (1.4.4)
Requirement already satisfied: pyparsing>=2.3.1 in
/Users/barnana/anaconda3/lib/python3.10/site-packages (from
matplotlib>=3.3.3->ISLP) (3.0.9)
Requirement already satisfied: fonttools>=4.22.0 in
/Users/barnana/anaconda3/lib/python3.10/site-packages (from
matplotlib>=3.3.3->ISLP) (4.25.0)
Requirement already satisfied: pytz>=2020.1 in
/Users/barnana/anaconda3/lib/python3.10/site-packages (from pandas>=0.0->ISLP)
(2022.7)
Requirement already satisfied: progressbar2<5.0.0,>=4.2.0 in
/Users/barnana/anaconda3/lib/python3.10/site-packages (from pygam>=0.0->ISLP)
(4.2.0)
Requirement already satisfied: threadpoolctl>=2.0.0 in
/Users/barnana/anaconda3/lib/python3.10/site-packages (from scikit-
learn>=1.2->ISLP) (3.2.0)
```

```
Requirement already satisfied: patsy>=0.5.2 in
/Users/barnana/anaconda3/lib/python3.10/site-packages (from
statsmodels>=0.13->ISLP) (0.5.3)
Requirement already satisfied: future>=0.15.2 in
/Users/barnana/anaconda3/lib/python3.10/site-packages (from
autograd>=1.5->lifelines>=0.0->ISLP) (0.18.3)
Requirement already satisfied: wrapt>=1.0 in
/Users/barnana/anaconda3/lib/python3.10/site-packages (from
formulaic>=0.2.2->lifelines>=0.0->ISLP) (1.14.1)
Requirement already satisfied: typing-extensions>=4.2.0 in
/Users/barnana/anaconda3/lib/python3.10/site-packages (from
formulaic>=0.2.2->lifelines>=0.0->ISLP) (4.4.0)
Requirement already satisfied: astor>=0.8 in
/Users/barnana/anaconda3/lib/python3.10/site-packages (from
formulaic>=0.2.2->lifelines>=0.0->ISLP) (0.8.1)
Requirement already satisfied: interface-meta>=1.2.0 in
/Users/barnana/anaconda3/lib/python3.10/site-packages (from
formulaic>=0.2.2->lifelines>=0.0->ISLP) (1.3.0)
Requirement already satisfied: six in
/Users/barnana/anaconda3/lib/python3.10/site-packages (from
patsy>=0.5.2->statsmodels>=0.13->ISLP) (1.16.0)
Requirement already satisfied: python-utils>=3.0.0 in
/Users/barnana/anaconda3/lib/python3.10/site-packages (from
progressbar2<5.0.0,>=4.2.0->pygam>=0.0->ISLP) (3.7.0)
Requirement already satisfied: comm>=0.1.1 in
/Users/barnana/anaconda3/lib/python3.10/site-packages (from
ipykernel->jupyter>=0.0->ISLP) (0.1.2)
Requirement already satisfied: ipython>=7.23.1 in
/Users/barnana/anaconda3/lib/python3.10/site-packages (from
ipykernel->jupyter>=0.0->ISLP) (8.10.0)
Requirement already satisfied: pyzmq>=17 in
/Users/barnana/anaconda3/lib/python3.10/site-packages (from
ipykernel->jupyter>=0.0->ISLP) (23.2.0)
Requirement already satisfied: psutil in
/Users/barnana/anaconda3/lib/python3.10/site-packages (from
ipykernel->jupyter>=0.0->ISLP) (5.9.0)
Requirement already satisfied: traitlets>=5.4.0 in
/Users/barnana/anaconda3/lib/python3.10/site-packages (from
ipykernel->jupyter>=0.0->ISLP) (5.7.1)
Requirement already satisfied: tornado>=6.1 in
/Users/barnana/anaconda3/lib/python3.10/site-packages (from
ipykernel->jupyter>=0.0->ISLP) (6.1)
Requirement already satisfied: matplotlib-inline>=0.1 in
/Users/barnana/anaconda3/lib/python3.10/site-packages (from
ipykernel->jupyter>=0.0->ISLP) (0.1.6)
Requirement already satisfied: nest-asyncio in
/Users/barnana/anaconda3/lib/python3.10/site-packages (from
ipykernel->jupyter>=0.0->ISLP) (1.5.6)
```

```
Requirement already satisfied: jupyter-client>=6.1.12 in
/Users/barnana/anaconda3/lib/python3.10/site-packages (from
ipykernel->jupyter>=0.0->ISLP) (7.3.4)
Requirement already satisfied: debugpy>=1.0 in
/Users/barnana/anaconda3/lib/python3.10/site-packages (from
ipykernel->jupyter>=0.0->ISLP) (1.5.1)
Requirement already satisfied: appnope in
/Users/barnana/anaconda3/lib/python3.10/site-packages (from
ipykernel->jupyter>=0.0->ISLP) (0.1.2)
Requirement already satisfied: widgetsnbextension~=4.0.7 in
/Users/barnana/anaconda3/lib/python3.10/site-packages (from
ipywidgets->jupyter>=0.0->ISLP) (4.0.8)
Requirement already satisfied: jupyterlab-widgets~=3.0.7 in
/Users/barnana/anaconda3/lib/python3.10/site-packages (from
ipywidgets->jupyter>=0.0->ISLP) (3.0.8)
Requirement already satisfied: prompt-toolkit>=3.0.30 in
/Users/barnana/anaconda3/lib/python3.10/site-packages (from jupyter-
console->jupyter>=0.0->ISLP) (3.0.36)
Requirement already satisfied: jupyter-core!=5.0.*,>=4.12 in
/Users/barnana/anaconda3/lib/python3.10/site-packages (from jupyter-
console->jupyter>=0.0->ISLP) (5.2.0)
Requirement already satisfied: pygments in
/Users/barnana/anaconda3/lib/python3.10/site-packages (from jupyter-
console->jupyter>=0.0->ISLP) (2.11.2)
Requirement already satisfied: pandocfilters>=1.4.1 in
/Users/barnana/anaconda3/lib/python3.10/site-packages (from
nbconvert->jupyter>=0.0->ISLP) (1.5.0)
Requirement already satisfied: tinycss2 in
/Users/barnana/anaconda3/lib/python3.10/site-packages (from
nbconvert->jupyter>=0.0->ISLP) (1.2.1)
Requirement already satisfied: entrypoints>=0.2.2 in
/Users/barnana/anaconda3/lib/python3.10/site-packages (from
nbconvert->jupyter>=0.0->ISLP) (0.4)
Requirement already satisfied: MarkupSafe>=2.0 in
/Users/barnana/anaconda3/lib/python3.10/site-packages (from
nbconvert->jupyter>=0.0->ISLP) (2.1.1)
Requirement already satisfied: nbformat>=5.1 in
/Users/barnana/anaconda3/lib/python3.10/site-packages (from
nbconvert->jupyter>=0.0->ISLP) (5.7.0)
Requirement already satisfied: bleach in
/Users/barnana/anaconda3/lib/python3.10/site-packages (from
nbconvert->jupyter>=0.0->ISLP) (4.1.0)
Requirement already satisfied: jinja2>=3.0 in
/Users/barnana/anaconda3/lib/python3.10/site-packages (from
nbconvert->jupyter>=0.0->ISLP) (3.1.2)
Requirement already satisfied: beautifulsoup4 in
/Users/barnana/anaconda3/lib/python3.10/site-packages (from
nbconvert->jupyter>=0.0->ISLP) (4.11.1)
```

