Problem set 2

1. (Total: 20 points) We want to use logistic regression to predict the probability of default using income, balance, and student on the Default data set. Our goal is to estimate the test error of this logistic regression model using the validation set approach. Do not forget to set a random seed before beginning your analysis.

```
def confusion_table(confusion_mtx):
    """Renders a nice confusion table with labels"""
    confusion_df = pd.DataFrame({'y_pred=0': np.append(confusion_mtx[:, 0], confusion_mtx.sum(axis=0)[0]),
                                 'y_pred=1': np.append(confusion_mtx[:, 1], confusion_mtx.sum(axis=0)[1]),
                                 'Total': np.append(confusion_mtx.sum(axis=1), ''),
                                 '': ['y=0', 'y=1', 'Total']}).set_index('')
    return confusion_df

def total_error_rate(confusion_matrix):
    """Derive total error rate from confusion matrix"""
    return 1 - np.trace(confusion_mtx) / np.sum(confusion_mtx)
```

(a) (5 points) Using the validation set approach, estimate the test error of a logistic regression model that uses income and balance to predict default. In order to do this, you must perform the following steps:

1. Split the sample set into a training set and a validation set.
2. Fit a multiple logistic regression model using only the training observations.
3. Obtain a prediction of default status for each individual in the validation set by computing the posterior probability of default for that individual, and classifying the individual to the default category if the posterior probability is greater than 0.5.
4. Compute the validation set error, which is the fraction of the observations in the validation set that are misclassified.

```
[127] default_df = pd.read_csv('https://raw.githubusercontent.com/jasonm/islr-exercises/master/Data/Default.csv', index_col='Unnamed: 0')
      default_df = default_df.reset_index().drop('index', axis=1)

      # Check for missing
      assert default_df.isna().sum().sum() == 0

      # Rationalise types
      default_df = pd.get_dummies(default_df, dtype=np.float64).drop(['default_No', 'student_No'], axis=1)

      display(default_df.head())
```

	balance	income	default_Yes	student_Yes
0	729.526495	44361.625074	0.0	0.0
1	817.180407	12106.134700	0.0	1.0
2	1073.549164	31767.138947	0.0	0.0
3	529.250605	35704.493935	0.0	0.0
4	785.655883	38463.495879	0.0	0.0

```
# Create index for 50% holdout set
np.random.seed(1)
train = np.random.rand(len(default_df)) < 0.5

response   = 'default_Yes'
predictors = ['income', 'balance']

X_train = np.array(default_df[train][predictors])
X_test  = np.array(default_df[~train][predictors])
y_train = np.array(default_df[train][response])
y_test  = np.array(default_df[~train][response])

# Logistic regression
logit       = LogisticRegression()
model_logit = logit.fit(X_train, y_train)

# Predict
y_pred = model_logit.predict(X_test)

# Analysis
confusion_mtx = confusion_matrix(y_test, y_pred)
display(confusion_table(confusion_mtx))

total_error_rate_pct = np.around(total_error_rate(confusion_mtx) * 100, 4)
```

	y_pred=0	y_pred=1	Total
y=0	4846	0	4846
y=1	164	0	164
Total	5010	0	

```
total_error_rate: 3.2735%
```

(b) Repeat the process in (a) three times, using three different splits of the observations into a training set and a validation set. Describe your findings and comment on the results obtained.

```
from IPython.display import display, HTML
for s in range(1,4):
    display(HTML('<h3>Random seed = {}</h3>'.format(s)))
    # Create index for 50% holdout set
    np.random.seed(s)
    train = np.random.rand(len(default_df)) < 0.5

    response   = 'default_Yes'
    predictors = ['income', 'balance']

    X_train = np.array(default_df[train][predictors])
    X_test  = np.array(default_df[~train][predictors])
    y_train = np.array(default_df[train][response])
    y_test  = np.array(default_df[~train][response])

    # Logistic regression
    logit       = LogisticRegression()
    model_logit = logit.fit(X_train, y_train)

    # Predict
    y_pred = model_logit.predict(X_test)
```

```
    # Analysis
    confusion_mtx = confusion_matrix(y_test, y_pred)
    display(confusion_table(confusion_mtx))

    total_error_rate_pct = np.around(total_error_rate(confusion_mtx) * 100, 4)
    print('total_error_rate: {}%'.format(total_error_rate_pct))
```

Random seed = 1

	y_pred=0	y_pred=1	Total
y=0	4846	0	4846
y=1	164	0	164
Total	5010	0	

total_error_rate: 3.2735%

Random seed = 2

	y_pred=0	y_pred=1	Total
y=0	4691	1	4692
y=1	176	0	176
Total	4867	1	

total_error_rate: 3.636%

Random seed = 3

	y_pred=0	y_pred=1	Total
y=0	4773	1	4774
y=1	163	0	163
Total	4936	1	

total_error_rate: 3.3219%

(c) Now consider a logistic regression model that predicts the probability of default using income, balance, and a dummy variable for student. Estimate the test error for this model using the validation set approach. Comment on whether or not including a dummy variable for student leads to a reduction in the test error rate.

```
for s in range(1,4):
    display(HTML('<h3>Random seed = {}</h3>'.format(s)))
    # Create index for 50% holdout set
    np.random.seed(s)
    train = np.random.rand(len(default_df)) < 0.5

    response   = 'default_Yes'
    predictors = ['income', 'balance', 'student_Yes']

    X_train = np.array(default_df[train][predictors])
    X_test  = np.array(default_df[~train][predictors])
    y_train = np.array(default_df[train][response])
    y_test  = np.array(default_df[~train][response])

    # Logistic regression
    logit       = LogisticRegression()
    model_logit = logit.fit(X_train, y_train)

    # Predict
    y_pred = model_logit.predict(X_test)

    # Analysis
    confusion_mtx = confusion_matrix(y_test, y_pred)
    display(confusion_table(confusion_mtx))

    total_error_rate_pct = np.around(total_error_rate(confusion_mtx) * 100, 4)
    print('total_error_rate: {}%'.format(total_error_rate_pct))
```

Random seed = 1

	y_pred=0	y_pred=1	Total
y=0	4846	0	4846
y=1	164	0	164
Total	5010	0	

```
total_error_rate: 3.2735%
```

Random seed = 2

	y_pred=0	y_pred=1	Total
y=0	4688	4	4692
y=1	172	4	176
Total	4860	8	

```
total_error_rate: 3.6154%
```

Random seed = 3

	y_pred=0	y_pred=1	Total
y=0	4773	1	4774
y=1	163	0	163
Total	4936	1	

```
total_error_rate: 3.3219%
```

It is difficult to discern if the student predictor has improved the model because of the variation in results.

2. Suppose we collect data for a group of bank customers with variables
   - default A variable with levels No and Yes indicating whether the customer defaulted on their debt
   - student A variable with levels No and Yes indicating whether the customer is a student
   - balance The average balance that the customer has remaining on their credit card after making their monthly payment
   - income of a customer

   We fit a logistic regression and produce estimated coefficient, $\hat{\beta}0 = -15.05$, $\hat{\beta}$studentYes = -0.5149, $\hat{\beta}$balance = 0.003738, $\hat{\beta}$income = −0.00000791.