```
Requirement already satisfied: mistune<2,>=0.8.1 in
/Users/barnana/anaconda3/lib/python3.10/site-packages (from
nbconvert->jupyter>=0.0->ISLP) (0.8.4)
Requirement already satisfied: nbclient>=0.5.0 in
/Users/barnana/anaconda3/lib/python3.10/site-packages (from
nbconvert->jupyter>=0.0->ISLP) (0.5.13)
Requirement already satisfied: jupyterlab-pygments in
/Users/barnana/anaconda3/lib/python3.10/site-packages (from
nbconvert->jupyter>=0.0->ISLP) (0.1.2)
Requirement already satisfied: defusedxml in
/Users/barnana/anaconda3/lib/python3.10/site-packages (from
nbconvert->jupyter>=0.0->ISLP) (0.7.1)
Requirement already satisfied: ipython-genutils in
/Users/barnana/anaconda3/lib/python3.10/site-packages (from
notebook->jupyter>=0.0->ISLP) (0.2.0)
Requirement already satisfied: prometheus-client in
/Users/barnana/anaconda3/lib/python3.10/site-packages (from
notebook->jupyter>=0.0->ISLP) (0.14.1)
Requirement already satisfied: Send2Trash>=1.8.0 in
/Users/barnana/anaconda3/lib/python3.10/site-packages (from
notebook->jupyter>=0.0->ISLP) (1.8.0)
Requirement already satisfied: terminado>=0.8.3 in
/Users/barnana/anaconda3/lib/python3.10/site-packages (from
notebook->jupyter>=0.0->ISLP) (0.17.1)
Requirement already satisfied: argon2-cffi in
/Users/barnana/anaconda3/lib/python3.10/site-packages (from
notebook->jupyter>=0.0->ISLP) (21.3.0)
Requirement already satisfied: nbclassic>=0.4.7 in
/Users/barnana/anaconda3/lib/python3.10/site-packages (from
notebook->jupyter>=0.0->ISLP) (0.5.2)
Requirement already satisfied: qtpy>=2.0.1 in
/Users/barnana/anaconda3/lib/python3.10/site-packages (from
qtconsole->jupyter>=0.0->ISLP) (2.2.0)
Requirement already satisfied: decorator in
/Users/barnana/anaconda3/lib/python3.10/site-packages (from
ipython>=7.23.1->ipykernel->jupyter>=0.0->ISLP) (5.1.1)
Requirement already satisfied: jedi>=0.16 in
/Users/barnana/anaconda3/lib/python3.10/site-packages (from
ipython>=7.23.1->ipykernel->jupyter>=0.0->ISLP) (0.18.1)
Requirement already satisfied: stack-data in
/Users/barnana/anaconda3/lib/python3.10/site-packages (from
ipython>=7.23.1->ipykernel->jupyter>=0.0->ISLP) (0.2.0)
Requirement already satisfied: pexpect>4.3 in
/Users/barnana/anaconda3/lib/python3.10/site-packages (from
ipython>=7.23.1->ipykernel->jupyter>=0.0->ISLP) (4.8.0)
Requirement already satisfied: pickleshare in
/Users/barnana/anaconda3/lib/python3.10/site-packages (from
ipython>=7.23.1->ipykernel->jupyter>=0.0->ISLP) (0.7.5)
```

```
Requirement already satisfied: backcall in
/Users/barnana/anaconda3/lib/python3.10/site-packages (from
ipython>=7.23.1->ipykernel->jupyter>=0.0->ISLP) (0.2.0)
Requirement already satisfied: platformdirs>=2.5 in
/Users/barnana/anaconda3/lib/python3.10/site-packages (from jupyter-
core!=5.0.*,>=4.12->jupyter-console->jupyter>=0.0->ISLP) (2.5.2)
Requirement already satisfied: notebook-shim>=0.1.0 in
/Users/barnana/anaconda3/lib/python3.10/site-packages (from
nbclassic>=0.4.7->notebook->jupyter>=0.0->ISLP) (0.2.2)
Requirement already satisfied: jupyter-server>=1.8 in
/Users/barnana/anaconda3/lib/python3.10/site-packages (from
nbclassic>=0.4.7->notebook->jupyter>=0.0->ISLP) (1.23.4)
Requirement already satisfied: fastjsonschema in
/Users/barnana/anaconda3/lib/python3.10/site-packages (from
nbformat>=5.1->nbconvert->jupyter>=0.0->ISLP) (2.16.2)
Requirement already satisfied: jsonschema>=2.6 in
/Users/barnana/anaconda3/lib/python3.10/site-packages (from
nbformat>=5.1->nbconvert->jupyter>=0.0->ISLP) (4.17.3)
Requirement already satisfied: wcwidth in
/Users/barnana/anaconda3/lib/python3.10/site-packages (from prompt-
toolkit>=3.0.30->jupyter-console->jupyter>=0.0->ISLP) (0.2.5)
Requirement already satisfied: ptyprocess in
/Users/barnana/anaconda3/lib/python3.10/site-packages (from
terminado>=0.8.3->notebook->jupyter>=0.0->ISLP) (0.7.0)
Requirement already satisfied: argon2-cffi-bindings in
/Users/barnana/anaconda3/lib/python3.10/site-packages (from
argon2-cffi->notebook->jupyter>=0.0->ISLP) (21.2.0)
Requirement already satisfied: soupsieve>1.2 in
/Users/barnana/anaconda3/lib/python3.10/site-packages (from
beautifulsoup4->nbconvert->jupyter>=0.0->ISLP) (2.3.2.post1)
Requirement already satisfied: webencodings in
/Users/barnana/anaconda3/lib/python3.10/site-packages (from
bleach->nbconvert->jupyter>=0.0->ISLP) (0.5.1)
Requirement already satisfied: parso<0.9.0,>=0.8.0 in
/Users/barnana/anaconda3/lib/python3.10/site-packages (from
jedi>=0.16->ipython>=7.23.1->ipykernel->jupyter>=0.0->ISLP) (0.8.3)
Requirement already satisfied: pyrsistent!=0.17.0,!=0.17.1,!=0.17.2,>=0.14.0 in
/Users/barnana/anaconda3/lib/python3.10/site-packages (from
jsonschema>=2.6->nbformat>=5.1->nbconvert->jupyter>=0.0->ISLP) (0.18.0)
Requirement already satisfied: attrs>=17.4.0 in
/Users/barnana/anaconda3/lib/python3.10/site-packages (from
jsonschema>=2.6->nbformat>=5.1->nbconvert->jupyter>=0.0->ISLP) (22.1.0)
Requirement already satisfied: websocket-client in
/Users/barnana/anaconda3/lib/python3.10/site-packages (from jupyter-
server>=1.8->nbclassic>=0.4.7->notebook->jupyter>=0.0->ISLP) (0.58.0)
Requirement already satisfied: anyio<4,>=3.1.0 in
/Users/barnana/anaconda3/lib/python3.10/site-packages (from jupyter-
server>=1.8->nbclassic>=0.4.7->notebook->jupyter>=0.0->ISLP) (3.5.0)
```

```
Requirement already satisfied: cffi>=1.0.1 in
/Users/barnana/anaconda3/lib/python3.10/site-packages (from argon2-cffi-
bindings->argon2-cffi->notebook->jupyter>=0.0->ISLP) (1.15.1)
Requirement already satisfied: executing in
/Users/barnana/anaconda3/lib/python3.10/site-packages (from stack-
data->ipython>=7.23.1->ipykernel->jupyter>=0.0->ISLP) (0.8.3)
Requirement already satisfied: pure-eval in
/Users/barnana/anaconda3/lib/python3.10/site-packages (from stack-
data->ipython>=7.23.1->ipykernel->jupyter>=0.0->ISLP) (0.2.2)
Requirement already satisfied: asttokens in
/Users/barnana/anaconda3/lib/python3.10/site-packages (from stack-
data->ipython>=7.23.1->ipykernel->jupyter>=0.0->ISLP) (2.0.5)
Requirement already satisfied: sniffio>=1.1 in
/Users/barnana/anaconda3/lib/python3.10/site-packages (from
anyio<4,>=3.1.0->jupyter-
server>=1.8->nbclassic>=0.4.7->notebook->jupyter>=0.0->ISLP) (1.2.0)
Requirement already satisfied: idna>=2.8 in
/Users/barnana/anaconda3/lib/python3.10/site-packages (from
anyio<4,>=3.1.0->jupyter-
server>=1.8->nbclassic>=0.4.7->notebook->jupyter>=0.0->ISLP) (3.4)
Requirement already satisfied: pycparser in
/Users/barnana/anaconda3/lib/python3.10/site-packages (from
cffi>=1.0.1->argon2-cffi-bindings->argon2-cffi->notebook->jupyter>=0.0->ISLP)
(2.21)
```

[98]:
```
!pip install pytorch-lightning
```

```
Requirement already satisfied: pytorch-lightning in
/Users/barnana/anaconda3/lib/python3.10/site-packages (2.0.6)
Requirement already satisfied: typing-extensions>=4.0.0 in
/Users/barnana/anaconda3/lib/python3.10/site-packages (from pytorch-lightning)
(4.4.0)
Requirement already satisfied: fsspec[http]>2021.06.0 in
/Users/barnana/anaconda3/lib/python3.10/site-packages (from pytorch-lightning)
(2022.11.0)
Requirement already satisfied: tqdm>=4.57.0 in
/Users/barnana/anaconda3/lib/python3.10/site-packages (from pytorch-lightning)
(4.64.1)
Requirement already satisfied: lightning-utilities>=0.7.0 in
/Users/barnana/anaconda3/lib/python3.10/site-packages (from pytorch-lightning)
(0.9.0)
Requirement already satisfied: PyYAML>=5.4 in
/Users/barnana/anaconda3/lib/python3.10/site-packages (from pytorch-lightning)
(6.0)
Requirement already satisfied: torch>=1.11.0 in
/Users/barnana/anaconda3/lib/python3.10/site-packages (from pytorch-lightning)
(1.12.1)
Requirement already satisfied: torchmetrics>=0.7.0 in
```