   (a) Estimate the probability that a student with a balance of $3,000 and income $70,000 does default on a loan.

$$p(x) = \frac{1}{e^{-(\beta_0+\beta_1x_1+\beta_2x_2+\beta_3x_3)}+1}$$

$$\beta_0+\beta_1x_1+\beta_2x_2+\beta_3x_3 = -15.05 - 0.5149 + 0.003738 \times 3000 - 0.00000791 \times 70{,}000$$

$$= -4.9046$$

$$p(x) = 0.00741$$

(b) Estimate the probability that borrower a balance of $3,000 and income $70,000 who is not a student does default on a loan.

$$p(x) = \frac{1}{e^{-(\beta_0+\beta_1x_1+\beta_2x_2+\beta_3x_3)}+1}$$

$$\beta_0+\beta_1x_1+\beta_2x_2+\beta_3x_3 = -15.05 + 0.003738 \times 3000 - 0.00000791 \times 70{,}000$$

$$= -4.3897$$

$$p(x) = 0.0124$$

(c)How much income would the student in part (a) need to make to have a 90% chance of getting approved for a loan?

$a = -15.05 - 0.5149 + 0.003738 * 3000 - 0.00000791 * x.$

$0.9 = \frac{1}{e^{-a}+1}$

$e^{-a} = \frac{1}{9}$

$a = 0.9542.$

$x = -\frac{5.3051}{0.00000791}$

$x = -670682.680$

(d) How much income would the borrower in part (b) need to make to have a 90% chance of getting approved for a loan?

$a = -15.05 + 0.003738 * 3000 - 0.00000791 * 2$

$e^{-a} = \frac{1}{9}$

$a = 0.9542$

$x = -\frac{4.7902}{0.00000791}$

$x = 605587.863$

3. This problem involves the Boston data set, which can be found in the file Boston.csv. This data set contains the following columns:
crim per capita crime rate by town.
zn proportion of residential land zoned for lots over 25,000 sq.ft.
indus proportion of non-retail business acres per town.
chas Charles River dummy variable (= 1 if tract bounds river; 0 otherwise).
nox nitrogen oxides concentration (parts per 10 million).
rm average number of rooms per dwelling.
age proportion of owner-occupied units built prior to 1940.
dis weighted mean of distances to five Boston employment centres.
rad index of accessibility to radial highways.
tax full-value property-tax rate per $10,000.
ptratio pupil-teacher ratio by town.
black 1000(Bk −0.63)2 where Bk is the proportion of blacks by town.
lstat lower status of the population (percent).
medv median value of owner-occupied homes in $1,000s.
We want to predict whether a given suburb has a crime rate above or below the median using the other variables in this data set.

(a) For each predictor, fit a simple logistic regression model to predict the response. In which of the models is there a statistically significant association between the predictor and the response (set $\alpha = 0.05$)

```
[ ] file_url ='https://raw.githubusercontent.com/Yeshaswini18/DataScience/main/Assignment1/Boston.csv'
    df = pd.read_csv(file_url)
```

```
    df['crim'].median()

    0.25651
```

```
[ ] df['crim'].values[df['crim'].values > 0.25651 ] = 1
    df['crim'].values[df['crim'].values < 0.25651 ] = 0
```

```
[ ] X = df[['zn']]
    y = df['crim']
```

```
[ ] X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.2,random_state=0)
```

```
    from sklearn.linear_model import LogisticRegression
    logreg = LogisticRegression()
    logreg.fit(X_train,y_train)
    y_pred=logreg.predict(X_test)
```

```
[ ] import statsmodels.api as sm
    X2=sm.add_constant(X_train)
    est=sm.OLS(y_train,X2)
    est2=est.fit()
    est2.summary()
```

OLS Regression Results

Dep. Variable:	crim	**R-squared:**	0.198
Model:	OLS	**Adj. R-squared:**	0.196
Method:	Least Squares	**F-statistic:**	99.31
Date:	Sat, 20 Nov 2021	**Prob (F-statistic):**	4.73e-21
Time:	20:39:43	**Log-Likelihood:**	-248.62
No. Observations:	404	**AIC:**	501.2
Df Residuals:	402	**BIC:**	509.2
Df Model:	1		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
const	0.6112	0.025	24.483	0.000	0.562	0.660
zn	-0.0094	0.001	-9.965	0.000	-0.011	-0.008

Omnibus:	2424.469	**Durbin-Watson:**	2.133
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	55.817
Skew:	-0.358	**Prob(JB):**	7.58e-13
Kurtosis:	1.325	**Cond. No.**	29.5

```
import statsmodels.api as sm
from sklearn.linear_model import LogisticRegression

X = df[['indus']]
y = df['crim']
X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.2,random_state=0)
logreg = LogisticRegression()
logreg.fit(X_train,y_train)
y_pred=logreg.predict(X_test)
X2=sm.add_constant(X_train)
est=sm.OLS(y_train,X2)
est2=est.fit()
est2.summary()
```

OLS Regression Results

Dep. Variable:	crim	**R-squared:**	0.334
Model:	OLS	**Adj. R-squared:**	0.332
Method:	Least Squares	**F-statistic:**	201.4
Date:	Sat, 20 Nov 2021	**Prob (F-statistic):**	2.42e-37
Time:	20:39:43	**Log-Likelihood:**	-211.18
No. Observations:	404	**AIC:**	426.4
Df Residuals:	402	**BIC:**	434.4
Df Model:	1		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
const	0.0399	0.038	1.044	0.297	-0.035	0.115
indus	0.0414	0.003	14.191	0.000	0.036	0.047

Omnibus:	3.968	**Durbin-Watson:**	2.130
Prob(Omnibus):	0.138	**Jarque-Bera (JB):**	3.749
Skew:	-0.194	**Prob(JB):**	0.153
Kurtosis:	3.270	**Cond. No.**	24.8

```
import statsmodels.api as sm
from sklearn.linear_model import LogisticRegression

X = df[['chas']]
y = df['crim']
X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.2,random_state=0)
logreg = LogisticRegression()
logreg.fit(X_train,y_train)
y_pred=logreg.predict(X_test)
X2=sm.add_constant(X_train)
est=sm.OLS(y_train,X2)
est2=est.fit()
est2.summary()
```

OLS Regression Results

Dep. Variable:	crim	**R-squared:**	0.003
Model:	OLS	**Adj. R-squared:**	0.001
Method:	Least Squares	**F-statistic:**	1.379
Date:	Sat, 20 Nov 2021	**Prob (F-statistic):**	0.241
Time:	20:39:43	**Log-Likelihood:**	-292.53
No. Observations:	404	**AIC:**	589.1
Df Residuals:	402	**BIC:**	597.1
Df Model:	1		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
const	0.4920	0.026	19.067	0.000	0.441	0.543
chas	0.1151	0.098	1.174	0.241	-0.078	0.308

Omnibus:	1762.735	**Durbin-Watson:**	2.087
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	66.440
Skew:	0.001	**Prob(JB):**	3.74e-15
Kurtosis:	1.013	**Cond. No.**	3.96

```
import statsmodels.api as sm
from sklearn.linear_model import LogisticRegression

X = df[['nox']]
y = df['crim']
X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.2,random_state=0)
logreg = LogisticRegression()
logreg.fit(X_train,y_train)
y_pred=logreg.predict(X_test)
X2=sm.add_constant(X_train)
est=sm.OLS(y_train,X2)
est2=est.fit()
est2.summary()
```

OLS Regression Results

Dep. Variable:	crim	**R-squared:**	0.536
Model:	OLS	**Adj. R-squared:**	0.535
Method:	Least Squares	**F-statistic:**	464.0
Date:	Sat, 20 Nov 2021	**Prob (F-statistic):**	5.59e-69
Time:	20:39:43	**Log-Likelihood:**	-138.21
No. Observations:	404	**AIC:**	280.4
Df Residuals:	402	**BIC:**	288.4
Df Model:	1		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
const	-1.2682	0.084	-15.128	0.000	-1.433	-1.103
nox	3.1809	0.148	21.540	0.000	2.891	3.471

Omnibus:	16.495	**Durbin-Watson:**	2.063
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	11.480
Skew:	0.293	**Prob(JB):**	0.00322
Kurtosis:	2.417	**Cond. No.**	11.4

```
import statsmodels.api as sm
from sklearn.linear_model import LogisticRegression

X = df[['rm']]
y = df['crim']
X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.2,random_state=0)
logreg = LogisticRegression()
logreg.fit(X_train,y_train)
y_pred=logreg.predict(X_test)
X2=sm.add_constant(X_train)
est=sm.OLS(y_train,X2)
est2=est.fit()
est2.summary()
```

OLS Regression Results

Dep. Variable:	crim	**R-squared:**	0.029
Model:	OLS	**Adj. R-squared:**	0.026
Method:	Least Squares	**F-statistic:**	11.84
Date:	Sat, 20 Nov 2021	**Prob (F-statistic):**	0.000639
Time:	20:39:43	**Log-Likelihood:**	-287.35
No. Observations:	404	**AIC:**	578.7
Df Residuals:	402	**BIC:**	586.7
Df Model:	1		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
const	1.2683	0.225	5.647	0.000	0.827	1.710
rm	-0.1219	0.035	-3.441	0.001	-0.192	-0.052

Omnibus:	1904.188	**Durbin-Watson:**	2.083
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	59.719
Skew:	0.027	**Prob(JB):**	1.08e-13
Kurtosis:	1.117	**Cond. No.**	59.4

```
import statsmodels.api as sm
from sklearn.linear_model import LogisticRegression

X = df[['age']]
y = df['crim']
X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.2,random_state=0)
logreg = LogisticRegression()
logreg.fit(X_train,y_train)
y_pred=logreg.predict(X_test)
X2=sm.add_constant(X_train)
est=sm.OLS(y_train,X2)
est2=est.fit()
est2.summary()
```

OLS Regression Results

Dep. Variable:	crim	**R-squared:**	0.405
Model:	OLS	**Adj. R-squared:**	0.404
Method:	Least Squares	**F-statistic:**	273.7
Date:	Sat, 20 Nov 2021	**Prob (F-statistic):**	2.93e-47
Time:	20:39:43	**Log-Likelihood:**	-188.33
No. Observations:	404	**AIC:**	380.7
Df Residuals:	402	**BIC:**	388.7
Df Model:	1		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
const	-0.2825	0.051	-5.532	0.000	-0.383	-0.182
age	0.0113	0.001	16.543	0.000	0.010	0.013

Omnibus:	8.209	**Durbin-Watson:**	2.048
Prob(Omnibus):	0.016	**Jarque-Bera (JB):**	8.446
Skew:	-0.340	**Prob(JB):**	0.0147
Kurtosis:	2.805	**Cond. No.**	198.

```
import statsmodels.api as sm
from sklearn.linear_model import LogisticRegression

X = df[['dis']]
y = df['crim']
X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.2,random_state=0)
logreg = LogisticRegression()
logreg.fit(X_train,y_train)
y_pred=logreg.predict(X_test)
X2=sm.add_constant(X_train)
est=sm.OLS(y_train,X2)
est2=est.fit()
est2.summary()
```

OLS Regression Results

Dep. Variable:	crim	**R-squared:**	0.387
Model:	OLS	**Adj. R-squared:**	0.386
Method:	Least Squares	**F-statistic:**	254.0
Date:	Sat, 20 Nov 2021	**Prob (F-statistic):**	1.13e-44
Time:	20:39:43	**Log-Likelihood:**	-194.29
No. Observations:	404	**AIC:**	392.6
Df Residuals:	402	**BIC:**	400.6
Df Model:	1		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
const	1.0645	0.040	26.322	0.000	0.985	1.144
dis	-0.1497	0.009	-15.938	0.000	-0.168	-0.131

Omnibus:	63.034	**Durbin-Watson:**	2.162
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	28.169
Skew:	-0.463	**Prob(JB):**	7.64e-07
Kurtosis:	2.097	**Cond. No.**	9.29

```
X = df[['rad']]
y = df['crim']
X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.2,random_state=0)
logreg = LogisticRegression()
logreg.fit(X_train,y_train)
y_pred=logreg.predict(X_test)
X2=sm.add_constant(X_train)
est=sm.OLS(y_train,X2)
est2=est.fit()
est2.summary()
```

OLS Regression Results

Dep. Variable:	crim	**R-squared:**	0.368
Model:	OLS	**Adj. R-squared:**	0.367
Method:	Least Squares	**F-statistic:**	234.4
Date:	Sat, 20 Nov 2021	**Prob (F-statistic):**	5.16e-42
Time:	20:39:43	**Log-Likelihood:**	-200.42
No. Observations:	404	**AIC:**	404.8
Df Residuals:	402	**BIC:**	412.8
Df Model:	1		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
const	0.1701	0.029	5.811	0.000	0.113	0.228
rad	0.0351	0.002	15.311	0.000	0.031	0.040

Omnibus:	82.420	**Durbin-Watson:**	2.055
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	61.173
Skew:	0.846	**Prob(JB):**	5.21e-14
Kurtosis:	2.124	**Cond. No.**	18.9

```
X = df[['tax']]
y = df['crim']
X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.2,random_state=0)
logreg = LogisticRegression()
logreg.fit(X_train,y_train)
y_pred=logreg.predict(X_test)
X2=sm.add_constant(X_train)
est=sm.OLS(y_train,X2)
est2=est.fit()
est2.summary()
```

OLS Regression Results

Dep. Variable:	crim	**R-squared:**	0.362
Model:	OLS	**Adj. R-squared:**	0.360
Method:	Least Squares	**F-statistic:**	228.0
Date:	Sat, 20 Nov 2021	**Prob (F-statistic):**	4.00e-41
Time:	20:39:43	**Log-Likelihood:**	-202.47
No. Observations:	404	**AIC:**	408.9
Df Residuals:	402	**BIC:**	416.9
Df Model:	1		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
const	-0.2101	0.051	-4.113	0.000	-0.310	-0.110
tax	0.0018	0.000	15.099	0.000	0.002	0.002

Omnibus:	18.830	**Durbin-Watson:**	2.066
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	17.242
Skew:	0.445	**Prob(JB):**	0.000180
Kurtosis:	2.519	**Cond. No.**	1.12e+03

```
X = df[['ptratio']]
y = df['crim']
X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.2,random_state=0)
logreg = LogisticRegression()
logreg.fit(X_train,y_train)
y_pred=logreg.predict(X_test)
X2=sm.add_constant(X_train)
est=sm.OLS(y_train,X2)
est2=est.fit()
est2.summary()
```

OLS Regression Results

Dep. Variable:	crim	**R-squared:**	0.056
Model:	OLS	**Adj. R-squared:**	0.054
Method:	Least Squares	**F-statistic:**	23.81
Date:	Sat, 20 Nov 2021	**Prob (F-statistic):**	1.54e-06
Time:	20:39:43	**Log-Likelihood:**	-281.60
No. Observations:	404	**AIC:**	567.2
Df Residuals:	402	**BIC:**	575.2
Df Model:	1		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
const	-0.4957	0.206	-2.412	0.016	-0.900	-0.092
ptratio	0.0539	0.011	4.879	0.000	0.032	0.076

Omnibus:	2215.859	**Durbin-Watson:**	2.068
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	50.938
Skew:	0.063	**Prob(JB):**	8.69e-12
Kurtosis:	1.265	**Cond. No.**	158.

```
X = df[['black']]
y = df['crim']
X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.2,random_state=0)
logreg = LogisticRegression()
logreg.fit(X_train,y_train)
y_pred=logreg.predict(X_test)
X2=sm.add_constant(X_train)
est=sm.OLS(y_train,X2)
est2=est.fit()
est2.summary()
```