```
/Users/barnana/anaconda3/lib/python3.10/site-packages (from pytorch-lightning)
(1.0.1)
Requirement already satisfied: numpy>=1.17.2 in
/Users/barnana/anaconda3/lib/python3.10/site-packages (from pytorch-lightning)
(1.25.1)
Requirement already satisfied: packaging>=17.1 in
/Users/barnana/anaconda3/lib/python3.10/site-packages (from pytorch-lightning)
(22.0)
Requirement already satisfied: aiohttp!=4.0.0a0,!=4.0.0a1 in
/Users/barnana/anaconda3/lib/python3.10/site-packages (from
fsspec[http]>2021.06.0->pytorch-lightning) (3.8.5)
Requirement already satisfied: requests in
/Users/barnana/anaconda3/lib/python3.10/site-packages (from
fsspec[http]>2021.06.0->pytorch-lightning) (2.28.1)
Requirement already satisfied: async-timeout<5.0,>=4.0.0a3 in
/Users/barnana/anaconda3/lib/python3.10/site-packages (from
aiohttp!=4.0.0a0,!=4.0.0a1->fsspec[http]>2021.06.0->pytorch-lightning) (4.0.2)
Requirement already satisfied: multidict<7.0,>=4.5 in
/Users/barnana/anaconda3/lib/python3.10/site-packages (from
aiohttp!=4.0.0a0,!=4.0.0a1->fsspec[http]>2021.06.0->pytorch-lightning) (6.0.4)
Requirement already satisfied: aiosignal>=1.1.2 in
/Users/barnana/anaconda3/lib/python3.10/site-packages (from
aiohttp!=4.0.0a0,!=4.0.0a1->fsspec[http]>2021.06.0->pytorch-lightning) (1.3.1)
Requirement already satisfied: charset-normalizer<4.0,>=2.0 in
/Users/barnana/anaconda3/lib/python3.10/site-packages (from
aiohttp!=4.0.0a0,!=4.0.0a1->fsspec[http]>2021.06.0->pytorch-lightning) (2.0.4)
Requirement already satisfied: attrs>=17.3.0 in
/Users/barnana/anaconda3/lib/python3.10/site-packages (from
aiohttp!=4.0.0a0,!=4.0.0a1->fsspec[http]>2021.06.0->pytorch-lightning) (22.1.0)
Requirement already satisfied: frozenlist>=1.1.1 in
/Users/barnana/anaconda3/lib/python3.10/site-packages (from
aiohttp!=4.0.0a0,!=4.0.0a1->fsspec[http]>2021.06.0->pytorch-lightning) (1.4.0)
Requirement already satisfied: yarl<2.0,>=1.0 in
/Users/barnana/anaconda3/lib/python3.10/site-packages (from
aiohttp!=4.0.0a0,!=4.0.0a1->fsspec[http]>2021.06.0->pytorch-lightning) (1.9.2)
Requirement already satisfied: certifi>=2017.4.17 in
/Users/barnana/anaconda3/lib/python3.10/site-packages (from
requests->fsspec[http]>2021.06.0->pytorch-lightning) (2023.5.7)
Requirement already satisfied: idna<4,>=2.5 in
/Users/barnana/anaconda3/lib/python3.10/site-packages (from
requests->fsspec[http]>2021.06.0->pytorch-lightning) (3.4)
Requirement already satisfied: urllib3<1.27,>=1.21.1 in
/Users/barnana/anaconda3/lib/python3.10/site-packages (from
requests->fsspec[http]>2021.06.0->pytorch-lightning) (1.26.14)
```

[99]:
```python
from ISLP import load_data
import pandas as pd
import numpy as np
import seaborn as sns
import math
import matplotlib.pyplot as plt
import statsmodels.api as sm
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.model_selection import train_test_split, GridSearchCV, KFold,␣
 ↪cross_val_score
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error
from sklearn.linear_model import RidgeCV
from sklearn.linear_model import LassoCV
from sklearn.decomposition import PCA
from sklearn.pipeline import Pipeline
from sklearn.cross_decomposition import PLSRegression
import itertools
from sklearn.tree import plot_tree
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import BaggingRegressor
from sklearn.ensemble import RandomForestRegressor
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.metrics import accuracy_score, confusion_matrix
from sklearn.neighbors import KNeighborsClassifier
from sklearn.linear_model import LogisticRegression
import warnings
warnings.filterwarnings("ignore", category=FutureWarning)
warnings.filterwarnings("ignore", category=DeprecationWarning)
import torch
import torch.nn as nn
import torch.nn.functional as F
from sklearn.metrics import classification_report
```

[100]:
```
# Creating a function to print output in green and bold
# ANSI escape code for green color and bold font
GREEN_BOLD = '\033[1;32m'

# ANSI escape code to reset colors and font style
RESET = '\033[0m'

def print_green_bold(*args):
    text = ' '.join(str(arg) for arg in args)
    print(GREEN_BOLD + text + RESET)
```

1 Chapter 8

1.1 Question 8

In the lab, a classification tree was applied to the Carseats data set after converting Sales into a qualitative response variable. Now we will seek to predict Sales using regression trees and related approaches, treating the response as a quantitative variable. (a) Split the data set into a training set and a test set.

[142]:
```
carseats_c8q8 = load_data('Carseats')
carseats_c8q8.head(20)
```

[142]:
```
    Sales  CompPrice  Income  Advertising  Population  Price ShelveLoc  Age  \
0    9.50        138      73           11         276    120       Bad   42
1   11.22        111      48           16         260     83      Good   65
2   10.06        113      35           10         269     80    Medium   59
3    7.40        117     100            4         466     97    Medium   55
4    4.15        141      64            3         340    128       Bad   38
5   10.81        124     113           13         501     72       Bad   78
6    6.63        115     105            0          45    108    Medium   71
7   11.85        136      81           15         425    120      Good   67
8    6.54        132     110            0         108    124    Medium   76
9    4.69        132     113            0         131    124    Medium   76
10   9.01        121      78            9         150    100       Bad   26
11  11.96        117      94            4         503     94      Good   50
12   3.98        122      35            2         393    136    Medium   62
13  10.96        115      28           11          29     86      Good   53
14  11.17        107     117           11         148    118      Good   52
15   8.71        149      95            5         400    144    Medium   76
16   7.58        118      32            0         284    110      Good   63
17  12.29        147      74           13         251    131      Good   52
18  13.91        110     110            0         408     68      Good   46
19   8.73        129      76           16          58    121    Medium   69

    Education Urban   US
0          17   Yes  Yes
1          10   Yes  Yes
2          12   Yes  Yes
3          14   Yes  Yes
4          13   Yes   No
5          16    No  Yes
6          15   Yes   No
7          10   Yes  Yes
8          10    No   No
9          17    No  Yes
10         10    No  Yes
11         13   Yes  Yes
12         18   Yes   No
```

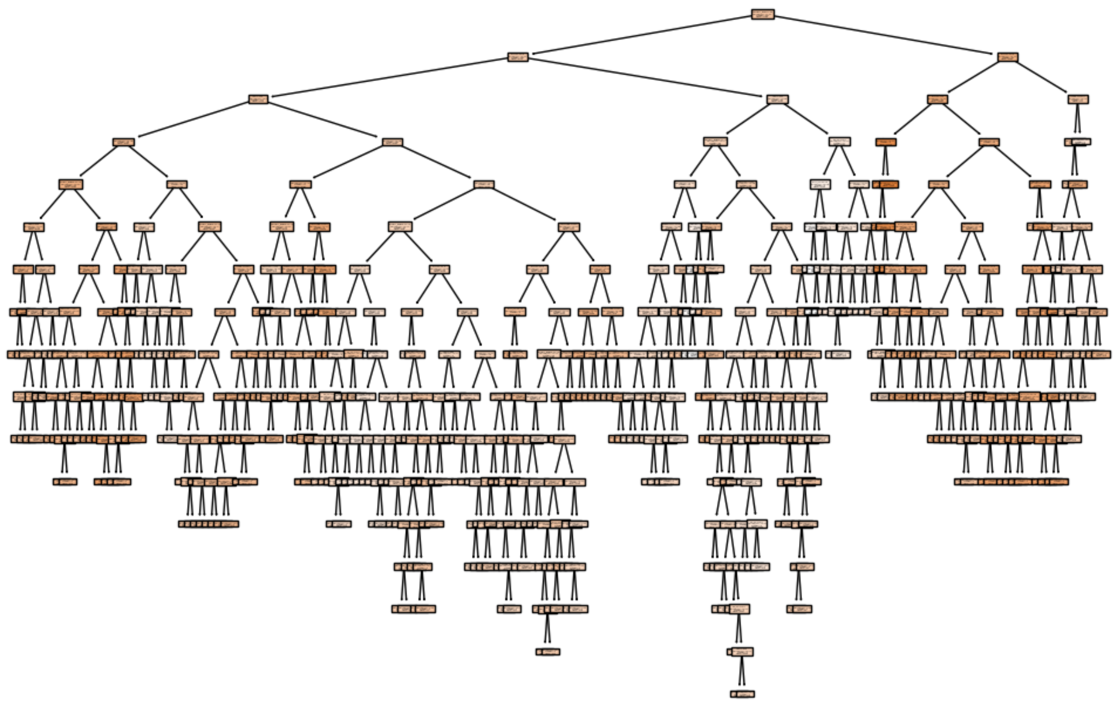
```
13          18   Yes  Yes
14          18   Yes  Yes
15          18    No   No
16          13   Yes   No
17          10   Yes  Yes
18          17    No  Yes
19          12   Yes  Yes
```

[143]:
```
# Convert categorical features into dummy variables
# This is because the DecisionTreeRegressor from scikit-learn cannot handle␣
 ↪categorical/string features directly
carseats_df_c8q8=pd.
 ↪get_dummies(carseats_c8q8,columns=['ShelveLoc','Urban','US'],drop_first=True)