OLS Regression Results

Dep. Variable:	crim	**R-squared:**	0.123
Model:	OLS	**Adj. R-squared:**	0.120
Method:	Least Squares	**F-statistic:**	56.18
Date:	Sat, 20 Nov 2021	**Prob (F-statistic):**	4.26e-13
Time:	20:39:43	**Log-Likelihood:**	-266.80
No. Observations:	404	**AIC:**	537.6
Df Residuals:	402	**BIC:**	545.6
Df Model:	1		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
const	1.2093	0.097	12.406	0.000	1.018	1.401
black	-0.0020	0.000	-7.495	0.000	-0.002	-0.001

Omnibus:	1956.541	**Durbin-Watson:**	2.134
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	61.287
Skew:	0.227	**Prob(JB):**	4.92e-14
Kurtosis:	1.147	**Cond. No.**	1.54e+03

```
X = df[['lstat']]
y = df['crim']
X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.2,random_state=0)
logreg = LogisticRegression()
logreg.fit(X_train,y_train)
y_pred=logreg.predict(X_test)
X2=sm.add_constant(X_train)
est=sm.OLS(y_train,X2)
est2=est.fit()
est2.summary()
```

OLS Regression Results

Dep. Variable:	crim	**R-squared:**	0.221
Model:	OLS	**Adj. R-squared:**	0.219
Method:	Least Squares	**F-statistic:**	113.8
Date:	Sat, 20 Nov 2021	**Prob (F-statistic):**	1.44e-23
Time:	20:39:43	**Log-Likelihood:**	-242.85
No. Observations:	404	**AIC:**	489.7
Df Residuals:	402	**BIC:**	497.7
Df Model:	1		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
const	0.0906	0.044	2.048	0.041	0.004	0.178
lstat	0.0322	0.003	10.669	0.000	0.026	0.038

Omnibus:	168.543	**Durbin-Watson:**	2.025
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	23.380
Skew:	0.145	**Prob(JB):**	8.38e-06
Kurtosis:	1.858	**Cond. No.**	29.5

```
X = df[['medv']]
y = df['crim']
X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.2,random_state=0)
logreg = LogisticRegression()
logreg.fit(X_train,y_train)
y_pred=logreg.predict(X_test)
X2=sm.add_constant(X_train)
est=sm.OLS(y_train,X2)
est2=est.fit()
est2.summary()
```

OLS Regression Results

Dep. Variable:	crim	**R-squared:**	0.099
Model:	OLS	**Adj. R-squared:**	0.097
Method:	Least Squares	**F-statistic:**	44.34
Date:	Sat, 20 Nov 2021	**Prob (F-statistic):**	9.07e-11
Time:	20:39:43	**Log-Likelihood:**	-272.08
No. Observations:	404	**AIC:**	548.2
Df Residuals:	402	**BIC:**	556.2
Df Model:	1		
Covariance Type:	nonrobust		

	coef	**std err**	**t**	**P>|t|**	**[0.025**	**0.975]**
const	0.8862	0.063	14.148	0.000	0.763	1.009
medv	-0.0171	0.003	-6.659	0.000	-0.022	-0.012

Omnibus:	3724.320	**Durbin-Watson:**	2.072
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	39.383
Skew:	0.116	**Prob(JB):**	2.81e-09
Kurtosis:	1.488	**Cond. No.**	64.7

To find which predictors are significant, we have to test H0:β1=0. All predictors have a p-value less than 0.05 except “chas”, so we may conclude that there is a statistically significant association between each predictor and the response except for the “chas” predictor in this particular case.

(b) Fit a multiple logistic model to predict the response using all of the predictors. Describe your results.

(i) Do any of the predictors appear to be statistically significant (set α = 0.05)? If so, which ones?

```
X = df[['zn', 'indus', 'chas', 'nox', 'rm', 'age', 'dis', 'rad', 'tax', 'ptratio', 'black', 'lstat', 'medv']]
y = df['crim']
X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.2,random_state=0)
logreg = LogisticRegression()
logreg.fit(X_train,y_train)
y_pred=logreg.predict(X_test)
X2=sm.add_constant(X_train)
est=sm.OLS(y_train,X2)
est2=est.fit()
est2.summary()
```

Covariance Type: nonrobust

	coef	std err	t	P>|t|	[0.025	0.975]
const	-1.3262	0.418	-3.176	0.002	-2.147	-0.505
zn	-0.0011	0.001	-1.074	0.284	-0.003	0.001
indus	-0.0010	0.005	-0.226	0.822	-0.010	0.008
chas	0.0398	0.065	0.613	0.541	-0.088	0.168
nox	1.9278	0.306	6.301	0.000	1.326	2.529
rm	0.0213	0.035	0.604	0.546	-0.048	0.091
age	0.0036	0.001	3.536	0.000	0.002	0.006
dis	0.0023	0.016	0.145	0.885	-0.029	0.034
rad	0.0153	0.005	3.153	0.002	0.006	0.025
tax	-0.0001	0.000	-0.455	0.650	-0.001	0.000
ptratio	0.0106	0.010	1.008	0.314	-0.010	0.031
black	-0.0003	0.000	-1.393	0.164	-0.001	0.000
lstat	0.0033	0.004	0.778	0.437	-0.005	0.012
medv	0.0071	0.004	1.990	0.047	8.46e-05	0.014

Omnibus:	17.632	Durbin-Watson:	2.058
Prob(Omnibus):	0.000	Jarque-Bera (JB):	17.572
Skew:	0.471	Prob(JB):	0.000153
Kurtosis:	2.604	Cond. No.	1.63e+04

We may reject the null hypothesis for "nox", "age", “rad”, “black” and “medv”.

(ii) Predict whether a given suburb has a crime rate above or below the median for a suburb with:
zn=18 indus=2.97 chas= 0 nox= 0.4 rm=6.575 age=63 dis=4.8
rad=3 tax=238 ptratio=15.3 black=376.7 lstat=8.23 medv=45

Ans: The crime rate is below median for the suburb with the above values

(iii)Compute the confusion matrix, accuracy, balanced accuracy, FP, and FP rates for Threshold=0.5. Explain what the confusion matrix is telling you about the types of mistakes made by logistic regression.

```
[ ] from sklearn import metrics
    cnf_matrix = metrics.confusion_matrix(y_test, y_pred)
    cnf_matrix
```

```
array([[43,  8],
       [ 9, 42]])
```

```
[ ] from sklearn.metrics import balanced_accuracy_score
    balanced_accuracy_score(y_test, y_pred)
```

```
0.8333333333333333
```

```
[ ] from sklearn.metrics import accuracy_score
    accuracy_score(y_test, y_pred)
```

```
0.8333333333333334
```

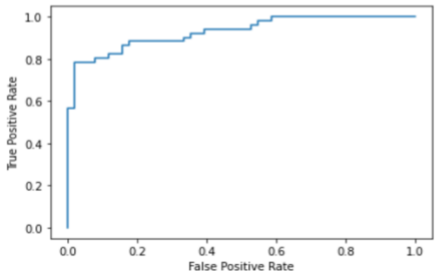
```
[ ] FP = cnf_matrix.sum(axis=0) - np.diag(cnf_matrix)
    FN = cnf_matrix.sum(axis=1) - np.diag(cnf_matrix)
    TP = np.diag(cnf_matrix)
    TN = cnf_matrix.sum() - (FP + FN + TP)
    FP = FP.astype(float)
    FN = FN.astype(float)
    TP = TP.astype(float)
    TN = TN.astype(float)

    # Sensitivity, hit rate, recall, or true positive rate
    TPR = TP/(TP+FN)
    # Specificity or true negative rate
    TNR = TN/(TN+FP)
    TPR
    TNR
```

```
array([0.82352941, 0.84313725])
```

Confusion matrix says that the misclassification rate of the logistic regression is high.

(iv) Draw the ROC curve. What is the AUC?

```
[ ] #define metrics
    y_pred_proba = logreg.predict_proba(X_test)[::,1]
    fpr, tpr, _ = metrics.roc_curve(y_test,  y_pred_proba)
    auc = metrics.roc_auc_score(y_test, y_pred_proba)
    #create ROC curve
    plt.plot(fpr,tpr)
    plt.ylabel('True Positive Rate')
    plt.xlabel('False Positive Rate')
    plt.show()
    auc
```



auc: 0.9284890426758939

(v) Determine the threshold for which (TP rate + (1-FP rate)) is maximal. Then, compute the confusion matrix, accuracy, balanced accuracy, FP, and FP rates for that threshold.
Ans: Threshold is 0.8

```
threshold = 0.8
y_pred = (logreg.predict_proba(X_test)[:, 1] > threshold).astype('float')
confusion_matrix(y_test, y_pred)
from sklearn.metrics import balanced_accuracy_score
ba = balanced_accuracy_score(y_test, y_pred)
from sklearn.metrics import accuracy_score
a = accuracy_score(y_test, y_pred)
FP = cnf_matrix.sum(axis=0) - np.diag(cnf_matrix)
FN = cnf_matrix.sum(axis=1) - np.diag(cnf_matrix)
TP = np.diag(cnf_matrix)
TN = cnf_matrix.sum() - (FP + FN + TP)
FP = FP.astype(float)
FN = FN.astype(float)
TP = TP.astype(float)
TN = TN.astype(float)

# Sensitivity, hit rate, recall, or true positive rate
TPR = TP/(TP+FN)
# Specificity or true negative rate
TNR = TN/(TN+FP)
```

```
[0.84313725 0.82352941]
[0.82352941 0.84313725]
Balanaced Accuracy: 0.8823529411764706
Accuracy: 0.8823529411764706
```

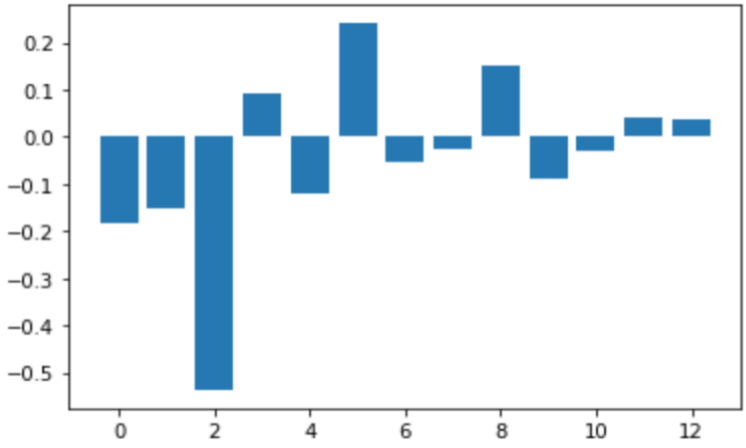
(vi) Determine the threshold for which the ROC curve has the minimum distance to the upper left corner (where TP rate=1 and FP rate=0). Note that this distance is√(1 −TP rate)2 + (FP rate)2. Then, compute the confusion matrix, accuracy, balanced accuracy, FP, and FP rates for that threshold.

```
threshold = 0.8
y_pred = (logreg.predict_proba(X_test)[:, 1] > threshold).astype('float')
confusion_matrix(y_test, y_pred)
from sklearn.metrics import balanced_accuracy_score
ba = balanced_accuracy_score(y_test, y_pred)
from sklearn.metrics import accuracy_score
a = accuracy_score(y_test, y_pred)
FP = cnf_matrix.sum(axis=0) - np.diag(cnf_matrix)
FN = cnf_matrix.sum(axis=1) - np.diag(cnf_matrix)
TP = np.diag(cnf_matrix)
TN = cnf_matrix.sum() - (FP + FN + TP)
FP = FP.astype(float)
FN = FN.astype(float)
TP = TP.astype(float)
TN = TN.astype(float)

# Sensitivity, hit rate, recall, or true positive rate
TPR = TP/(TP+FN)
# Specificity or true negative rate
TNR = TN/(TN+FP)
```

```
[0.84313725 0.82352941]
[0.82352941 0.84313725]
Balanaced Accuracy: 0.8823529411764706
Accuracy: 0.8823529411764706
```

(c) Which predictors matter most for predicting whether a given suburb has a crime rate above or below the median? (Find the first and the second most important variables)
Ans: “age” and “nox” are the two important predictors

```
from sklearn.datasets import make_classification
from sklearn.linear_model import LogisticRegression
from matplotlib import pyplot
# define dataset
X, y = make_classification(n_samples=1000, n_features=13, n_informative=8, n_redundant=5, random_state=1)
# define the model
model = LogisticRegression()
# fit the model
model.fit(X, y)
# get importance
importance = model.coef_[0]
# summarize feature importance
for i,v in enumerate(importance):
  print('Feature: %0d, Score: %.5f' % (i,v))
# plot feature importance
pyplot.bar([x for x in range(len(importance))], importance)
pyplot.show()
```



(d) Fit a multiple logistic model to predict the response using the first two most important variables in Part (c).

(i) Write out the model in equation form.

```
[29] X = df[['age', 'nox']]
     y = df['crim']
     X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.2,random_state=0)
     logreg = LogisticRegression()
     logreg.fit(X_train,y_train)
     y_pred=logreg.predict(X_test)
     X2=sm.add_constant(X_train)
     est=sm.OLS(y_train,X2)
     est2=est.fit()
     est2.summary()
```

OLS Regression Results

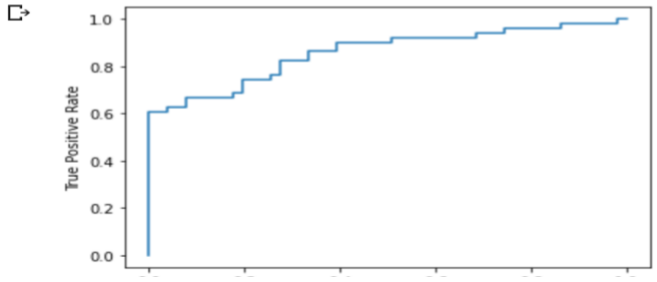
Dep. Variable:	crim	**R-squared:**	0.555
Model:	OLS	**Adj. R-squared:**	0.553
Method:	Least Squares	**F-statistic:**	249.9
Date:	Sun, 21 Nov 2021	**Prob (F-statistic):**	3.33e-71
Time:	00:45:51	**Log-Likelihood:**	-129.72
No. Observations:	404	**AIC:**	265.4
Df Residuals:	401	**BIC:**	277.5
Df Model:	2		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
const	-1.1514	0.087	-13.251	0.000	-1.322	-0.981
age	0.0037	0.001	4.147	0.000	0.002	0.005
nox	2.5139	0.216	11.618	0.000	2.089	2.939

Omnibus:	3.405	**Durbin-Watson:**	2.054
Prob(Omnibus):	0.182	**Jarque-Bera (JB):**	2.746
Skew:	0.083	**Prob(JB):**	0.253
Kurtosis:	2.632	**Cond. No.**	1.03e+03