# Separate the features (X) and the target variable (y)
X_c8q8 = carseats_df_c8q8.drop('Sales', axis=1)
y_c8q8 = carseats_df_c8q8['Sales']

# Split the data into a training set and a test set
X_train_c8q8, X_test_c8q8, y_train_c8q8, y_test_c8q8 = train_test_split(X_c8q8,␣
 ↪y_c8q8, test_size=0.2, random_state=42)
```

(b) Fit a regression tree to the training set. Plot the tree, and interpret the results. What test MSE do you obtain?

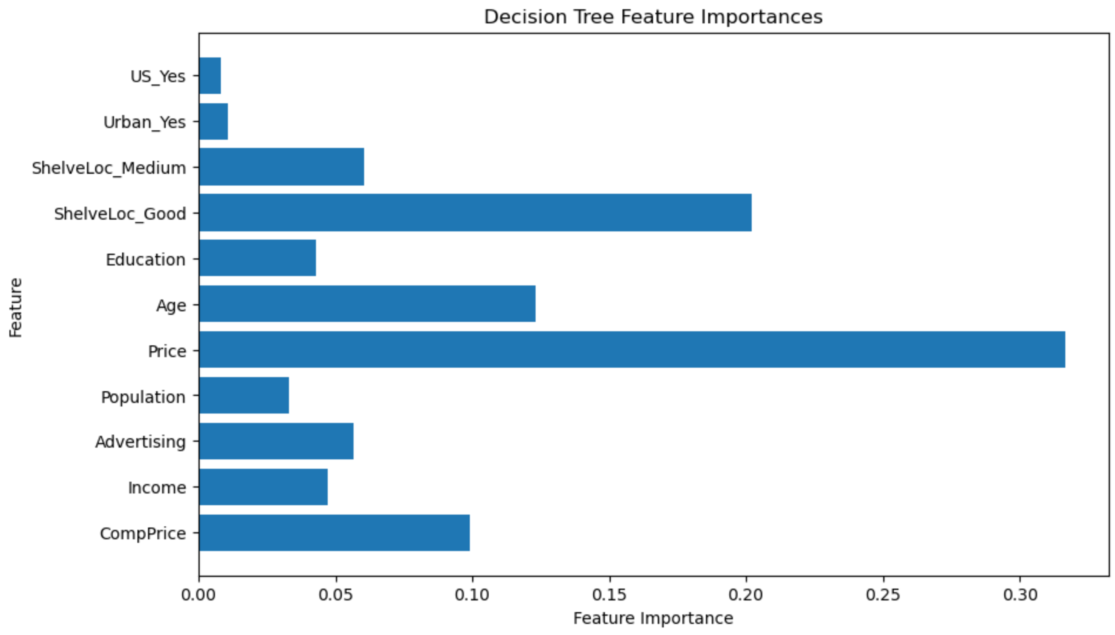
Note that we are simply running the Decision tree regressor without any hyperparameter tuning. We will do hyperparameter tuning (using GridSearchCV), cross validation and pruning in part (c).

[144]:
```
# Fit a regression tree to the training set
reg_tree_c8q8 = DecisionTreeRegressor(random_state=42)
reg_tree_c8q8.fit(X_train_c8q8, y_train_c8q8)

# Plot the decision tree
plt.figure(figsize=(12, 8))
plot_tree(reg_tree_c8q8, feature_names=X_c8q8.columns.tolist(), filled=True)
plt.show()

# Predict on the test set
y_pred_c8q8 = reg_tree_c8q8.predict(X_test_c8q8)

# Calculate the mean squared error (MSE) on the test set
test_mse_c8q8 = mean_squared_error(y_test_c8q8, y_pred_c8q8)
print_green_bold("Test MSE without hyperparameter tuning (GridSearchCV):",␣
 ↪test_mse_c8q8)
```



```
Test MSE without hyperparameter tuning (GridSearchCV):

6.178877500000005
```

[145]:
```
# Get feature importances
feature_importances_c8q8 = reg_tree_c8q8.feature_importances_

# Get the feature names
feature_names_c8q8 = X_c8q8.columns.tolist()

# Create a bar plot for feature importances
plt.figure(figsize=(10, 6))
plt.barh(range(len(feature_importances_c8q8)), feature_importances_c8q8,␣
 ↪align='center')
plt.yticks(range(len(feature_importances_c8q8)), feature_names_c8q8)
plt.xlabel('Feature Importance')
plt.ylabel('Feature')
plt.title('Decision Tree Feature Importances')
plt.show()
```



We find that the MSE value is fairly low in absolute terms which means the model has a high predictive power. The three most important predictors are Price, ShelveLocGood (the ShelveLoc dummy) and Age.

(c) Use cross-validation in order to determine the optimal level of tree complexity. Does pruning the tree improve the test MSE?

1.1.1 GridSearchCV and Cross Validation

[147]:
```
# Define the parameter grid for grid search
param_grid_c8q8 = {
    'max_depth': np.arange(1, 21),
    'min_samples_split': np.arange(2, 11),
    'min_samples_leaf': np.arange(1, 11)
}

# Initialize the GridSearchCV object with the decision tree model and parameter␣
↪grid
grid_search_c8q8 = GridSearchCV(reg_tree_c8q8, param_grid_c8q8, cv=5,␣
↪scoring='neg_mean_squared_error')

# Perform grid search with cross-validation on the training data
grid_search_c8q8.fit(X_train_c8q8, y_train_c8q8)

# Get the best decision tree model from the grid search
best_gridsearch_reg_tree_c8q8 = grid_search_c8q8.best_estimator_
```

```
# Print the best hyperparameters found during grid search
print_green_bold("Best Hyperparameters:", grid_search_c8q8.best_params_)

# Perform cross-validation with the best model
cv_scores_c8q8 = cross_val_score(best_gridsearch_reg_tree_c8q8, X_train_c8q8,␣
↪y_train_c8q8, cv=5, scoring='neg_mean_squared_error')

# Fit the best model to the full training data
best_gridsearch_reg_tree_c8q8.fit(X_train_c8q8, y_train_c8q8)

# Predict on the test set
y_pred_gridsearch_c8q8 = best_gridsearch_reg_tree_c8q8.predict(X_test_c8q8)

# Calculate the mean squared error (MSE) on the test set
test_mse_gridsearch_c8q8 = mean_squared_error(y_test_c8q8,␣
↪y_pred_gridsearch_c8q8)