(ii) Draw the ROC curve. What is the AUC?

```
y_pred_proba = logreg.predict_proba(X_test)[::,1]
fpr, tpr, _ = metrics.roc_curve(y_test,  y_pred_proba)
auc = metrics.roc_auc_score(y_test, y_pred_proba)
#create ROC curve
plt.plot(fpr,tpr)
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.show()
auc
```



auc: 0.857362

(iii) Determine the threshold for which (TP rate + (1-FP rate)) is maximal. Then, compute the confusion matrix, accuracy, balanced accuracy, FP, and FP rates for that threshold.

```
threshold = 0.7
y_pred = (logreg.predict_proba(X_test)[:, 1] > threshold).astype('float')
confusion_matrix(y_test, y_pred)
from sklearn.metrics import balanced_accuracy_score
ba = balanced_accuracy_score(y_test, y_pred)
from sklearn.metrics import accuracy_score
a = accuracy_score(y_test, y_pred)
FP = cnf_matrix.sum(axis=0) - np.diag(cnf_matrix)
FN = cnf_matrix.sum(axis=1) - np.diag(cnf_matrix)
TP = np.diag(cnf_matrix)
TN = cnf_matrix.sum() - (FP + FN + TP)
FP = FP.astype(float)
FN = FN.astype(float)
TP = TP.astype(float)
TN = TN.astype(float)

# Sensitivity, hit rate, recall, or true positive rate
TPR = TP/(TP+FN)
# Specificity or true negative rate
TNR = TN/(TN+FP)
```

```
 [0.84313725 0.82352941]
 [0.82352941 0.84313725]
 Balanaced Accuracy: 0.7941176470588235
 Accuracy: 0.7941176470588235
```

4. Suppose we produce ten bootstrapped samples from a data set containing red and green classes. We then apply a classification tree to each bootstrapped sample and, for a specific value of X, produce 10 estimates of P(Class is Red | X): 0.1,0.15,0.2,0.2,0.55,0.6,0.6,0.65,0.7, and 0.75.There are two common ways to combine these results together into a single class prediction. One is the majority vote approach. The second approach is to classify based on the average probability. In this example, what is the final classification under each of these two approaches?

Ans: Majority vote: 6 vs 4 => red
Average probability: P(Class is Red | X) = 4.5/10= 0.45

5. This problem involves the Carseats data set. We want to predict Sales using regression trees. Split the data set into a training set (75%) and a test set (25%)

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

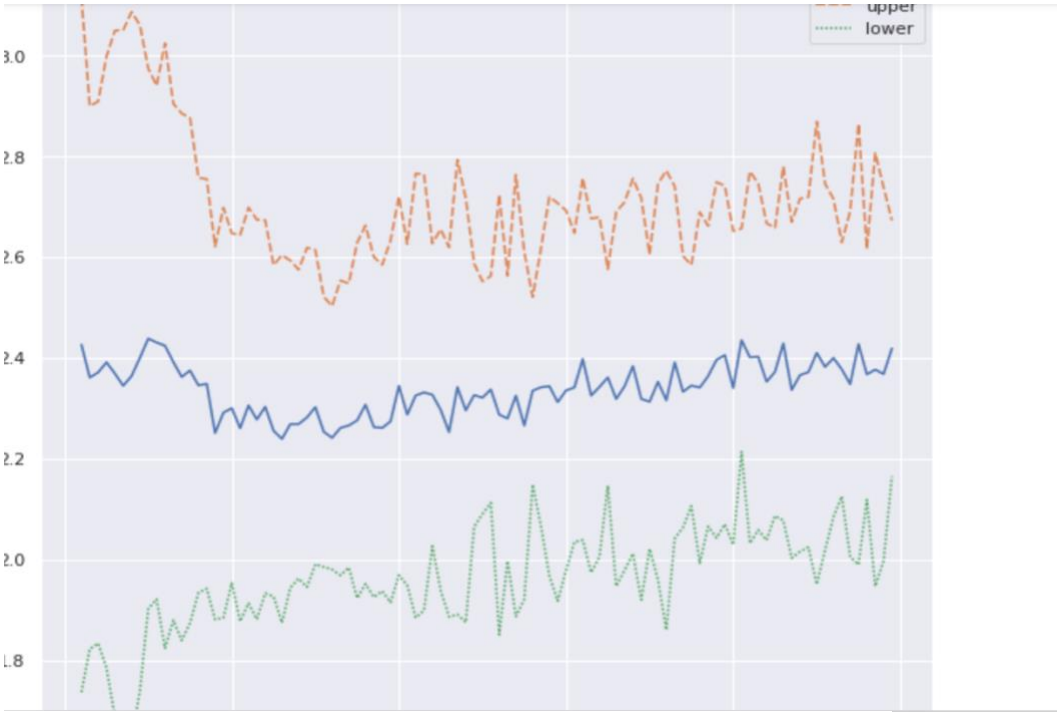
```
[131] carseats_df = pd.read_csv('https://raw.githubusercontent.com/JWarmenhoven/ISLR-python/master/Notebooks/Data/Carseats.csv')

      # Check for missing values
      assert carseats_df.isnull().sum().sum() == 0
      # Drop unused index
      carseats_df = carseats_df.drop('Unnamed: 0', axis=1)

      # Create index for training set
      np.random.seed(1)
      train = np.random.random(len(carseats_df)) > 0.5
```

```
[133] import patsy as pt
      preds = carseats_df.columns.drop(['Sales'])
      #preds_scaled = ['standardize({})'.format(p) for p in preds]
      f = 'Sales ~ 0 +' + ' + '.join(preds)
      y, X = pt.dmatrices(f, carseats_df)
      y = y.flatten()

      # Fit Sklearn's tree regressor
      clf = tree.DecisionTreeRegressor(max_depth=5).fit(X[train], y[train])

      # Measure test set MSE
      y_hat = clf.predict(X[~train])
      mse = metrics.mean_squared_error(y[~train], y_hat)

      # Get proportion of correct classifications on test set
      print('Test MSE: {}'.format(np.around(mse, 3)))
      print('Test RMSE: {}'.format(np.around(np.sqrt(mse), 3)))

      Test MSE: 5.09
      Test RMSE: 2.256
```

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

```
import seaborn as sns
sns.set()
cv_folds = 10
tuning_param = 'max_leaf_nodes'
columns=[tuning_param, 'RMSE', 'upper', 'lower']

results = []
for m in np.arange(2, 100):
    regr = tree.DecisionTreeRegressor(max_leaf_nodes=m)
    scores = cross_val_score(regr, X[train], y[train], cv=cv_folds, scoring='neg_mean_squared_error')
    rmses = np.sqrt(np.absolute(scores))
    rmse = np.mean(rmses)
    conf_int = np.std(rmses) *2
    results += [[m, rmse, rmse+conf_int, rmse-conf_int]]


# Plot classification accuracy for each max_depth cv result
plot_df = pd.DataFrame(np.asarray(results), columns=columns).set_index(tuning_param)
plt.figure(figsize=(10,10))
sns.lineplot(data=plot_df)
plt.ylabel('RMSE')
plt.show();
```

```
# Show chosen model
chosen = plot_df[plot_df['RMSE'] == plot_df['RMSE'].min()]
display(chosen)

# Use chosen model for test prediction
regr  = tree.DecisionTreeRegressor(max_leaf_nodes=int(chosen.index[0])).fit(X[train], y[train])
y_hat = regr.predict(X[~train])
mse = metrics.mean_squared_error(y[~train], y_hat)

# Get proportion of correct classifications on test set
print('Test MSE : {}'.format(np.around(mse, 3)))
print('Test RMSE: {}'.format(np.around(np.sqrt(mse), 3)))
```