# Print the average cross-validated MSE and test MSE
print_green_bold("Test MSE with GridSearchCV and Cross-Validation:",␣
↪test_mse_gridsearch_c8q8)
```

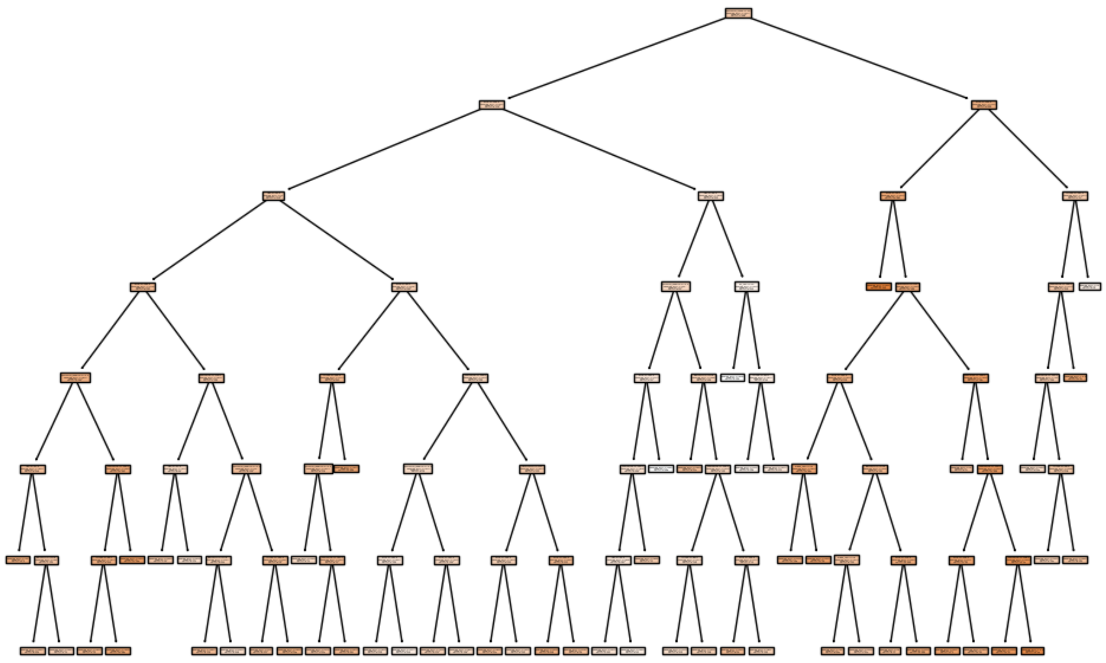
```
Best Hyperparameters: {'max_depth': 7, 'min_samples_leaf': 1,

'min_samples_split': 7}
Test MSE with GridSearchCV and Cross-Validation: 4.618605512774292
```

1.1.2 Pruning

```
[148]: # Prune the decision tree by setting the ccp_alpha parameter
ccp_alpha_values_c8q8 = np.arange(0, 0.1, 0.005)
test_mse_values_c8q8 = []

for ccp_alpha in ccp_alpha_values_c8q8:
    pruned_tree_c8q8 = DecisionTreeRegressor(
        max_depth=best_gridsearch_reg_tree_c8q8.max_depth,
        min_samples_split=best_gridsearch_reg_tree_c8q8.min_samples_split,
        min_samples_leaf=best_gridsearch_reg_tree_c8q8.min_samples_leaf,
        random_state=42,
        ccp_alpha=ccp_alpha
    )

    # Perform cross-validation with the pruned tree
    cv_scores_c8q8 = cross_val_score(pruned_tree_c8q8, X_train_c8q8,␣
↪y_train_c8q8, cv=5, scoring='neg_mean_squared_error')
    avg_cv_mse = -cv_scores_c8q8.mean()  # Calculate the average MSE manually
    test_mse_values_c8q8.append(avg_cv_mse)

# Find the optimal ccp_alpha that minimizes the MSE
```

```
optimal_ccp_alpha_c8q8 = ccp_alpha_values_c8q8[np.argmin(test_mse_values_c8q8)]

# Create the final pruned decision tree with the optimal ccp_alpha
final_pruned_tree_c8q8 = DecisionTreeRegressor(
    max_depth=best_gridsearch_reg_tree_c8q8.max_depth,
    min_samples_split=best_gridsearch_reg_tree_c8q8.min_samples_split,
    min_samples_leaf=best_gridsearch_reg_tree_c8q8.min_samples_leaf,
    random_state=42,
    ccp_alpha=optimal_ccp_alpha_c8q8
)

# Fit the final pruned tree to the full training data
final_pruned_tree_c8q8.fit(X_train_c8q8, y_train_c8q8)

# Predict on the test set using the final pruned tree
y_pred_pruned_c8q8 = final_pruned_tree_c8q8.predict(X_test_c8q8)

# Calculate the mean squared error (MSE) on the test set with the pruned tree
test_mse_pruned_c8q8 = mean_squared_error(y_test_c8q8, y_pred_pruned_c8q8)

# Print the optimal ccp_alpha and test MSE with the pruned tree
print_green_bold("Optimal ccp_alpha:", optimal_ccp_alpha_c8q8)
print_green_bold("Test MSE with Pruned Tree:", test_mse_pruned_c8q8)

# Plot the final pruned decision tree
plt.figure(figsize=(12, 8))
plot_tree(final_pruned_tree_c8q8, feature_names=X_c8q8.columns.tolist(),␣
  ↪filled=True)
plt.show()
```

```
Optimal ccp_alpha: 0.0
Test MSE with Pruned Tree: 4.618605512774292
```



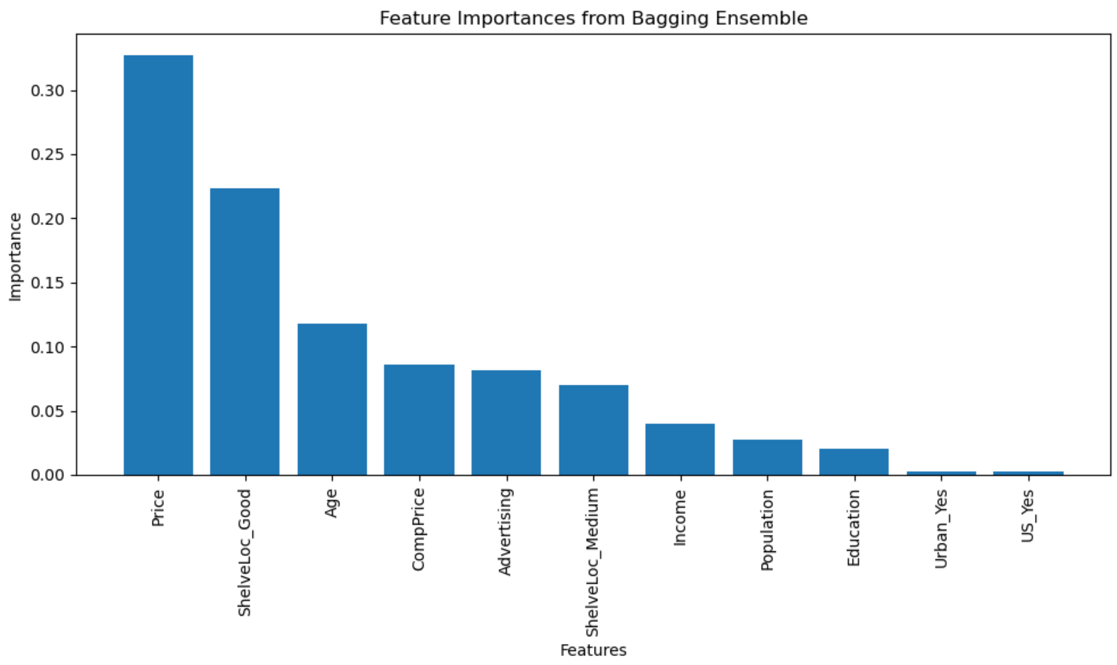
We found that using GridSearchCV and cross validation brings down the MSE from 6.18 to 4.62. If we apply pruning to this optimal model, however, we do not see a further decrease in MSE. This suggests that the decision tree, with the optimal hyperparameters, is already at an appropriate level of complexity and generalization for the data.

(d) Use the bagging approach in order to analyze this data. What test MSE do you obtain? Use the feature_importance_ values to determine which variables are most important.

[149]:
```python
# Use the optimal pruned decision tree regressor as the base estimator
base_tree_c8q8 = final_pruned_tree_c8q8

# Create the BaggingRegressor with 100 base estimators (trees)
bagging_reg_c8q8 = BaggingRegressor(base_estimator=base_tree_c8q8,␣
  ↪n_estimators=100, random_state=42)

# Fit the BaggingRegressor to the training data
bagging_reg_c8q8.fit(X_train_c8q8, y_train_c8q8)

# Predict on the test set using the bagged model
y_pred_bagging_c8q8 = bagging_reg_c8q8.predict(X_test_c8q8)

# Calculate the mean squared error (MSE) on the test set
test_mse_bagging_c8q8 = mean_squared_error(y_test_c8q8, y_pred_bagging_c8q8)
```

```python
# Print the test MSE
print_green_bold("Test MSE with Bagging:", test_mse_bagging_c8q8)

# Calculate feature importances for each tree
feature_importances_per_tree_c8q8 = []
for tree in bagging_reg_c8q8.estimators_:
    feature_importances_per_tree_c8q8.append(tree.feature_importances_)

# Calculate the average feature importances
feature_importances_c8q8 = np.mean(feature_importances_per_tree_c8q8, axis=0)

# Create a dictionary to map feature names to their importances
feature_importance_dict_c8q8 = dict(zip(X_c8q8.columns,␣
 ↪feature_importances_c8q8))

# Sort the features by importance (descending order)
sorted_feature_importances_c8q8 = sorted(feature_importance_dict_c8q8.items(),␣
 ↪key=lambda x: x[1], reverse=True)

# Extract the feature names and importances for plotting
features_c8q8, importances_c8q8 = zip(*sorted_feature_importances_c8q8)

# Plot the feature importances
plt.figure(figsize=(10, 6))
plt.bar(features_c8q8, importances_c8q8)
plt.xticks(rotation=90)
plt.xlabel("Features")
plt.ylabel("Importance")
plt.title("Feature Importances from Bagging Ensemble")
plt.tight_layout()
plt.show()
```

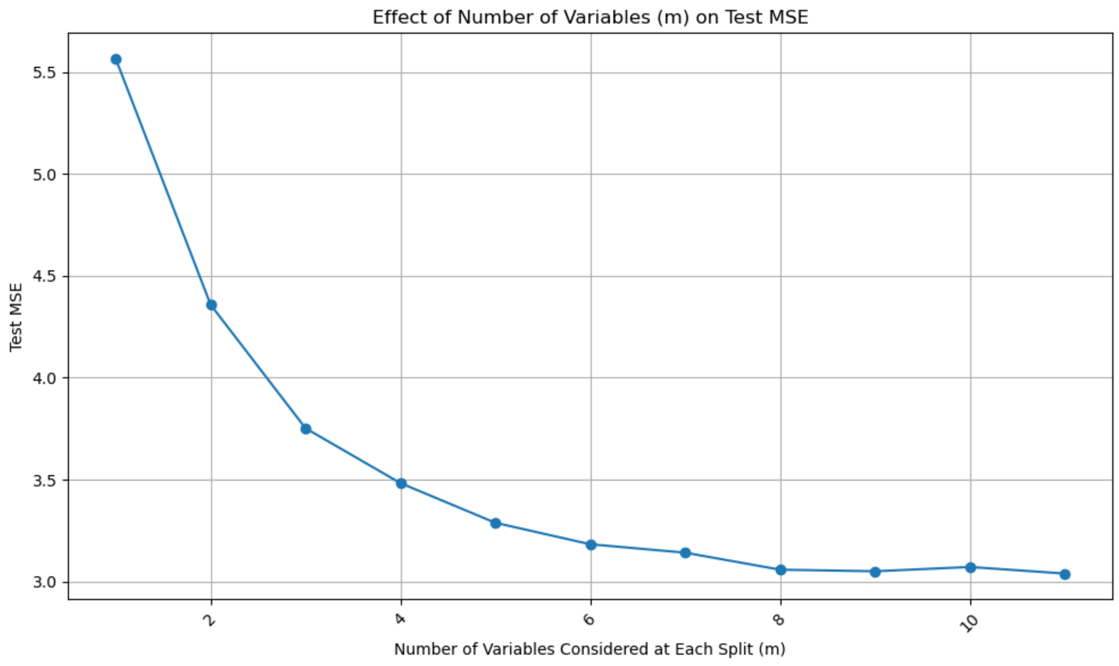
```
Test MSE with Bagging: 3.2485716159814695
```



We note that bagging brings down the MSE from 4.62 to 3.25.

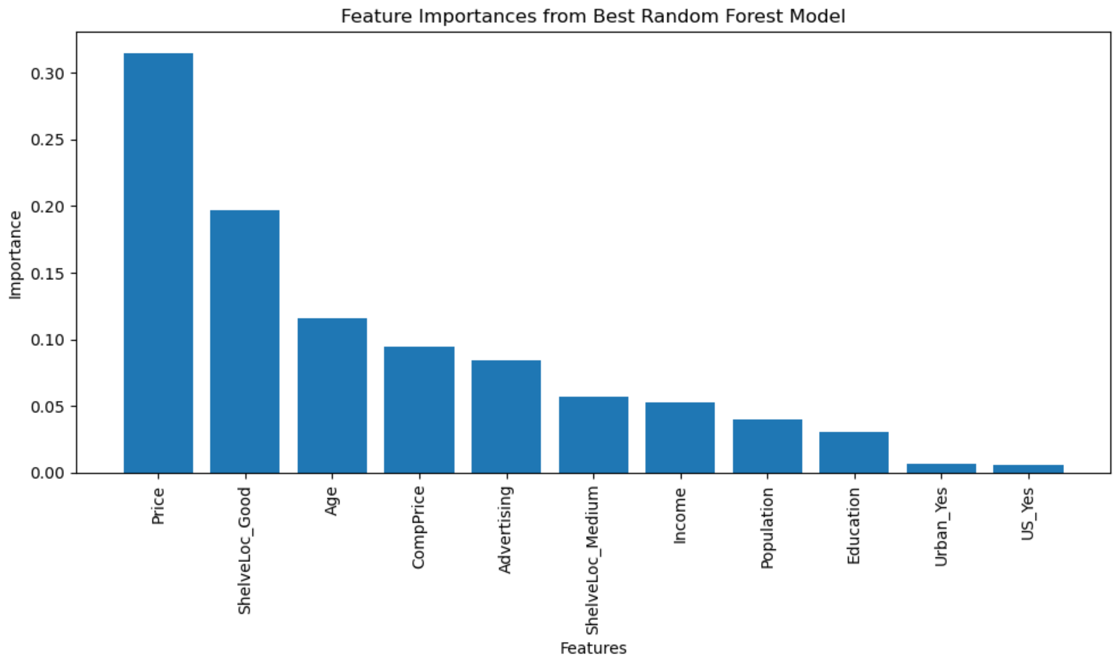
(e) Use random forests to analyze this data. What test MSE do you obtain? Use the feature_importance_ values to determine which variables are most important. Describe the effect of m, the number of variables considered at each split, on the error rate obtained.

[152]:
```
# Define a list of m values (number of variables considered at each split)
m_values_c8q8 = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10,11, None]

# Create empty lists to store test MSE and feature importances for each m value
test_mse_values_c8q8 = []
feature_importances_list_c8q8 = []

for m in m_values_c8q8:
    # Create the RandomForestRegressor with 1000 trees (adjust n_estimators as␣
 ↪needed)
    random_forest_reg_c8q8 = RandomForestRegressor(n_estimators=1000,␣
 ↪max_features=m, random_state=42)

    # Fit the RandomForestRegressor to the training data
    random_forest_reg_c8q8.fit(X_train_c8q8, y_train_c8q8)

    # Predict on the test set using the random forest model
    y_pred_rf_c8q8 = random_forest_reg_c8q8.predict(X_test_c8q8)
    # Calculate the mean squared error (MSE) on the test set
    test_mse_c8q8 = mean_squared_error(y_test_c8q8, y_pred_rf_c8q8)

    # Append the test MSE to the list
    test_mse_values_c8q8.append(test_mse_c8q8)

    # Get the feature importances from the Random Forest
    feature_importances_c8q8 = random_forest_reg_c8q8.feature_importances_
    feature_importances_list_c8q8.append(feature_importances_c8q8)

# Find the index of the best model (lowest test MSE)
best_model_index_c8q8 = np.argmin(test_mse_values_c8q8)

# Get the best model based on the lowest test MSE
best_random_forest_reg_c8q8 = RandomForestRegressor(n_estimators=1000,␣
 ↪max_features=m_values_c8q8[best_model_index_c8q8], random_state=42)
best_random_forest_reg_c8q8.fit(X_train_c8q8, y_train_c8q8)

# Plot the test MSE values for different m values
plt.figure(figsize=(10, 6))
plt.plot(m_values_c8q8, test_mse_values_c8q8, marker='o')
plt.xlabel("Number of Variables Considered at Each Split (m)")
plt.ylabel("Test MSE")
plt.title("Effect of Number of Variables (m) on Test MSE")
plt.xticks(rotation=45)
plt.grid(True)
plt.tight_layout()
plt.show()
```



We can clearly see from the above plot that up until m=8, as m goes up, the MSE goes down. However post m=8, any changes in the test MSE are negligible.

[153]:
```
# Plot the feature importances for the best model
best_feature_importances_c8q8 =␣
 ↪feature_importances_list_c8q8[best_model_index_c8q8]
feature_names_c8q8 = X_c8q8.columns
sorted_indices_c8q8 = np.argsort(best_feature_importances_c8q8)[::-1]
sorted_feature_importances_c8q8 =␣
 ↪best_feature_importances_c8q8[sorted_indices_c8q8]
sorted_feature_names_c8q8 = feature_names_c8q8[sorted_indices_c8q8]

plt.figure(figsize=(10, 6))
plt.bar(range(len(sorted_feature_importances_c8q8)),␣
 ↪sorted_feature_importances_c8q8)
plt.xticks(range(len(sorted_feature_importances_c8q8)),␣
 ↪sorted_feature_names_c8q8, rotation=90)
plt.xlabel("Features")
plt.ylabel("Importance")
plt.title("Feature Importances from Best Random Forest Model")
plt.tight_layout()
plt.show()
```



1.2 Question 11

This question uses the Caravan data set. (a) Create a training set consisting of the first 1,000 observations, and a test set consisting of the remaining observations.