max_leaf_nodes			
26.0	2.239325	2.603824	1.874826

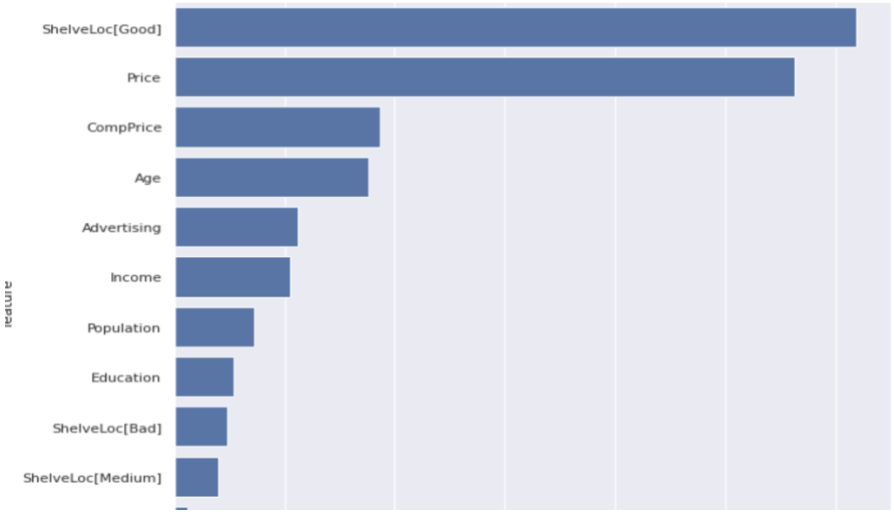
```
Test MSE : 4.716
Test RMSE: 2.172
```

10-fold cross validation selects a pruned tree model that achieves test MSE of 4.524, an improvement on the unpruned model (5.066). Interestingly 100-fold, 5-fold and 2-fold CV were all unable to select an improvement on the unpruned model.

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

```
[136] # Bagging with 100 trees
      # although I'm using RandomForestRegressor algo here this is Bagging because max_features = n_predictors

      max_features = X.shape[1]
      tree_count   = 100

      regr   = RandomForestRegressor(max_features=max_features, random_state=0, n_estimators=tree_count)
      regr.fit(X[train], y[train])
      y_hat = regr.predict(X[~train])

      mse = metrics.mean_squared_error(y[~train], y_hat)
      rmse = np.sqrt(mse)

      print('Test MSE : {}'.format(np.around(mse, 3)))
      print('Test RMSE: {}'.format(np.around(rmse, 3)))

      Test MSE : 2.611
      Test RMSE: 1.616
```

```
[137] # Plot feature by importance in this model
      plot_df = pd.DataFrame({'feature': X.design_info.column_names, 'importance': regr.feature_importances_})

      plt.figure(figsize=(10,10))
      sns.barplot(x='importance', y='feature', data=plot_df.sort_values('importance', ascending=False),
                  color='b')
      plt.xticks(rotation=90);
```



Bagging yields a significantly improved test MSE of 2.615 compared with 4.524 for the optimal pruned tree.

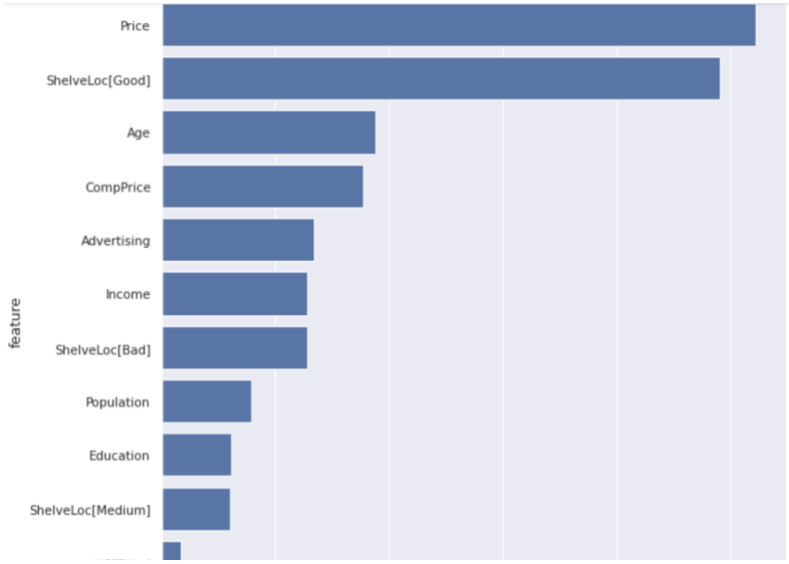
The bagging model indicates that instore Shelve Location (Good) and Price of the carseat are the most significant features affecting Sales revenue.

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

```
# Random Forest with 100 trees and 4 features considered at each split

max_features = 7
tree_count   = 100

regr   = RandomForestRegressor(max_features=max_features, random_state=0, n_estimators=tree_count)
regr.fit(X[train], y[train])
y_hat = regr.predict(X[~train])

mse = metrics.mean_squared_error(y[~train], y_hat)
rmse = np.sqrt(mse)

print('MSE test: {}'.format(np.around(mse, 3)))
print('RMSE test: {}'.format(np.around(rmse, 3)))
```

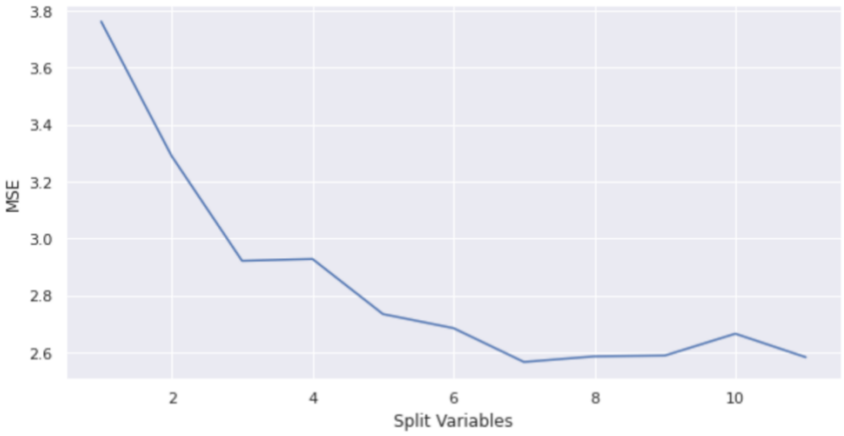
```
MSE test: 2.566
RMSE test: 1.602
```

```
[139] # Plot feature by importance in this model
plot_df = pd.DataFrame({'feature': X.design_info.column_names, 'importance': regr.feature_importances_})

plt.figure(figsize=(10,10))
sns.barplot(x='importance', y='feature', data=plot_df.sort_values('importance', ascending=False),
            color='b')
plt.xticks(rotation=90);
```



Random forest with 7 predictors at each split yields a test MSE 2.544 similar to bagging (2.615). A similar feature importance is ascribed.

```python
# Describe the effect of m, the number of variables considered at each split, on the error rate obtained.

results = []
for max_features in np.arange(1, X.shape[1]):

    tree_count   = 100

    regr    = RandomForestRegressor(max_features=max_features, random_state=0, n_estimators=tree_count)
    regr.fit(X[train], y[train])
    y_hat = regr.predict(X[~train])

    mse = metrics.mean_squared_error(y[~train], y_hat)
    rmse = np.sqrt(mse)

    results += [[max_features, mse]]

plt.figure(figsize=(10,5))
sns.lineplot(x='Split Variables', y='MSE', data=pd.DataFrame(results, columns=['Split Variables', 'MSE']));
```