```
[154]: caravan_c8q11=load_data('Caravan')
       caravan_c8q11.head(20)
```

```
[154]:     MOSTYPE  MAANTHUI  MGEMOMV  MGEMLEEF  MOSHOOFD  MGODRK  MGODPR  MGODOV  \
       0        33         1        3         2         8       0       5       1
       1        37         1        2         2         8       1       4       1
       2        37         1        2         2         8       0       4       2
       3         9         1        3         3         3       2       3       2
       4        40         1        4         2        10       1       4       1
       5        23         1        2         1         5       0       5       0
       6        39         2        3         2         9       2       2       0
       7        33         1        2         3         8       0       7       0
       8        33         1        2         4         8       0       1       3
       9        11         2        3         3         3       3       5       0
       10       10         1        4         3         3       1       4       1
       11        9         1        3         3         3       1       3       2
       12       33         1        2         3         8       1       4       1
       13       41         1        3         3        10       0       5       0
       14       23         1        1         2         5       0       6       1
       15       33         1        2         3         8       0       7       0
       16       38         1        2         3         9       0       6       0
       17       22         2        3         3         5       0       5       0
       18       13         1        4         2         3       2       4       0
       19       31         1        2         4         7       0       2       0

           MGODGE  MRELGE  …  APERSONG  AGEZONG  AWAOREG  ABRAND  AZEILPL  \
       0        3       7  …         0        0        0       1        0
       1        4       6  …         0        0        0       1        0
       2        4       3  …         0        0        0       1        0
       3        4       5  …         0        0        0       1        0
       4        4       7  …         0        0        0       1        0
       5        5       0  …         0        0        0       0        0
       6        5       7  …         0        0        0       0        0
       7        2       7  …         0        0        0       0        0
       8        6       6  …         0        0        0       0        0
       9        2       7  …         0        0        0       1        0
       10       4       7  …         0        0        0       0        0
       11       4       7  …         0        0        0       1        0
       12       4       6  …         0        0        0       0        0
       13       4       7  …         0        0        0       0        0
       14       2       1  …         0        0        0       1        0
       15       2       7  …         0        0        0       1        0
       16       3       7  …         0        0        0       0        0
```

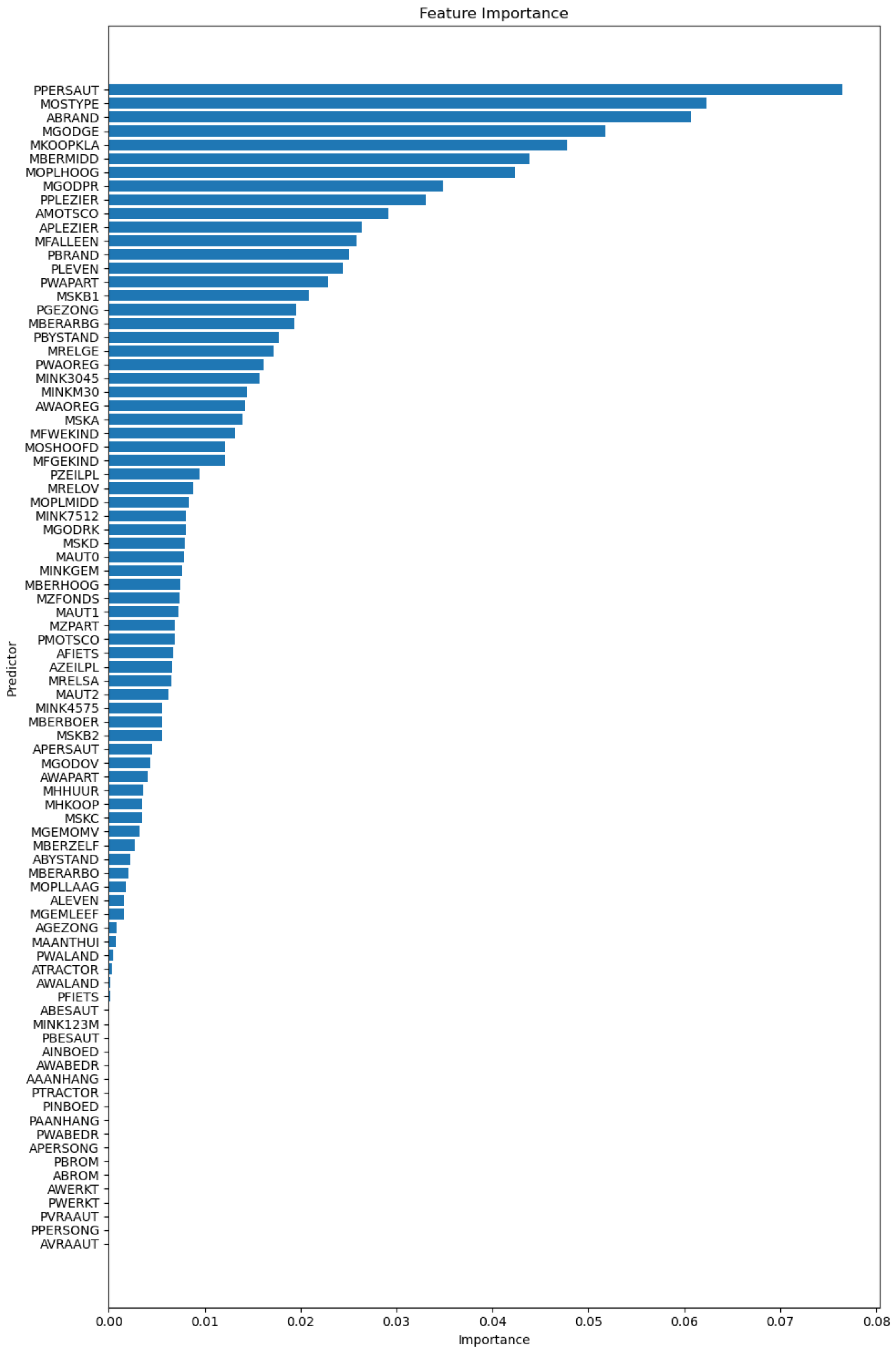
```
17        4       7  …         0        0        0       1        0
18        3       7  …         0        0        0       1        0
19        7       9  …         0        0        0       0        0

    APLEZIER  AFIETS  AINBOED  ABYSTAND  Purchase
0          0       0        0         0        No
1          0       0        0         0        No
2          0       0        0         0        No
3          0       0        0         0        No
4          0       0        0         0        No
5          0       0        0         0        No
6          0       0        0         0        No
7          0       0        0         0        No
8          0       0        0         0        No
9          0       0        0         0        No
10         0       0        0         0        No
11         0       0        0         0        No
12         0       0        0         0        No
13         0       0        0         0        No
14         0       0        0         0        No
15         0       0        0         0        No
16         0       0        0         0        No
17         0       0        0         0        No
18         0       0        0         0        No
19         0       0        0         0        No

[20 rows x 86 columns]
```

[155]:
```python
train_set_c8q11 = caravan_c8q11.iloc[:1000,:]
test_set_c8q11 = caravan_c8q11.iloc[1000:,:]

# Reset the index of both datasets
train_set_c8q11.reset_index(drop=True, inplace=True)
test_set_c8q11.reset_index(drop=True, inplace=True)
```

(b) Fit a boosting model to the training set with Purchase as the response and the other variables as predictors. Use 1,000 trees, and a shrinkage value of 0.01. Which predictors appear to be the most important?

[156]:
```python
# Split the train and test dataframes
X_train_c8q11 = train_set_c8q11.drop(columns=['Purchase'])
y_train_c8q11 = train_set_c8q11['Purchase']

X_test_c8q11 = test_set_c8q11.drop(columns=['Purchase'])
y_test_c8q11 = test_set_c8q11['Purchase']

# Create the GradientBoostingClassifier with the specified parameters
n_trees_c8q11 = 1000
```

```
learning_rate_c8q11 = 0.01
boosting_model_c8q11 = GradientBoostingClassifier(n_estimators=n_trees_c8q11,␣
 ↪learning_rate=learning_rate_c8q11)

# Fit the model to the training data
boosting_model_c8q11.fit(X_train_c8q11, y_train_c8q11)

# Get the feature importances
feature_importances_c8q11 = boosting_model_c8q11.feature_importances_

# Create a dataframe to show the importance of each predictor
importance_df_c8q11 = pd.DataFrame({'Predictor': X_train_c8q11.columns,␣
 ↪'Importance': feature_importances_c8q11})
# Sort the feature importances in descending order
importance_df_c8q11 = importance_df_c8q11.sort_values(by='Importance',␣
 ↪ascending=True)

# Plot the feature importance in ascending order as a horizontal bar plot
plt.figure(figsize=(10, 15))
plt.barh(importance_df_c8q11['Predictor'], importance_df_c8q11['Importance'])
plt.ylabel('Predictor')
plt.xlabel('Importance')
plt.title('Feature Importance')
plt.tight_layout()
plt.show()
```


Feature Importance
Predictor
Importance
PPERSAUT
MOSTYPE
ABRAND
MGODGE
MKOOPKLA
MBERMIDD
MOPLHOOG
MGODPR
PPLEZIER
AMOTSCO
APLEZIER
MFALLEEN
PBRAND
PLEVEN
PWAPART
MSKB1
PGEZONG
MBERARBG
PBYSTAND
MRELGE
PWAOREG
MINK3045
MINKM30
AWAOREG
MSKA
MFWEKIND
MOSHOOFD
MFGEKIND
PZEILPL
MRELOV
MOPLMIDD
MINK7512
MGODRK
MSKD
MAUT0
MINKGEM
MBERHOOG
MZFONDS
MAUT1
MZPART
PMOTSCO
AFIETS
AZEILPL
MRELSA
MAUT2
MINK4575
MBERBOER
MSKB2
APERSAUT
MGODOV
AWAPART
MHHUUR
MHKOOP
MSKC
MGEMOMV
MBERZELF
ABYSTAND
MBERARBO
MOPLLAAG
ALEVEN
MGEMLEEF
AGEZONG
MAANTHUI
PWALAND
ATRACTOR
AWALAND
PFIETS
ABESAUT
MINK123M
PBESAUT
AINBOED
AWABEDR
AAANHANG
PTRACTOR
PINBOED
PAANHANG
PWABEDR
APERSONG
PBROM
ABROM
AWERKT
PWERKT
PVRAAUT
PPERSONG
AVRAAUT
0.00
0.01
0.02
0.03
0.04
0.05
0.06
0.07
0.08

The above plot tells us which features are the most important.

(c) Use the boosting model to predict the response on the test data. Predict that a person will make a purchase if the estimated probability of purchase is greater than 20%. Form a confusion matrix. What fraction of the people predicted to make a purchase do in fact make one? How does this compare with the results obtained from applying KNN or logistic regression to this data set?

[157]:
```
# Predict the response on the test data
y_pred_boost_c8q11 = boosting_model_c8q11.predict_proba(X_test_c8q11)
pred_purchase_20_c8q11 = (y_pred_boost_c8q11[:, 1] > 0.2).astype(int)

# Convert predicted labels to strings
pred_purchase_20_str_c8q11 = np.where(pred_purchase_20_c8q11 == 1, 'Yes', 'No')

# Create a new DataFrame to store the predictions and add it to the test_set
test_set_pred_c8q11 = test_set_c8q11.copy()
test_set_pred_c8q11['Predicted_Purchase'] = pred_purchase_20_str_c8q11

# Form the confusion matrix
conf_matrix_c8q11 = confusion_matrix(y_test_c8q11, pred_purchase_20_str_c8q11)

# Extract the true positive (TP), false positive (FP), true negative (TN), and␣
↪false negative (FN) values
TP_c8q11 = conf_matrix_c8q11[1, 1]
FP_c8q11 = conf_matrix_c8q11[0, 1]
TN_c8q11 = conf_matrix_c8q11[0, 0]
FN_c8q11 = conf_matrix_c8q11[1, 0]

# # Calculate the true positive rate (sensitivity)
# true_positive_rate_c8q11 = TP_c8q11 / (TP_c8q11 + FN_c8q11)

# # Calculate the accuracy
# accuracy_c8q11 = accuracy_score(y_test_c8q11, pred_purchase_20_str_c8q11)

# # Print the accuracy
# print_green_bold("Accuracy:", accuracy_c8q11)

# Calculate the Precision
precision_c8q11 = TP_c8q11 / (TP_c8q11 + FP_c8q11)

# Print the confusion matrix
print_green_bold("Confusion Matrix:")
print_green_bold(conf_matrix_c8q11)
```

```
# Print the Precision
print_green_bold("The fraction of people predicted to make a purchase that do␣
 ↪in fact make one, i.e. Precision :",precision_c8q11)
```

```
Confusion Matrix:
[[4331  202]

 [ 252   37]]
The fraction of people predicted to make a purchase that do in fact make

one, i.e. Precision : 0.15481171548117154
```

[158]:
```
from sklearn.preprocessing import StandardScaler
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import confusion_matrix, precision_score

# Standardize the features
scaler = StandardScaler()
X_train_standardized_c8q11 = scaler.fit_transform(X_train_c8q11)
X_test_standardized_c8q11 = scaler.transform(X_test_c8q11)

# Initialize variables to store the best precision and corresponding n_neighbors
best_precision = 0
best_n_neighbors = 0

# Iterate over different n_neighbors values
for n_neighbors in range(1, 11):
    # Train KNN model
    knn_c8q11 = KNeighborsClassifier(n_neighbors=n_neighbors)
    knn_c8q11.fit(X_train_standardized_c8q11, y_train_c8q11)

    # Predict probabilities for the testing dataset
    knn_probs_c8q11 = knn_c8q11.predict_proba(X_test_standardized_c8q11)[:, 1]

    # Set the threshold to 0.2 (20%) for the KNN model
    knn_predictions_c8q11 = knn_probs_c8q11 > 0.2

    # Convert boolean predictions to string format to match true labels
    knn_predictions_c8q11 = ['Yes' if pred else 'No' for pred in␣
 ↪knn_predictions_c8q11]

    # Calculate precision for KNN model with pos_label='Yes'
    knn_precision_c8q11 = precision_score(y_test_c8q11, knn_predictions_c8q11,␣
 ↪pos_label='Yes')

    # Update the best_precision and best_n_neighbors if the current precision␣
 ↪is higher
    if knn_precision_c8q11 > best_precision:
        best_precision = knn_precision_c8q11
        best_n_neighbors = n_neighbors

# Print the optimal n_neighbors, precision, and confusion matrix for the best␣
↪model
print_green_bold("Optimal n_neighbors:", best_n_neighbors)
print_green_bold("Precision:", best_precision)

# Train the best model on the full training dataset and get the confusion matrix
best_knn_c8q11 = KNeighborsClassifier(n_neighbors=best_n_neighbors)
best_knn_c8q11.fit(X_train_standardized_c8q11, y_train_c8q11)
best_knn_probs_c8q11 = best_knn_c8q11.predict_proba(X_test_standardized_c8q11)[:
↪, 1]
best_knn_predictions_c8q11 = best_knn_probs_c8q11 > 0.2
best_knn_predictions_c8q11 = ['Yes' if pred else 'No' for pred in␣
↪best_knn_predictions_c8q11]
best_knn_cm_c8q11 = confusion_matrix(y_test_c8q11, best_knn_predictions_c8q11)

print_green_bold("Confusion Matrix for the best model:")
print_green_bold(best_knn_cm_c8q11)
```

```
Optimal n_neighbors: 10
Precision: 0.1784037558685446
Confusion Matrix for the best model:
[[4358  175]

 [ 251   38]]
```

[159]:
```
logreg_model_c8q11 = LogisticRegression()
logreg_model_c8q11.fit(X_train_standardized_c8q11, y_train_c8q11)



y_pred_prob_logreg_c8q11 = logreg_model_c8q11.
↪predict_proba(X_test_standardized_c8q11)[:, 1]

threshold = 0.2
y_pred_logreg_c8q11 = (y_pred_prob_logreg_c8q11 > threshold).astype(int)

# Convert predicted labels to strings 'Yes' or 'No'
pred_purchase_str_logreg_c8q11 = np.where(y_pred_logreg_c8q11 == 1, 'Yes', 'No')

# Form the confusion matrix
conf_matrix_logreg_c8q11= confusion_matrix(y_test_c8q11,␣
↪pred_purchase_str_logreg_c8q11)
print_green_bold("Confusion Matrix:")
print_green_bold(conf_matrix_logreg_c8q11)

# Calculate precision
```

```
precision_logreg_c8q11 = precision_score(y_test_c8q11,␣
    ↪pred_purchase_str_logreg_c8q11, pos_label='Yes')
print_green_bold("Precision:", precision_logreg_c8q11)
```

```
Confusion Matrix:
[[4291  242]

 [ 241   48]]
Precision: 0.16551724137931034
```

The question asks - What fraction of the people predicted to make a purchase do in fact make one? How does this compare with the results obtained from applying KNN or logistic regression to this data set? This fraction is titled as Precision and we have used it to evaluate three models - Gradient Boosting, kNN and Logistic regression. We found that all three models give us similar levels of precision with kNN having the highest value of 18%.

```
[ ]:
```