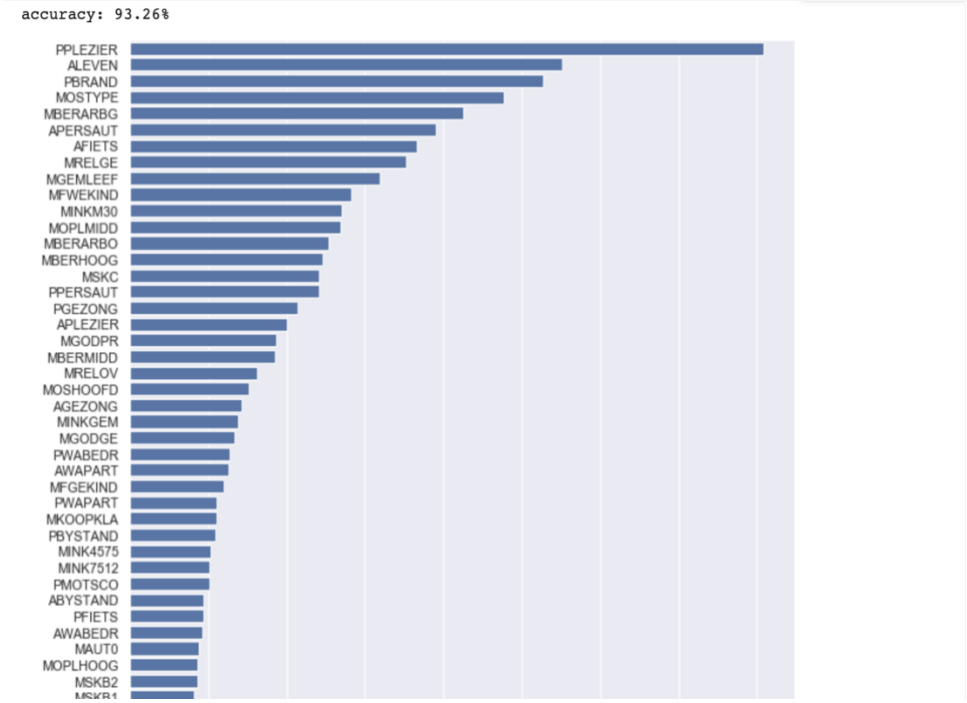


6. This question uses the Caravan data set. Create a training set consisting of the first 1,000 observations, and a test set consisting of the remaining observations.

(a) 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?

```
caravan_df = pd.read_csv('./data/Caravan.csv').drop('Unnamed: 0', axis=1)

# Patsy feature processing
f = 'C(Purchase) ~ ' + ' + '.join(caravan_df.columns.drop(['Purchase']))
y, X = pt.dmatrices(f, caravan_df)
y = y[:, 1]

# Display processed features
display(pd.DataFrame(X, columns=X.design_info.column_names).head())

# Index for Training set of 1000
np.random.seed(1)
train_sample = np.random.choice(np.arange(len(caravan_df)), size=1000, replace=False)
train = np.asarray([(i in train_sample) for i in caravan_df.index])
max_features = 'auto'
tree_count   = 1000
learning_rate = 0.01

model = GradientBoostingClassifier(max_features=max_features,
                                   random_state=1,
                                   n_estimators=tree_count,
                                   learning_rate=learning_rate)

model = model.fit(X[train], y[train])
#y_hat_test  = regr.predict(X[~train])

accuracy = model.score(X[~train], y[~train])
print('accuracy: {}%'.format(np.around(accuracy*100, 2)))

# Plot feature by importance in this model

plot_df = pd.DataFrame({'feature': X.design_info.column_names, 'importance': model.feature_importances_})

plt.figure(figsize=(10,20))
sns.barplot(x='importance', y='feature', data=plot_df.sort_values('importance', ascending=False),
            color='b')
plt.xticks(rotation=90);
```

```
accuracy: 93.26%
```



Here we use boosting to predict if someone purchases caravan insurance, a classification problem. The boosting model yields a test set prediction accuracy of 93.2%.

The model suggests that the ten most important predictors are:

- PPLEIZER: Contribution boat policies
- ALEVAN: Number of life insurances
- PBRAND: Contribution fire policies
- MOSTYPE: Customer Subtype; see L0
- MBERARBG: Skilled labourers
- APERSAUT: Number of car policies
- AFIETS: Number of bicycle policies
- MRELGE: Married
- MGEMLEEF: Avg age
- MFWEKIND: Household with children

Broadly these predictors indicate whether customer has other insurance policies, and their level of family commitment e.g. married, with children.

We can't tell the direction in which these predictors are related with response

(b) 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 logistic regression to this data set?

```
max_features = 'auto'
tree_count   = 1000
learning_rate = 0.01

model = GradientBoostingClassifier(max_features=max_features,
                                   random_state=1,
                                   n_estimators=tree_count,
                                   learning_rate=learning_rate)

model = model.fit(X[train], y[train])
#y_hat_test  = regr.predict(X[~train])


# Boosting stats
threshold = 0.2
y_hat_proba = model.predict_proba(X[~train])
y_hat = (y_hat_proba[:, 1] > threshold).astype(np.float64)
confusion_mat = confusion_matrix(y[~train], y_hat)

# What fraction of the people predicted to make a purchase do in fact make one?
pos_pred_val = np.around(confusion_mat[:, 1][1] / np.sum(confusion_mat[:, 1]), 5)

display(HTML('<h4>BOOSTING: Confusion matrix</h4>'))
print(confusion_mat)

print('\nPositive Predictive Value: {}'.format(pos_pred_val))
```

BOOSTING: Confusion matrix

```
[[4405  120]
 [ 266   31]]

Positive Predictive Value: 0.2053
```

```
# KNN

# PREDICT
for K in range(1, 10):
    # model
    model = KNeighborsClassifier(n_neighbors=K).fit(X[train], y[train])
    # Predict
    y_pred = model.predict(X[~train])

    # Confusion table
    display(HTML('<h3>K={}</h3>'.format(K)))
    confusion_mtx = confusion_matrix(y[~train], y_pred)
    print(confusion_mtx)

    ## Classifier stats
    pos_pred_val = np.around(confusion_mtx[:, 1][1] / np.sum(confusion_mtx[:, 1]), 5)
    print('\nPositive Predictive Value: {}'.format(pos_pred_val))
```

K=1

```
[[4280  245]
 [ 274   23]]

Positive Predictive Value: 0.08582
```

K=2

```
[[4507   18]
 [ 293    4]]

Positive Predictive Value: 0.18182
```

K=3

```
[[4476   49]
 [ 288    9]]

Positive Predictive Value: 0.15517
```

K=4

```
[[4524    1]
 [ 297    0]]

Positive Predictive Value: 0.0
```

K=5

```
[[4522    3]
 [ 297    0]]

Positive Predictive Value: 0.0
```

K=6

```
[[4525    0]
 [ 297    0]]
```

K=6

```
[[4525    0]
 [ 297    0]]

Positive Predictive Value: nan
```

K=7

```
[[4525    0]
 [ 297    0]]

Positive Predictive Value: nan
```

K=8

```
[[4525    0]
 [ 297    0]]

Positive Predictive Value: nan
```

K=9

```
[[4525    0]
 [ 297    0]]

Positive Predictive Value: nan
```

KNN performs best when k=2 achieving a Positive Predictive Value of 0.182, which is slightly worse but not dissimilar to the boosting result (0.205). The boosting model also excels in achieving roughly 8 times more True Positive predictions, so we would certainly choose the boosting model in this case.

It is worth noting that the dataset in this setting is highly dimensional with over 80 predictors, so the effectiveness of KNN is likely hindered by the curse of dimensionality